z/OS

Distributed File Service zSeries File System Administration

SC24-5989-02

z/OS

Distributed File Service zSeries File System Administration

SC24-5989-02

Note:

Before using this information and the product it supports, read the information in Appendix B, “Notices” on page 197.

Third Edition (September 2002)

This is a major revision of SC24-5989-01.

This edition applies to Version 1 Release 4 of z/OS™ (5694-A01) and Version 1 Release 4 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Order documents through your IBM® representative or the IBM branch office serving your locality. Documents are not stocked at the address given below.

IBM welcomes your comments. You may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States customers only): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
- Title and order number of this document
- Page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

About this document

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

How this document is organized

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

- Part 1, "zFS administration guide" on page 1 discusses guidance information for the zSeries File System (zFS).
- Part 2, "zFS administration reference" on page 47 discusses the zSeries File System (zFS) reference information which includes z/OS system commands, zFS commands, and zFS data sets.

Conventions used in this document

This document uses the following typographic conventions

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

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

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

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

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

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

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

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

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

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

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

This document used the following keying convention:

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

Entering commands
When instructed to enter a command, type the command name and then press **<Return>**.

Where to find more information

Where necessary, this document references information in other documents, For complete titles and order numbers for all elements of z/OS, refer to the*z/OS: Information Roadmap*, SA22-7500.

For information about installing Distributed File Service components, refer to the *z/OS: Program Directory*, GI10-0669.

Information concerning Distributed File Service zSeries File System-related messages can be found in the *z/OS: Distributed File Service Messages and Codes* document.

Softcopy publications

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

Internet sources

The softcopy z/OS publications are also available for web-browsing and for viewing or printing PDFs using the following URL:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv`

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

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for most messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX Systems Services running OMVS). You can also download code from the *z/OS:Collection*, SK3T-4269 and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS:Collection* , SK3T-4269 or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

Accessing licensed documents on the web

z/OS licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

`http://www.ibm.com/servers/resourcelink`

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code.

To obtain your IBM Resource Link user ID and password, log on to:

`http://www.ibm.com/servers/resourcelink`

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM**or IBM Resource Link to print licensed documents.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this document, send your comments by using Resource Link at http://www.ibm.com/servers/resourcelink. Select Feedback on the Navigation bar on the left. Be sure to include the name of the document, the form number of the document, the version of the document, if applicable, and the specific location of the text you are commenting on (for example, a page number or table number.)

Summary of Changes

Summary of changes
for SC24-5989-02
z/OS Version 1 Release 4

This document contains information previously presented in SC24-5989-01, which supports z/OS Version 1 Release 3.

New information

- Configuration changes can be made to zFS without stopping and restarting zFS.
- A zFS aggregate can be dynamically extended.
- Different zFS file systems can have the same name if they are in different multi-file system aggregates.
- System symbols can be used in the IOEFSPRM file.
- There are several new pfsctl APIs.
- Information is added to indicate this document supports z/OS.e.

Changed information

- Several zfsadm commands (and pfsctl APIs) allow you to specify the USS file system name (the mount file system name) instead of the zFS file system name.

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

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this document— for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of changes
for SC24-5989-01
z/OS Version 1 Release 3

This document contains information previously presented in SC24-5989-00, which supports z/OS Version 1 Release 2.

New information

- zFS file systems support UNIX ACLs.
- An appendix with z/OS product accessibility information has been added.

Changed information

- MOUNT commands for zFS file systems can now be placed in BPXPRMxx.

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

Summary of changes
for SC24-5989-00
z/OS Version 1 Release 2
updated December 2001 (web only)

This document contains information previously presented in SC24-5989-00, which supports z/OS Version 1 Release 2.

New information

- Added new chapter for APIs, Chapter 13, “zFS application programming interfaces” on page 113.
- **-size** can now be specified on the **ioeagfmt** utility.

Changed information

- Commands that required UID 0 can now be issued without UID 0 if the user has READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

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

Part 1. zFS administration guide

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

Chapter 1. zSeries File System (zFS) overview

The z/OS Distributed File Service zSeries File System (zFS) is a z/OS UNIX file system that can be used in addition to the Hierarchical File System (HFS). zFS file systems contain files and directories (including Access Control Lists (ACLs)) that can be accessed with the z/OS hierarchical file system file Application Programming Interfaces (APIs). 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, NFS, etc.).

Note: Refer to the *z/OS: UNIX System Services Planning*, GA22-7800, for more information on ACLs.

zFS does not replace HFS, rather zFS is complimentary to HFS. HFS is still required for z/OS installation and the root file system must be HFS.

Features

zFS provides many features and benefits:

Performance zFS provides significant performance gains in many customer environments accessing files approaching 8K in size that are frequently accessed and updated. The access performance of smaller files is equivalent to HFS.

Restart zFS provides a reduced 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.

Space Sharing
As an optional function, zFS allows the administrator to define multiple zFS file systems in a single data set. This allows space that becomes available from erasing files in one file system to be available to other file systems in the same data set.

Note: This function is currently only available in non-sysplex sharing environments.

Cloning As an optional function, zFS allows the administrator to make a read-only clone of a file system in the same data set. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system. The clone operation happens relatively quickly and does not take up too much additional space because only the metadata[1] is copied.

Note: This function is currently only available in non-sysplex sharing environments.

Terminology

In order to discuss the details of zFS administration, a new concept and some new terminology is introduced. The new concept is multiple file systems in a single data set. This is in contrast to HFS which always has a single file system per data set.

The data set that contains zFS file systems is called a ***zFS aggregate***. A zFS aggregate can contain one[2] or more zFS file systems. A zFS aggregate is a Virtual Storage Access Method Linear Data Set (VSAM LDS). Once the zFS aggregate is defined and formatted, one or more zFS file systems can be created in the aggregate. A zFS aggregate that contains only a single zFS file system can be defined and is called a compatibility mode aggregate. Compatibility mode aggregates are more like HFS. It is recommended that

1. Metadata consists of things like owner, permissions and data block pointers.
2. Actually, a zFS aggregate can contain zero or more zFS file systems.

as you begin to use zFS, you use compatibility mode aggregates. Later, if appropriate, you can use aggregates that contain multiple file systems. These aggregates are called multi-file system aggregates.

The term ***zFS file system*** refers to a hierarchical organization of files and directories that has a root directory and can be mounted into the z/OS UNIX hierarchy. zFS file systems reside on DASD. The term ***zFS Physical File System (PFS)*** refers to the code that runs in the zFS address space. The zFS PFS can handle many users accessing many zFS file systems at the same time.

When discussing cloning, the zFS file system that is the source file system is referred to as the **read-write file system**. The zFS file system that is the result of the clone operation is called the **backup file system**. The backup file system is a read-only file system and can only be mounted as read-only.

When discussing file systems, the term **zFS file system name** refers to the name of the file system as zFS knows it. The term **USS file system name** or **mount file system name** refers to the name of the file system as USS knows it. We make this distinction because you can now specify a USS file system name (as specified in the MOUNT FILESYSTEM option) that is different from the zFS file system name (as optionally specified in the MOUNT PARM FILESYSTEM suboption). This latter specification may be required when working with multiple zFS file systems that have the same zFS file system name (in different zFS aggregates).

Chapter 2. Post installation processing

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

Note: If you are only using the zFS support of the Distributed File Service (and not the DCE DFS support nor the SMB server support of the Distributed File Service), DCE DFS and SMB do not need to be configured and DCE does not need to be configured. For more information on DCE DFS, refer to the *z/OS: Distributed File Service DFS Administration* document. For more information on SMB, refer to the *z/OS: Distributed File Service SMB Administration* document.

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

- Define the zFS physical file system to z/OS UNIX
- Create or update the zFS parameter data set (IOEFSPRM)
- Define zFS aggregates and file systems
- Create mount points and mount zFS file systems
- Change owner/group and set permissions on file system root
- Add **/usr/sbin/mount** commands to **/etc/rc** for all added file systems so that they are mounted when the system is re-IPLed.

| If you are using z/OS Version 1 Release 3 or later (and all your systems in the sysplex are running
| z/OS Version1 Release 3 or later), you can put MOUNT statements in your BPXPRMxx member(s) to
| cause zFS file systems to be mounted at IPL.

zFS installation and configuration steps

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

1. Install and perform post-installation of the Distributed File Service by following the applicable instructions in the *z/OS: Program Directory*, GI10-0669 or the *ServerPac: Installing Your Order*.
   a. Ensure that the target and distribution libraries for the Distributed File Service are available.
   b. Run the prefix.SIOESAMP(IOEISMKD) job from UID 0 to create the symbolic links used by the Distributed File Service. This job reads the member prefix.SIOESAMP(IOEMKDIR) to delete and create the symbolic links.
   c. Ensure that the DDDEFS for the Distributed File Service are defined by running the prefix.SIOESAMP(IOEISDDD) job.
   d. Install the Load Library for the Distributed File Service. The Load Library (hlq.SIOELMOD) must be APF authorized and must be in link list.
   e. Install the samples (hlq.SIOEASAMP).
   f. Install the sample PROC for ZFS (hlq.SIOEPROC).
   g. Create a JCL PROC for the ZFS Started Task in SYS1.PROCLIB by copying the sample PROC from the previous step.

      The DDNAME IOEZPRM identifies the optional IOEFSPRM data set. Although this DD statement is optional, it is recommended that it be included to identify the parameter data set to be used for ZFS. For now, it is suggested that this DD refer to a PDS with a member called IOEFSPRM that has a single line that begins with an asterisk (*) in column 1. Subsequent modifications can be made to the IOEFSPRM member, refer to “IOEFSPRM” on page 106.

      | If you want to run ZFS so that it is not under control of JES, see step 2 below. You might want to
      | do this so that ZFS does not interfere with shutting down JES. In that case, if you are sharing the
      | ZFS PROC among members of a sysplex and you are running one or more members at a release

prior to z/OS Version 1 Release 3, you must include several additional DD statements in the ZFS PROC. This will avoid having LE attempt to dynamically allocate these to JES spool. They are:

```
SYSIN    DD DUMMY
SYSPRINT DD DUMMY
SYSOUT   DD DUMMY
CEEDUMP  DD DUMMY
```

These DD's can be included in systems running z/OS Version 1 Release 3 and later.

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(CREATE)
 UACC(NONE)
RDEFINE STARTED DFS.** STDATA(USER(DFS))
RDEFINE STARTED DFSCM.** STDATA(USER(DFS))
RDEFINE STARTED ZFS.** STDATA(USER(DFS))
SETROPTS RACLIST(STARTED)
SETROPTS RACLIST(STARTED) REFRESH
```

Note: The DFS user ID must have at least ALTER authority to all VSAM LDSes 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.

2. Create a BPXPRMxx entry for ZFS.

   Add the following FILESYSTYPE statement to your BPXPRMxx:

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

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

   You can specify that ZFS should not run under control of JES by specifying SUB=MSTR as in the following example:

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

   See step 1g above for information about DD's that may need to be included in the ZFS PROC.

3. Run the **dfs_cpfiles** program.

   Running this program as described in the program directory is recommended even if you plan to only use the zFS support. The only zFS configuration file is the **IOEFSPRM** data set and it is not created by the **dfs_cpfiles** program. But, to complete the installation of the Distributed File Service, the **dfs_cpfiles** program should be run to create other files needed by SMB or DCE DFS support. This avoids problems if the other support (SMB or DCE DFS) supplied by the Distributed File Service is subsequently activated.

   To run the **dfs_cpfiles** program:

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

4. Create or update the zFS parameter data set (IOEFSPRM).

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

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

If you are running a sysplex, you may want to have different IOEFSPRM data sets for different systems. Refer to Chapter 5, “Sysplex considerations” on page 15 for reasons why you may need to use different **IOEFSPRM** data sets. In this case, you may want to specify a system qualifier in the data set name in the IOEZPRM DD. For example:

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

The PDS (organization PO) should have a record format of FB with a record length of 80. The block size can be any multiple of 80 that is appropriate for the device. A sample **IOEFSPRM** is provided in hlq.SIOESAMP(IOEFSPRM). Refer to Chapter 12, “zFS data sets” on page 105 for a full description of the options that can be specified in **IOEFSPRM**.

5. 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. Refer to Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates” on page 11 for details on creating zFS file systems.

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

   A directory can be created with the OMVS **mkdir** command. (You can also use an existing directory.) The TSO/E **MOUNT** command or the OMVS **/usr/sbin/mount** command can be used to mount the zFS file system on the directory. Refer to Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates” on page 11 for details on mounting zFS file systems.

   Note: Steps 5 and 6 can be repeated as many times as necessary for each permanently mounted zFS file system. Only step 5 is needed for zFS automounted file systems (assuming that the automount file system has been set up.)

7. Add entries so that the zFS file systems are mounted automatically on the next IPL.

   In z/OS Version 1 Release 3, MOUNT commands for zFS file systems can be placed in BPXPRMxx members. For example:

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

   Prior releases did not allow MOUNT commands for zFS file systems in BPXPRMxx members and recommended that you place **/usr/sbin/mount** commands in **/etc/rc** to cause zFS file systems to be mounted during IPL. For example:

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

   If you have placed **/usr/sbin/mount** commands in **/etc/rc**, they can be left there. Or, in systems that are running z/OS Version 1 Release 3, they can be moved to BPXPRMxx members. MOUNT commands for zFS file systems cannot be placed in BPXPRMxx members that are shared with systems that are running a level of z/OS less than Version 1 Release 3.

Chapter 3. Managing zFS processes

This chapter describes the zFS address space and then discusses starting zFS, stopping zFS, and other activities required to manage zFS.

zFS runs as a z/OS UNIX colony address space. There must be an entry in a BPXPRMxx parmlib member for ZFS and the ZFS PROC must be available. zFS is started by z/OS UNIX based on the FILESYSTYPE statement for ZFS in the BPXPRMxx parmlib member.

zFS can be started at IPL if the BPXPRMxx parmlib member is in the IEASYSxx parmlib member's OMVS=(xx,yy) list. It can also be started later by using the **setomvs reset=(xx)** operator command.

zFS can be stopped using the **p zfs** operator command. When zFS is stopped, you receive the following message:

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

To restart zFS, reply **r** to message nn. (For example, **r 1,r**). If you want zFS to remain stopped, you can reply **i** and this removes the prompt. In this case, zFS may be restarted at a later time using the **setomvs reset=(xx)** operator command.

Note: Stopping zFS may have shared HFS (sysplex) implications. Refer to Chapter 5, "Sysplex considerations" on page 15 for information on shared HFS and zFS.

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

This chapter discusses creating compatibility mode aggregates and file systems. Refer to Chapter 8, “Multi-file system aggregates” on page 25 for information on multi-file system aggregates.

Creating a compatibility mode aggregate

A zFS file system is created in a zFS aggregate (which is a VSAM Linear Data Set). When using compatibility mode aggregates, the aggregate and the file system are created at the same time. For simplicity, we refer to a file system in a compatibility mode aggregate as a compatibility mode file system. A compatibility mode file system is created using the **IOEAGFMT** utility. This is a two step process:

1. Create a VSAM Linear Data Set using **IDCAMS**.
2. Format the VSAM LDS as a compatibility mode aggregate and create a file system in the aggregate using **IOEAGFMT**.

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

Figure 1 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(2))
/*
//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 1. Job to create a compatibility mode file system

Note, the **-compat** parameter in the CREATE step. That is what tells **IOEAGFMT** to create a compatibility mode file system. The result of this job is a VSAM LDS that is formatted as a zFS aggregate and contains one zFS file system. The zFS file system has the same name as the zFS aggregate (and the VSAM LDS). The size of the zFS file system (that is, its quota) is based on the size of the aggregate.

The default for the size of the aggregate is the number of 8K blocks that fits in the primary allocation. You can specify a **-size** option giving the number of 8K blocks for the aggregate. If you specify a number that is less than (or equal to) the number of blocks that fits into the primary allocation, the primary allocation size is used. If you specify a number that is larger than the number of 8K blocks that fits into the primary allocation, the VSAM LDS is extended to the size specified. This occurs during its initial formatting. Sufficient space must be available on the volume(s). Multiple volumes may be specified on the DEFINE of the VSAM LDS. DFSMS decides when to allocate on these volumes during extension. VSAM LDSes

greater then 4 GB may be specified by using the extended format and extended addressability capability in the data class of the data set. Refer to *z/OS: DFSMS: Using Data Sets*, SC26-7410, for information on VSAM data sets greater than 4 GB in size.

There are several other options that can be used 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. Refer to Chapter 11, “zFS commands” on page 55 for more information on **IOEAGFMT**.

The zFS file system can now be mounted into the z/OS UNIX hierarchy. This is accomplished with the TSO/E **MOUNT** command. Here is an example of mounting the compatibility mode file system that was just created:

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

This assumes that the directory `/usr/mountpt1` exists and is available to become a `mountpoint`. Note that the `TYPE` parameter of the **MOUNT** command specifies ZFS. This is required for any zFS file system. All other forms of the mount function are also supported (for example, the **/usr/sbin/mount** command, the automount facility, etc.). Once the zFS file system is mounted, applications and commands can be executed and files and directories can be accessed in zFS just as in HFS.

Here is an example of mounting the compatibility mode file system that was just created using the OMVS **mount** command:

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

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(s) to extend the aggregate's VSAM Linear Data Set. The size specified on the **zfsadm grow** command must be larger than the current size of the aggregate.

For example, suppose a 2 cylinder (primary allocation, 3390) aggregate has a total of 179 8K blocks and a (potential) secondary allocation of 1 cylinder. 179 8K blocks is 1432K bytes. A **zfsadm aggrinfo** command for this aggregate might show 1296K with 136K reserved. This is a total of 1432K. A **zfsadm grow** command would need to specify a size greater than 1432 to actually grow the aggregate. **zfsadm grow** does this by calling DFSMS to allocate the additional DASD space. You may need to specify a few blocks larger than the current size before an allocation occurs because DFSMS may require some number of reserved blocks. For example, you may need to specify a size of 1441 before the extension actually occurs. Refer to the following example:

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

OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 1150 K free out of total 1296 (136 reserved)
zfsadm grow omvs.prv.aggr003.lds0003 1440

IOEZ00175E Error growing aggregate OMVS.PRV.AGGR003.LDS0003, error code=8 reason code=EF17625D
zfsadm grow omvs.prv.aggr003.lds0003 1441

IOEZ00173I Aggregate OMVS.PRV.AGGR003.LDS0003 successfully grown
OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 1798 K free out of total 1944 (208 reserved)
```

The aggregate now has a total size of 2152K bytes (1944 + 208). You can specify 0 for the size to get a secondary allocation size extension. The file system quota has also been increased based on the new aggregate size. Aggregates cannot be made smaller.

Dynamically growing a compatibility mode aggregate

An administrator can specify that an aggregate should be dynamically grown if it becomes full. This is specified by the AGGRGROW PARM on the MOUNT command or globally by the aggrgrow option of the IOEFSPRM file. The aggregate (that is, the VSAM Linear Data Set) must have secondary allocation specified when it is defined and space must be available on the volume(s). The aggregate will be extended when an operation cannot complete because the aggregate is full. If the extension is successful, the operation will be redriven transparently to the application.

Creating a multi-volume compatibility mode aggregate

To create a large zFS aggregate (for example, 10 full volumes), you need:

- 10 empty volumes, and
- a DFSMS DATACLASS that provides extended addressability (since the total size is greater than 4 GB), and
- a JOB that defines and formats the aggregate.

Assuming that each volume is a 3390 with 3338 cylinders (with 3336 cylinders free), that there are 15 tracks per cylinder and that you can get 6 8K blocks per track (15 x 6 = 90 8K blocks per cylinder), you should get 90 x 3336 = 300240 8K blocks per volume and 10 x 300240 = 3002400 8K blocks in the aggregate. Figure 2 is an example JOB that defines the VSAM Linear Data Set in the first step and formats it as a zFS aggregate in the second step. The FORMAT step formats the primary allocation (3336 cylinders, and then extends the data set by the -grow amount (300240 8K blocks) multiple times until it reaches the total -size amount (3002400 8K blocks).

```
//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 5) SHAREOPTIONS(2))
/*
//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 2. Job to create a multi-volume compatibility mode aggregate

Chapter 5. Sysplex considerations

zFS supports the ***shared HFS*** sysplex environment. That is, users in a sysplex can access zFS data that is ***owned*** by another system in the sysplex. zFS file systems are automoved on system failure. zFS file systems can be automounted. Some options may not apply. See *z/OS: UNIX System Services Planning* for information on automount. For full sysplex support, zFS must be running on all systems in the sysplex and all zFS file systems must be compatibility mode file systems (that is, they cannot be file systems in multi-file system aggregates).

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

- Only systems running zFS see zFS file systems. The file system hierarchy appears different when viewed from systems with zFS mounted file systems than it does from those systems not running zFS. Pathname traversal through zFS mountpoints have different results in such cases since the zFS file system is not mounted on those systems not running zFS.
- zFS file systems owned by another system is 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.
- File systems in multi-file system aggregates are not fully supported in a shared HFS environment. We recommend that you only use compatibility mode aggregates in a sysplex. Refer to “Multi-file system aggregates and shared HFS” on page 16 for more information about multi-file system aggregates and shared HFS.
- The **IOEFSPRM** file cannot be shared across systems in a sysplex when the file contains:
  - A multi-file system aggregate specification, or
  - A **msg_output_dsn** specification, or
  - A **trace_dsn** specification.

  However, it can be shared across systems in a sysplex if you can use system symbols to differentiate the data set names you are using in the **IOEFSPRM** file.

  In addition, you cannot share an **IOEFSPRM** file that has a **user_cache_size** specification of greater than 2048M if there are systems that are running at a level lower than z/OS Version 1 Release 3.

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

  If you are only using compatibility mode aggregates (and file systems), and you are not specifying a **msg_output_dsn** or a **trace_dsn** (or you can use system sysbols), and you use the same options for all ZFS PFSs on all systems, you can share the same **IOEFSPRM** file across systems.

  If you want to share your **IOEFSPRM** file and you want to specify data set names in the **IOEFSPRM** file, you may 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** file. You can also use system symbols in the **define_aggr** option of the **IOEFSPRM** file.

The following describes z/OS UNIX considerations that relate to the level of z/OS or OS/390® running on the members of the sysplex:

- When all members of the sysplex are at z/OS Version 1 Release 2 and some or all systems are running zFS:

  - All systems running zFS see zFS file systems. The file system hierarchy appears differently when viewed from systems with zFS mounted file systems than it does from those systems not running zFS. Pathname traversal through zFS mountpoints have different results in such cases since the zFS file system is not mounted on those systems not running zFS.
  - If a system running zFS is brought down,
    - zFS compatibility mode file systems owned by the system that can be automoved are automoved to another system running zFS. If this function fails to find another owner, the file system becomes unowned.
    - zFS file systems that are noautomove, become unowned.
    - File systems which are unowned are not visible in the file system hierarchy, but can be seen from a **D OMVS,F** operator command. To recover a file system that is mounted and unowned, the file system must be unmounted.
  - If zFS is brought down on one system in the sysplex,
    - zFS compatibility mode file systems owned by the system that can be automoved are automoved to another system running zFS. If this function fails to find another owner, the file system and all file systems mounted under it are unmounted in the sysplex.
    - zFS file systems that are noautomove and all file systems mounted under them are unmounted in the sysplex.
- When all members of the sysplex are **not** at z/OS Version 1 Release 2 and some or all systems are running zFS:
  - Only systems running zFS see zFS file systems. The file system hierarchy appears differently when viewed from systems with zFS mounted file systems than it does from those systems not running zFS. Pathname traversal through zFS mountpoints have different results in such cases since the zFS file system is not mounted on those systems not running zFS.
  - If a system running zFS is brought down:
    - zFS compatibility mode file systems owned by the system that can be automoved are automoved to another system running zFS. If this function fails to find another owner, the file system becomes unowned.
    - zFS file systems owned by the system that are noautomove become unowned.
    - File systems which are unowned are not visible in the file system hierarchy, but can be seen from a **D OMVS,F** operator command. To recover a file system that is mounted and unowned, the file system must be unmounted.
  - If zFS is brought down on a system:
    - All zFS file systems owned by any of the systems unmount even if this zFS does not own any zFS file system.

      Note: This means that terminating zFS on any one system causes all zFS file systems in the sysplex to be unmounted. Because of this, it is not recommended at this time to allow zFS file systems to be shared between systems in a shared HFS environment when all systems are not at a z/OS Version 1 Release 2 level of support or higher. In a sysplex environment where all members are not at the z/OS Version 1 Release 2 level of support, the following configuration choices are recommended:
      - Restrict zFS usage to one member of the shared HFS sysplex group, or,
      - Restrict zFS usage to images not participating in a shared HFS sysplex group.

      Alternatively, you should restrict zFS usage to a multi-system shared HFS sysplex where all systems are at the z/OS Version 1 Release 2 level of support or higher.

Multi-file system aggregates and shared HFS

Multi-file system aggregates have several restrictions and limitations in a shared HFS environment. We recommend that you only use compatibility mode aggregates in a shared HFS environment.

Note: Although the clone operation is allowed in a compatibility mode aggregate, if both file systems (the read-write and the backup file systems) are mounted, some restrictions and limitations apply to the compatibility mode aggregate because there are really two mounted file systems in the aggregate. **chmount** and **setomvs** cannot be used to move ownership. **MOUNT** (that is, the mount of the second file system from a different system) and **AUTOMOVE** (as long as you do not have incompatible AUTOMOVE SYSTEM LISTs) do work properly in z/OS Version 1 Release 4 and later.

If you want to access data that resides on a file system in a multi-file system aggregate on a sysplex, here are the known restrictions and limitations.

- All file systems in a multi-file system aggregate that is attached R/W (this can only be done on a single system) must be mounted on (that is, owned by) the same system. This also holds for a compatibility mode aggregate that has a clone. The read-write file system and the backup file system must be mounted on the same system.
- A file system in a multi-file system aggregate must be mounted **NOAUTOMOVE**. This is because z/OS UNIX is not aware of the relationship between zFS file systems in the same aggregate. The file systems cannot be moved by z/OS UNIX on a system failure. One of the file systems cannot be moved by itself. If you were to attempt to move a zFS file system (in a multi-file system aggregate) to another system (by using, for example the **chmount** command), the move would fail. This is because the (multi-file system) aggregate would need to be ″moved″ (attached) to the other system and all file systems in the aggregate would need to be moved to the other system together. This is not supported with an automatic mechanism. If you want to move ownership of a file system in a multi-file system aggregate, you must manually unmount each file system in the aggregate, detach the aggregate, attach the aggregate on the other system, and then mount all the file systems on the other system. This is a disruptive procedure to applications accessing data on those file systems.
- A zFS file system in a multi-file system aggregate should not be automounted since an automounted file system is always mounted automove. Also, automounted file systems might be mounted from (owned by) any system.
- If you stop zfs, all the file systems in the multi-file system aggregates attached to that system will be unmounted because they cannot be moved.
- When file systems are contained in a multi-file system aggregate that is to be attached R/O, the aggregate should be attached (R/O) on all systems (that are running zFS) before any mounts occur. A suggested way to do this is by using the **define_aggr** option in the **IOEFSPRM** file.

Chapter 6. Backing up zFS

This chapter describes how to back up a zFS aggregate using a DFSMSdss™ logical dump. To do this procedure, the aggregate should be attached. File systems in the aggregate may or may not be mounted. The following job consists of three steps:

1. Quiesces the aggregate (this drains any activity and suspends any new requests)
2. Backs up the aggregate (and all the file systems)
3. Unquiesces the aggregate (allowing zFS activity to continue).

The size of the target sequential data set should be at least as big as the aggregate being backed up. For the following example, this warning message is expected:

```
| ADR730W (001)-DTDSC(01), CLUSTER OMVS.PRV.AGGR004.LDS0004 IS OPEN
```

Figure 3 on page 20 shows an example of a job backing up a zFS aggregate.

```
//ZFSBKUP1 JOB (OS390),'PROGRAMMER',CLASS=A,
//             MSGCLASS=X,MSGLEVEL=(1,1)
//*-------------------------------------------------------------------
//* THIS JOB QUIESCES A ZFS AGGREGATE, DUMPS IT, THEN UNQUIESCES IT.
//*-------------------------------------------------------------------
//*-------------------------------------------------------------------
//* THIS STEP QUIESCES THE AGGREGATE.
//*-------------------------------------------------------------------
//QUIESCE  EXEC   PGM=IOEZADM,REGION=0M,
// PARM=('quiesce -aggregate OMVS.PRV.AGGR004.LDS0004')
//STEPLIB   DD    DISP=SHR,DSN=G1ZFS12.Z1222.LOAD
//*
//SYSPRINT  DD    SYSOUT=H
//STDOUT    DD    SYSOUT=H
//STDERR    DD    SYSOUT=H
//SYSUDUMP  DD    SYSOUT=H
//CEEDUMP   DD    SYSOUT=H
//*
//*-------------------------------------------------------------------
//* THIS STEP DUMPS THE AGGREGATE.
//*-------------------------------------------------------------------
//DUMP    EXEC PGM=ADRDSSU,REGION=4096K
//SYSPRINT DD  SYSOUT=*
//SYSABEND DD  SYSOUT=*
//OUT      DD  DSN=SUIMGUR.HIGHRISK.ZFS.DUMP1,
//             DISP=(NEW,CATLG,DELETE),SPACE=(CYL,(5,1),RLSE)
//SYSIN    DD  *
 DUMP DATASET(INCLUDE(OMVS.PRV.AGGR004.LDS0004)) -
 OUTDD(OUT) TOL(ENQF)
/*
/*
//*-------------------------------------------------------------------
//* THIS STEP UNQUIESCES THE AGGREGATE.
//*-------------------------------------------------------------------
//UNQUIES  EXEC   PGM=IOEZADM,REGION=0M,
// PARM=('unquiesce -aggregate OMVS.PRV.AGGR004.LDS0004')
//STEPLIB   DD    DISP=SHR,DSN=G1ZFS12.Z1222.LOAD
//*
//SYSPRINT  DD    SYSOUT=H
//STDOUT    DD    SYSOUT=H
//STDERR    DD    SYSOUT=H
//SYSUDUMP  DD    SYSOUT=H
//CEEDUMP   DD    SYSOUT=H
//*
//
```

Figure 3. Job to back up a zFS aggregate

The zFS aggregate can be restored using DFSMSdss logical restore. It is restored into a new aggregate (in this case, OMVS.PRV.AGGR005.LDS0005) if the original aggregate (in this case, OMVS.PRV.AGGR004.LDS0004) still exists. Figure 4 on page 21 is an example of a restore job.

```
//ZFSREST1 JOB (OS390),'PROGRAMMER',CLASS=A,
//             MSGCLASS=X,MSGLEVEL=(1,1)
//*-------------------------------------------------------------------
//* THIS JOB RESTORES A ZFS AGGREGATE.
//*-------------------------------------------------------------------
//*-------------------------------------------------------------------
//* THIS STEP RESTORES THE AGGREGATE.
//*-------------------------------------------------------------------
//ZFSREST EXEC PGM=ADRDSSU,REGION=0M
//SYSPRINT DD  SYSOUT=*
//SYSABEND DD  SYSOUT=*
//INDS     DD  DISP=SHR,DSN=SUIMGUR.HIGHRISK.ZFS.DUMP1
//SYSIN    DD  *
  RESTORE DATASET(INCLUDE(**)) -
          CATALOG -
          RENAMEU( -
           (OMVS.PRV.AGGR004.LDS0004, -
              OMVS.PRV.AGGR005.LDS0005) -
                 ) -
          WRITECHECK -
          INDD(INDS)
```

Figure 4. Job to restore a zFS aggregate

After the aggregate is restored, you need to do the following steps for a compatibility mode aggregate:

1. Unmount the original aggregate (in this case, OMVS.PRV.AGGR004.LDS0004) if it still exists (this also detaches it).
2. Mount the file system in the restored aggregate (in this case, OMVS.PRV.AGGR005.LDS0005).

After the aggregate is restored, you need to do the following steps for a multi-file system aggregate:

1. Unmount the file systems in the original aggregate (if any are mounted).
2. Detach the original aggregate (in this case, OMVS.PRV.AGGR004.LDS0004) if it still exists.
3. Attach the restored aggregate (in this case, OMVS.PRV.AGGR005.LDS0005).
4. Mount the file systems in the restored aggregate.

Another example of a logical restore of a zFS aggregate using DFSMSdss by replacing the existing aggregate is shown. The backup is restored into the original aggregate (in this case, OMVS.PRV.AGGR004.LDS0004). The aggregate cannot be mounted (or attached) during the restore operation. Figure 5 on page 22 is an example of a restore replace job.

```
//ZFSREST2 JOB (OS390),'PROGRAMMER',CLASS=A,
// MSGCLASS=X,MSGLEVEL=(1,1)
//*-------------------------------------------------
//* THIS JOB RESTORES A ZFS AGGREGATE.
//*-------------------------------------------------
//*-------------------------------------------------
//* THIS STEP RESTORES THE AGGREGATE.
//*-------------------------------------------------
//ZFSREST EXEC PGM=ADRDSSU,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//INDS DD DISP=SHR,DSN=SUIMGUR.HIGHRISK.ZFS.DUMP1
//SYSIN DD *
  RESTORE DATASET(INCLUDE(OMVS.PRV.AGGR004.LDS0004)) -
          CATALOG -
          REPLACE  -
          WRITECHECK -
          INDD(INDS)
```

Figure 5. Job to restore a zFS aggregate with replace

Chapter 7. Migrating data from HFS to zFS

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

Using the OMVS pax command

You can copy data from an HFS file system to a zFS file system by using the OMVS **pax** command with or without using an intermediate archive file. Refer to the *z/OS: UNIX System Services Command Reference*, SA22-7802, for more information on the **pax** command. When the data is being copied, the file system(s) being accessed must be mounted.

Using an intermediate archive file

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

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

1. Position 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 /tmp/zfs1.pax
   ```

3. Position to the target (zFS) file system mounted at **/etc/dce/testzfs1**

   ```
   cd /etc/dce/testzfs1
   ```

4. Read the archive file into the zFS file system mounted at **/etc/dce/testzfs1**

   ```
   pax -rvf -p e /tmp/zfs1.pax
   ```

Without using an intermediate archive file

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

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

1. Position to the source (HFS) file system mounted at **/etc/dfs**

   ```
   cd /etc/dfs
   ```

2. Copy the (HFS) file system mounted at /etc/dfs to the (zFS) file system mounted at **/etc/dce/testzfs1**

   ```
   pax -rwv -p e . /etc/dce/testzfs1
   ```

Chapter 8. Multi-file system aggregates

This chapter discusses multi-file system aggregates, however, it is recommended that you use compatibility mode aggregates first, refer to Chapter 1, “zSeries File System (zFS) overview” on page 3. They are more like HFS file systems. Compatibility mode aggregates have a single file system in the aggregate and are fully supported in a sysplex (shared HFS) environment. Refer to Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates” on page 11 for information on zFS compatibility mode aggregates and file systems.

If you are ready to use multi-file system aggregates, note that multi-file system aggregates are not supported in a sysplex shared HFS environment. Refer to Chapter 5, “Sysplex considerations” on page 15 for information on shared HFS and zFS file systems.

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

Creating a multi-file system aggregate

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

1. Create a VSAM LDS using IDCAMS
2. Format the VSAM LDS as a multi-file system aggregate using **ioeagfmt**.

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

Figure 6 shows an example of a job that creates and formats a zFS multi-file system aggregate.

```
//USERIDA  JOB ,'Multi-File System',
//         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.MULTI.AGGR002) -
            VOLUMES(PRV000) -
            LINEAR CYL(25 0) SHAREOPTIONS(2))
/*
//CREATE   EXEC   PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.PRV.MULTI.AGGR002')
//SYSPRINT DD     SYSOUT=H
//STDOUT   DD     SYSOUT=H
//STDERR   DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 6. Job to create a multi-file system aggregate

After the multi-file system aggregate is formatted, it has zero zFS file systems contained in it. The size of the aggregate is reported by **ioeagfmt** as the number of 8K blocks that fit into the primary allocation or as specified on the **-size** option. The multi-file system aggregate must then be attached on a system before any **zfsadm** commands can be issued against it.

The default for the size of the aggregate is the number of 8K blocks that fits in the primary allocation. You can specify a **-size** option giving the number of 8K blocks for the aggregate. If you specify a number that is less than (or equal to) the number of blocks that fits into the primary allocation, the primary allocation size is used. If you specify a number that is larger than the number of 8K blocks that will fit into the primary allocation, the VSAM LDS is extended to the size specified. (You may need to specify the **-grow** option if the extension will not fit on a single volume.) A secondary allocation on the VSAM LDS is not required. This occurs during its initial formatting. Sufficient space must be available on the volume(s). Multiple volumes may be specified on the DEFINE of the VSAM LDS. DFSMS decides when to allocate on these volumes during extension. VSAM LDSes greater then 4 GB may be specified by using the extended format and extended addressability capability in the data class of the data set. Refer to *z/OS: DFSMS: Using Data Sets*, SC26-7410, for information on VSAM data sets greater than 4 GB in size.

When you attach a multi-file system aggregate on a system, the ZFS Physical File System (PFS) must be active on that system. Refer to Chapter 3, “Managing zFS processes” on page 9 for information on starting and stopping ZFS. You can attach in one of the following ways:

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

After a new multi-file system aggregate is defined and formatted, the zFS administrator attaches it by issuing the **zfsadm attach** command and then adds a **define_aggr** statement for the aggregate in that system's **IOEFSPRM** file so that the aggregate is automatically attached each time the ZFS PFS is subsequently started (or restarted). The **define_aggr** statement does not have to be added to the **IOEFSPRM** file but then the aggregate would need to be attached (by using the **zfsadm attach** command) each time the ZFS PFS is started (or restarted).

The OMVS **zfsadm attach** command for the multi-file system aggregate just created is shown in the following example:

```
zfsadm attach -aggregate omvs.prv.multi.aggr002
```

A **define_aggr** statement in the **IOEFSPRM** file for the multi-file system aggregate just created is shown in the following example:

```
define_aggr cluster(omvs.prv.multi.aggr002)
```

Note: zFS aggregate names are case insensitive. Since they are always VSAM LDS names, they are always folded to upper case.

After the multi-file system aggregate is attached, the administrator can now create zFS file systems in the aggregate. This is accomplished using the OMVS **zfsadm create** command. The following example shows an example of creating a file system in the aggregate you just created and attached:

```
zfsadm create -filesystem OMVS.PRV.FS1 -aggregate omvs.prv.multi.aggr002 -size 5000
```

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

Note: zFS file system names are case sensitive. The file system name specified on the **zfsadm create** command is not folded to upper case. If you create a zFS file system using lower case letters, it must be mounted using these same lower case letters. This can be accomplished by using the

TSO/E **MOUNT** command and surrounding the file system name with a pair of three single quotes. Refer to the *z/OS: UNIX System Services Command Reference* for information on the TSO/E **MOUNT** command.

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

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

The quota of a file system can be displayed by using the **zfsadm lsquota** command and it can be increased by using the **zfsadm setquota** command. The quota of a file system can also be decreased (as long as the usage has not exceeded the new quota) by using the **zfsadm setquota** command. The quota is the number used when determining if a message to the operator is required due to the **FSFULL** parameter of the **MOUNT** command. Refer to “MOUNT” on page 62 for more information.

A file system’s quota can be dynamically increased if the **FSGROW PARM** was specified on the **MOUNT** or if the **fsgrow** option is specified in the **IOEFSPRM** file. You must specify the amount that the quota should grow (in k-bytes) and the number of times that the quota should be increased. (A history of the number of times the quota has been increased is not kept across instances of the ZFS PFS. That is, the number of times the quota has been increased is lost when ZFS is stopped and restarted.)

File system names must be unique across all attached aggregates on a system unless you enable this ability via the **allow_duplicate_filesystems** option in the **IOEFSPRM** file. When **allow_duplicate_filesystems=off** (the default), if you attempt to create a file system with a duplicate name, it will be denied. If you attempt to attach an aggregate that contains a duplicate file system name, the attach is successful but an error message for the duplicate file system name occurs and you are not able to use the duplicate named file system. When **allow_duplicate_filesystems=on**, you can create file systems with the same name as long as they reside in different aggregates. **zfsadm** commands that refer to file systems allow a **-aggregate** option to qualify the file system name. **MOUNT** allows an **AGGREGATE PARM**.

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

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

The previous example assumes that the directory `/etc/mountpt2` exists and is available to become a mount point. Note that the **TYPE** parameter of the **MOUNT** command specifies **ZFS**. This is required for any zFS file system. Once the zFS file system is mounted, applications and commands can be executed and files and directories can be accessed in zFS just as in HFS. Refer to Chapter 5, “Sysplex considerations” on page 15 for the reason that **NOAUTOMOVE** is specified for file systems in multi-file system aggregates.

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

Growing a multi-file system aggregate

If the sum of the quotas of all the file systems in an aggregate is greater than the physical space available in the aggregate, it is possible for a file system to run out of physical space before exceeding its quota. If this occurs, the application gets ENOSPC as a return code (the same return code it would get for exceeding its quota). The administrator can grow the aggregate (that is, cause an additional allocation to occur and format it to be part of the aggregate). This is accomplished with the **zfsadm grow** command. There must be space on the volume(s) to extend the aggregate's VSAM LDS. The size specified on the **zfsadm grow** command must be larger than the current size of the aggregate.

For example, suppose a 2 cylinder (primary allocation, 3390) aggregate has a total of 179 8K blocks and a (potential) secondary allocation of 1 cylinder. 179 8K blocks is 1432K bytes. A **zfsadm aggrinfo** command for this aggregate might show 1296K with 136K reserved. This is a total of 1432K. A **zfsadm grow** command would need to specify a size greater than 1432 to actually grow the aggregate. **zfsadm grow** does this by calling DFSMS to allocate the additional DASD space. You may need to specify a few blocks larger than the current size before an allocation occurs because DFSMS may require some number of reserved blocks. For example, you may need to specify a size of 1441 before the extension actually occurs. Refer to the following example:

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

OMVS.PRV.AGGR004.LDS0004 (R/W MULT): 1159 K free out of total 1296 (136 reserved)
zfsadm grow omvs.prv.aggr004.lds0004 1440

IOEZ00175E Error growing aggregate OMVS.PRV.AGGR004.LDS0004, error code=8 reason code=EF17625D
zfsadm grow omvs.prv.aggr004.lds0004 1441

IOEZ00173I Aggregate OMVS.PRV.AGGR004.LDS0004 successfully grown
OMVS.PRV.AGGR004.LDS0004 (R/W MULT): 1807 K free out of total 1944 (208 reserved)
```

The aggregate now has a total size of 2152K bytes (1944 + 208). The size of the aggregate is rounded up to the control area (CA) size. You can specify 0 for the size to get a secondary allocation size extension. In this case, a secondary allocation must have been specified on the VSAM LDS. File systems that have not exceeded their quota can now use the additional physical space that is available. (If necessary, a file system quota can be increased with the **zfsadm setquota** command.) Aggregates cannot be made smaller.

Dynamically growing a multi-file system aggregate

An administrator can specify that an aggregate should be dynamically grown if it becomes full. This is specified by the **-aggrgrow** option on the **zfsadm attach** command or the **aggrgrow** suboption of the **define_aggr** option of the **IOEFSPRM** file or globally by the **aggrgrow** option of the **IOEFSPRM** file. The aggregate (that is, the VSAM Linear Data Set) must have secondary allocation specified when it is defined and space must be available on the volume(s). The aggregate will be extended when an operation cannot complete because the aggregate is full. If the extension is successful, the operation will be redriven transparently to the application.

Cloning a file system

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

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

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

IOEZ00225I File system OMVS.PRV.FS1 successfully cloned.
```

A backup file system takes up a relatively small amount of space because only the metadata is copied, not the user data. The backup file system's data block pointers point to the same data blocks that the read-write file system's data block pointers point to. After a clone operation, if the read-write file system user data is updated, zFS ensures that new physical blocks are allocated to hold the updates, while maintaining the backup file system's data pointers to the original data. The backup file system remains a point-in-time read-only copy in the face of updates to the read-write file system. This backup file system can be mounted (read-only) so that users of the read-write file system can have an online backup of that file system available without administrative intervention. That is, if a user accidently erases a file from the read-write file system, they can simply copy the file from the backup into the read-write file system to restore the file to the time the backup was created.

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

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

Here is an example of an OMVS **mount** command for the backup file system:

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

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

You can clone (or reclone) a set of file systems (on a single system) with the **zfsadm clonesys** command. This can be specified in terms of file systems with a file system name prefix or file systems in an aggregate or both.

Comparing compatibility mode aggregates and multi-file system aggregates

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

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

aggregate is treated as a compatibility mode aggregate. The mount will only be successful if there is exactly one read-write file system in the aggregate. If the name of the (only) file system in the aggregate does not match the aggregate name, the file system name will be renamed so its name is the same as the aggregate name. For a mount of type RDWR, the file system name will be changed on disk. For a mount of type READ, the changed name will be kept in memory.

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

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

Chapter 9. Performance and debugging

This chapter discusses performance tuning techniques and what should be done if a problem occurs that requires IBM service assistance.

Performance tuning

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

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

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 7K or larger. For files smaller than 7K, I/O is normally performed through the metadata cache.

The user file cache is allocated in data spaces. Its size by default is 256M and can be tailored to meet your performance needs based on your overall system memory. The maximum size is 65536M (which is 64G). The general rule for any cache is to ensure a good hit ratio and additionally, for a user file cache it is good to have it large enough to allow write-behind activity to occur (if the cache is too small you need to recycle buffers more frequently and it could degrade write-behind performance). The operator **MODIFY ZFS,QUERY,ALL** command output shows the cache hit ratio. (Actually, it shows the ″Fault Ratio″. To get the hit ratio subtract the fault ratio from 100%).

In general you should have hit ratios of at least 80% or more, over 90% usually gives good performance. However, the desired hit ratio is very much workload dependent. For example, a zFS file system exported exclusively to SMB clients by using the SMB server would likely have a low hit ratio since the SMB client and the SMB server caches data, making the zFS cache achieve a low hit ratio in this case. That is expected and is not considered a problem.

NOREADAHEAD option

For sequential file access, read-ahead provides an overlap of I/O with processing that can result in smaller response time for file read requests. However for random file access, read-ahead can degrade performance. zFS generally attempts to first determine if a file's access pattern is sequential or random before it decides if read-ahead should be performed for that file. Since zFS really has no knowledge of an applications future requests sometimes zFS can make the wrong guess. For file systems that have random access patterns, which are typical for database systems such as Lotus Notes®, sequential access is rare and the administrator can disable read-ahead for that file system by specifying the **NOREADAHEAD** option for the **MOUNT** command. This ensures that zFS never performs read-ahead for any files in that file system and avoids any overhead due to unnecessary read-aheads.

Metadata cache

The metadata cache is used to contain all file system metadata which includes all directory contents, file status information (such as atime, mtime, size, permission bits, and so on), file system structures and additionally, it also caches data for files smaller than 7K. Essentially, zFS stores a file by using one of the following three methods:

inline If the file is smaller than 48 bytes, its data is stored in the structure that contains the status information for the file.

fragmented If the file is less than 7K it is stored in blocks on disk that could be shared with other files, hence multiple files are stored in the same physical disk block. Physical disk blocks are always 8K in size.

blocked Files larger than 7K 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.

Since 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). Since the metadata cache is the only component that knows about multiple files sharing the same disk blocks, small fragmented files are stored in the metadata cache (and also in the user file cache) and I/O is performed directly to or from the metadata cache for these small user files.

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

The metadata cache is stored in the primary address space and its default size is 32M. Since the metadata cache only contains metadata and small files it normally 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 and I/O rates from the metadata cache.

| An optional metadata backing cache can be specified that extends the size of the metadata cache. It
| resides in a data space and increases the amount of metadata that can be kept in memory. It may
| improve the performance of workloads that require large amounts of metadata.

Log files

Every zFS aggregate contains a log file used to record transactions describing changes to the file system structure. This log file is, by default, 1% of the aggregate size but is tailorable by the administrator on the **ioeagfmt** command. Usually, 1% is sufficient for most aggregates, especially large 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 8K buffers used to contain log file updates. Log file buffers are always written asynchronously to disk and normally only need to be waited upon when the log is becoming full, or if a file is being fsync'ed.

| The log file cache is stored in a data space and its default is 64M. The log file cache is grown dynamically
| by adding one 8K buffer for each attached aggregate. This ensures each aggregate always has one 8K
| buffer to use to record its most recent changes to file system metadata. Since log files are written
| asynchronously, the cache essentially allows write-behind of log files and since the cache is shared among
| all aggregates, aggregates that have a higher write rate use more buffers in the cache using a
| least-recently-used (LRU) algorithm.

The log file cache is a write-only cache, so a read hit ratio is not relevant, however the operator **MODIFY ZFS,QUERY,ALL** command does show log file I/O rates and I/O waits. Its desirable to make the log file cache large enough so that log file I/O waits do not occur too frequently compared to the log file I/O rates, however, every workload is different. For example, workloads that issued fsync operations force zFS to sync the log file more frequently.

Transaction cache

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

The transaction cache is stored in the zFS primary address space and its default is 2000. zFS dynamically increases the size of this cache based on the number of concurrent pending transactions (transactions that have not been fully committed to disk) in the zFS file system. Therefore, the administrator does not have to tailor the transaction cache size, but the **MODIFY ZFS,QUERY,ALL** output shows how large the transaction cache is at any given time.

Vnode cache

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

The vnode cache is stored in the zFS primary address space and the default number of vnodes is 16384. As with any cache a good hit ratio is desirable and the operator **MODIFY ZFS,QUERY,ALL** command shows the vnode cache hit ration. Since the vnode cache is essentially backed by the metadata cache, if the vnode hit ratio is low but the metadata cache hit ratio is high your performance may not suffer too much since a vnode cache miss only requires some pathlength to initialize the vnode structures.

Fixed storage

By default, zFS does not fix any 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/or the log file cache by choosing the **fixed** option for the cache. This ensures the cache experiences no paging and avoids the overhead of page fixing for each I/O but comes at the expense of using real storage for the given cache which means the real storage is not available for other applications.

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

I/O balancing

Any file system's performance is heavily dependent on DASD I/O performance. If any channel(s) or DASD volume(s) are overloaded, then excessive I/O waits could occur for that DASD.

Performance products such as RMF™ will show DASD performance.

zFS operator **MODIFY ZFS,QUERY,ALL** commands also provide reports that show I/O rates per aggregate, and file system request rates per aggregate and per file system. This information, along with DASD performance information from RMF or performance products similar to RMF can be used to easily balance I/O among your DASD. For example, the QUERY output can be used to show which file systems could be moved to different DASD to achieve a better balance among disks.

Total cache size

Currently, the ZFS address space is restricted to 2GB of total storage. Therefore, the total storage size for all of the caches that reside in the ZFS address space must be less than 2GB. Since storage is needed in addition to the ZFS address space caches to process file requests and for products zFS might use, generally you should restrict total ZFS address space cache storage to approximately 1.5GB. The zFS

operator **MODIFY ZFS,QUERY,ALL** command shows total zFS storage allocated which includes storage allocated for all the caches and everything else zFS might need. zFS terminates during initialization if it cannot obtain all the storage for the caches as directed by the **IOEFSPRM** file. The ZFS address space caches include the metadata cache, the transaction cache and the vnode cache. (The user data cache, the log file cache and the metadata backing cache reside in data spaces and do not use ZFS address space storage.)

Performance Monitoring

zFS performance can be monitored via the operator MODIFY command. The syntax of this command is:

```
MODIFY ZFS,QUERY,<report>,<option>
```

where <report> can be one of the following:

KN Provides counts of calls made to zFS from USS and the average response time of each call. This is the basic measure of zFS performance. There are no <option>s for this report.

VM This provides performance information for the user file cache including cache hit ratios, I/O rates and storage usage. There are no <option>s for this report.

LFS This provides detailed filesystem statistics including the performance of the zFS metadata caches, the vnode cache and the aggregate I/O statistics. There are no <option>s for this report.

LOCK This provides a measure of how much lock contention and how often USS threads wait for certain events such as user file cache reclaim. There are no <option>s for this report.

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

FILE This provides a detailed breakdown of requests per zFS filesystem and aggregate. By default this report lists only filesystems and aggregates that had active requests since the last statistics reset. If you use the ALL option you get all filesystem and aggregates regardless of whether they were active or not.

ALL This shows all the above reports, for the **STOR** report **DETAILS** is off and for the **FILE** report only active filesystems are shown.

You can also reset the statistics for any given zFS report or reset all of the internal zFS statistics, this is done via:

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

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

This is useful if you want zFS performance only for a given time of day, such as during peak usage. For example, if you want performance of zFS between 1PM and 3PM you could issue MODIFY ZFS,RESET,ALL at 1PM and issue MODIFY ZFS,QUERY,ALL at 3PM.

zFS query output is written to the system log and the zFS message output dataset (specified via the msg_output_dsn option in IOEFSPRM).

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

```
KN:
          zFS Kernel USS PFS Calls
          ------------------------
Operation              Count          Avg Time
---------          ----------        ----------
zfs_opens               1617             0.210
zfs_closes              1617             5.811
```

```
| zfs_reads              64153             0.216
| zfs_writes             14591             3.670
| zfs_ioctls                 1             0.027
| zfs_getattrs            1681             0.039
| zfs_setattrs               0             0.000
| zfs_accesses            5155             0.333
| zfs_lookups             1942             0.316
| zfs_creates              307            17.282
| zfs_removes              308            51.553
| zfs_links                  1             0.954
| zfs_renames               13             2.350
| zfs_mkdirs                27             6.394
| zfs_rmdirs                27            25.128
| zfs_readdirs               0             0.000
| zfs_symlinks               4             2.248
| zfs_readlinks            119             0.002
| zfs_fsyncs                 1             1.243
| zfs_truncs                 1             0.062
| zfs_lockctls               0             0.000
| zfs_audits                 0             0.000
| zfs_inactives            333             0.529
| zfs_recoverys              0             0.000
| zfs_vgets                  0             0.000
| zfs_pfsctls                0             0.000
| zfs_statfss                7             0.063
| zfs_mounts                 0             0.000
| zfs_unmounts               0             0.000
| ---------         ----------        ----------
| *TOTALS*               91905             1.108
```

This report shows basic zFS performance, it shows all calls made to zFS by USS since the last statistics reset and the average response time in milliseconds for each request. These requests are the official interface between USS and zFS and is the most fundamental measure of zFS performance because it includes any CPU, I/O wait time or lock wait time.

Its important to note that the times here are only the zFS portion of the overall command response time, for example, a mkdir command issued from OMVS 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 USS is not included here.

```
| VM:
|
|                    User File (VM) Caching System Statistics
|                    ----------------------------------------
|
| External Requests:
| ------------------
| Reads          64145     Fsyncs             1     Opens           1026
| Writes         14591     Setattrs          17     Unmaps           307
| Asy Reads      45653     Getattrs        2127     Schedules        608
|
| File System Reads:
| ------------------
| Reads Faulted           4     (Fault Ratio    0.06%)
| Writes Faulted          0     (Fault Ratio    0.00%)
| Read Waits              0     (Wait Ratio    0.00%)
| Total Reads             6
|
| File System Writes:
| -------------------
| Scheduled Writes          4595     Sync  Waits           77
| Error Writes                 0     Error Waits            0
| Scheduled deletes            0
| Page Reclaim Writes          0     Reclaim Waits          0
| Write Waits                  0     (Wait Ratio    0.00%)
|
```

```
File Management: (File struct size=168)
---------------------------------------
Max Files:           64     Allocated:          121
Lookups       1026     Hits          714 (Hit Ratio     69.590%)

Page Management (Segment Size = 64K) (Page Size = 4K)
-----------------------------------------------------
Total Pages       76800     Free              76729
Segments           4547
Steal Invocations          0     Waits for Reclaim           0

Number of dataspaces used:     5  Pages per dataspace:       15360

Dataspace   Allocated     Free
Name        Segments      Pages
--------    ----------    ----------
ZFSUCD00             2         15350
ZFSUCD01             4         15350
ZFSUCD02             2         15340
ZFSUCD03             3         15338
ZFSUCD04             3         15351
```

This report shows the performance of the zFS user file cache. This size of the cache is controlled by the **user_cache_size** zFS configuration option or the **zfsadm config** command.

The zFS user file cache data 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 currently cached, each cached file is broken into 64K segments and each segment is broken into 4K pages. A segment is assigned to a dataspace, hence the pages for any given segment belong only to one dataspace. A file's segments can be scattered throughout multiple segments.

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

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

External Requests: This section of the report describes the requests made to the user file cache to perform operations as requested by applications. Reads, Writes show how often the cache was called to read or write files. Asy Reads shows how often read-ahead is performed. Fsync shows how often applications requested that zFS sync a file's data to disk. Unmaps are the count of file deletions.

File System Reads: This section shows how often the cache had to read data from disk for a file. The ″Reads Faulted″ shows the count of read requests that needed to perform at least 1 I/O to read the requested portion of the file from disk. ″Writes Faulted″ show the count of how often a write to a file needed to perform a read from disk. If a write only updates a portion of a page of a file on disk and that page is not in memory then the page needs to be read in (the zFS I/O driver can only perform I/O in whole pages) before the new data is written to the in-memory page. Read Waits show how often a read had to wait for a pending I/O (for example, how often a read of a file found that the desired range of the file is pending read probably due to asynchronous read ahead). ″Total Reads″ is the total number of file system reads made for any reason. Cache misses and read I/Os degrade application response time, the goal is for all these numbers to be as low as possible. Increasing the cache size is the usual method for lowering these numbers.

File System Writes: This section shows how often the cache wrote data to disk. ″Scheduled Writes″ is the count of how often the cache wrote out dirty segments for a file. Segments are written as soon as every

page becomes dirty. When a file is closed all of its dirty segments are scheduled asynchronously and segments are also written asynchronously during filesystem syncs via the zFS sync daemon (which by default runs every 30 seconds).

"Sync Waits" is the count of how often an fsync request needed to wait on pending I/O for dirty segments.

"Error Writes" and "Error Waits" are error handling paths and should almost always be 0 unless a disk hardware error occurs. Whenever an unexpected error occurs for a file all of its dirty segments are written and synced to disk. (Note that a filesystem 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" is the count of times a pending I/O was cancelled due to a file being deleted. In this case the data is not desired to be on disk (because the file is 0 link count) and thus if an I/O wait can be avoided by cancelling the I/O it is done. Thus this is a performance optimization for file remove.

"Page Reclaim Writes" is the count of times that a dirty segment had to be written to reclaim space in the cache. "Page Reclaim Waits" is the count of times that the reclaim function needed to wait on pending I/O to reclaim the pages of a segment.

"Write Waits" is the count of times a write occured to a page that was already pending I/O. In this case the I/O needs to be waited upon before the page is updated with the new data.

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 (the KN report shows that) then the cache may be too small.

File Management: This section of the report shows how often the file structure representing a file was found in the cache. A miss here usually only results in some extra CPU. The default size of the file cache is the vnode_cache_size zFS parameter.

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

```
LFS:
                              zFS Vnode Op Counts

Vnode Op                      Count   Vnode Op                      Count
----------------------  ----------   ----------------------  ----------
efs_hold                         0   efs_readdir                    114
efs_rele                         0   efs_create                     307
efs_inactive                     0   efs_remove                     308
efs_getattr                   2926   efs_rename                      13
efs_setattr                     17   efs_mkdir                       27
efs_access                    6772   efs_rmdir                       27
efs_lookup                    2303   efs_link                         1
efs_getvolume                    0   efs_symlink                      4
efs_getlength                    0   efs_readlink                     5
efs_afsfid                       0   efs_rdwr                         0
efs_fid                          0   efs_fsync                        0
efs_vmread                       6   efs_waitIO                    4439
efs_vmwrite                   4595   efs_cancelIO                  4268
efs_clrsetid                     0   efs_audit                        0
efs_atime                        0   efs_vmblkinfo                  131

Total zFS Vnode Ops      26263


                          zFS Vnode Cache Statistics

Vnodes      Requests     Hits    Ratio  Allocates   Deletes
```

```
| ---------- ---------- ---------- ----- ---------- ----------
|        130        926        921 99.460%      338        318
|
| zFS Vnode structure size: 296 bytes
|
|                          Metadata Caching Statistics
|
| Buffers    (K bytes)  Requests     Hits    Ratio  Updates
| ---------- --------- ---------- ---------- ----- ----------
|       4224     33792     139453     104957 75.2%     179863
|
|                       Metadata Backing Caching Statistics
|
| Buffers    (K bytes)  Requests     Hits    Ratio  Discards
| ---------- --------- ---------- ---------- ----- ----------
|       4096     32768        340          0  0.0%          0
|
|
|                           Directory Cache Statistics
|
| Dir Blocks (K bytes)  Requests     Hits    Ratio  Deletes
| ---------- --------- ---------- ---------- ----- ----------
|        256      2048       3961       3934  99.318%         27
|
|                          Transaction Cache Statistics
|
| Transactions started:   11422  Lookups on tran:  238405    EC Merges: 1195
| Allocated Transactions:  2000  (Act= 0, Pend= 0, Comp= 1456, Free= 544)
|
|                   I/O Summary By Type
|                   -------------------
|
| Count       Waits       Cancels     Merges      Type
| ----------  ----------  ----------  ----------  ----------
|        177         209           0           0  File System Metadata
|       1418         197           0        1027  Log File
|       4595         150        1727           0  User File Data
|
|                   I/O Summary By Circumstance
|                   ---------------------------
|
| Count       Waits       Cancels     Merges      Circumstance
| ----------  ----------  ----------  ----------  ------------
|          6           6           0           0  Metadata cache read
|          3           1           0           0  User file cache direct read
|          0           0           0           0  Log file read
|          0           0           0           0  Metadata cache async delete write
|          0           0           0           0  Metadata cache async write
|        148          68           0           0  Metadata cache lazy write
|          0           0           0           0  Metadata cache sync delete write
|          0           0           0           0  Metadata cache sync write
|       4440          77        1727           0  User File cache direct write
|          0           0           0           0  Metadata cache file sync write
|         78          51           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
|         97         156           0           0  Metadata cache log file full write
|       1418         197           0        1027  Log file write
|          0           0           0           0  Metadata cache shutdown write
|
|                     zFS I/O by Currently Attached Aggregate
|
| DASD   PAV
| VOLSER IOs Mode       Reads     K bytes      Writes     K bytes  Dataset Name
| ------ --- ----  ----------  ----------  ----------  ----------  ------------
```

```
PRV001   1  R/W           0           0           0           0  SUDFS5.PRIVATE.LFSNET
PRV001   1  R/W           0           0           0           0  SUDFS5.PRIVATE.LFS106
PRV002   1  R/W           9         188        3427      205608  SUDFS5.PRIVATE.LFSFS
PRV001   1  R/W           0           0           0           0  SUDFS5.PRIVATE.TESTGROW
------           ----------  ----------  ----------  ----------
     4                   9         188        3427      205608  *TOTALS*


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

zFS Vnode Op Counts: This section shows the number of calls to the lower layer zFS components. One request from USS typically requires more than one lower layer call.

zFS Vnode Cache Statistics: This section shows the zFS vnode cache statistics. It shows the number of currently allocated vnodes and the vnode hit ratio. ″Allocates″ and ″Deletes″ show requests to create new vnodes (for operations like create or mkdir) and delete vnodes (for operations like 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 USS. In general zFS tries to honor the setting of the vnode_cache_size parameter and recycle vnode structures to represent different files. However, if USS requests more vnodes than zFS has allocated then zFS must allocate vnodes to avoid applications failing. In general a good hit ratio for this cache is desirable since a miss means initializing the data structures and the initialization requires a read of the objects status from disk. Often this would be in the metadata cache but it's not guaranteed. Hence a vnode cache lookup miss may sometimes require an I/O wait.

The vnode structure size is shown, however there are additional data structures anchored from the vnode which also takes space, everything added together yields over 1K of storage per vnode. So you should 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, and often more useful than an extremely large vnode cache.

Metadata Caching Statistics: This section shows the basic performance characteristics of the metadata cache. The metadata cache contains a cache of all disk blocks that contain metadata and any file data for files less than 7K in size. For files smaller than 7K, zFS will place multiple files in one disk block (for zFS a disk block is 8K bytes). Only the lower metadata management layers have the block fragmentation information so user file I/O for small files is performed directly through this cache rather than the user file cache. The statistics show the total number of buffers (each are 8K in size), the total bytes, the request rates and hit ratio of the cache. The higher the hit ratio the better the performance. Metadata is accessed frequently in zFS and all metadata is contained only (for the most part) in the metadata cache so a hit ratio of 80% or more is usually desirable.

Metadata Backing Cache Statistics: This section describes the performance of the extension to the metadata cache. The size of this extension is controlled by the metaback_cache_size configuration option. The backing cache is stored in a dataspace and is used only to avoid metadata reads from disk. All metadata updates and write I/O are performed from the primary metadata cache. Similar statistics to the metadata cache are shown for this cache. Every hit in this cache avoids one disk read, but normally the metadata backing cache is not needed except for workloads with many small user files or that are constrained in the zFS primary address space (possibly due to a large demand of zFS vnodes made by USS and its applications). Thus if the zFS address space has primary space available, the space should be given to the primary metadata cache. In the example above the metadata backing cache is providing no performance benefit (as shown by its 0 hit ratio). The metadata backing cache is not created by default. It can only be created by specifying the metaback_cache_size configuration option of the **IOEFSPRM** file or the zfsadm config command.

Directory Cache Statistics: zFS maintains a fixed size small cache of directory buffers for convenience. This directory cache is also backed by the metadata cache (that is, a directory page is always read to/from the metadata cache into/out of the directory cache). The size of this cache cannot be controlled by the administrator and usually does not have any performance ramifications.

Transaction Cache Statistics: zFS updates metadata on disk by writing the changes to the metadata to a log file. Each operation will create one or more transactions, write the updates to the logs associated with the transaction and then end the transaction. Each transaction has an associated state (Active, Pending, Complete or Committed):

Active There are still records being written to the log file describing updates being made by this transaction, hence the transaction was started but has not yet ended. (This is shown as ″Act″ in the report.)

Complete
The transaction has ended, all updates were written to the log file and the end transaction record is also written to the log for that transaction. (This is shown as ″Comp″ in the report.)

Committed
The transaction has ended and all updates are written to the log file AND all the log file pages that contain information about this transaction reside on disk. At this point the transaction is guaranteed. The update would not be lost if the system crashed. (In the report statistics for this count is not shown. As soon as a transaction is committed the structure representing the transaction is ″Free″ for reuse for another transaction.)

Equivalence Classes
zFS does not use a common technique called 2 phase locking or commit. Rather, transactions that are related are grouped into equivalence classes. zFS will decide when a transaction is related to or dependent on another transaction. When this determination is made the transactions are grouped into an equivalence class. Any transactions in the same equivalence class are committed together or backed out together in the event of a system crash. By using equivalence classes, threads running transactions simply run in parallel without added serialization between the two (other than locks if they hit common structures) and simply add their associated transactions to the same class. This thus increases throughput. The merge of equivalence classes occurs when two transactions that need to be made equivalent are both already in equivalence classes. In this case both classes are merged ″EC Merges″.

Pending
A transaction is pending when all its updates are written to the log file but other transactions in its same equivalence class have not ended. (This is ″Pend″.)

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

I/O Summary By Type & Circumstance: This section is mainly for IBM internal use in diagnosing performance related problems. zFS keeps detailed statistics on how often it performs I/O for various circumstances and how often it waits on that I/O to allow for easy determination of performance problems.

zFS I/O by Currently Attached Aggregate: The zFS I/O driver is essentially an I/O queue manager (one I/O queue per DASD). It uses Media Manager to issue I/O to VSAM datasets. It generally sends no more than 1 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 2 and round any fraction up. (Example, for a PAV DASD with 5 paths zFS will issue at most 3 I/Os at one time to Media Manager).

The reason zFS limits the I/O is that it uses a dynamic reordering and prioritization scheme to improve performance by reordering the I/O queue on demand. Thus high priority I/Os (I/Os that are currently being waited on for example) are placed up front, and an I/O can be made high priority at any time during its life.

This reordering has been proven to provide the best performance, and for PAV DASD, performance tests have shown that not sending quite as many I/Os as available paths allows zFS to reorder I/Os and leave paths available for I/Os that become high priority.

Another feature of the zFS I/O driver is that by queueing I/Os it allows I/Os to be cancelled (and thus the overhead of doing the I/O is removed). This is done in cases where a file was written and then immediately deleted for example. Finally, the zFS I/O driver will merge adjacent I/Os into one larger I/O to reduce I/O scheduling overhead, this is often done with log file I/Os since 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 users lose data in a crash) and yet allow them to be batched together for performance if the disk has a high load.

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

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

Finally, the number of times a thread processing a request must wait on I/O and the average wait time in milliseconds is shown. By using this information in conjunction with the KN report, you can break down zFS response time into what percentage of the response time is for I/O wait. To reduce I/O waits you can run with larger cache sizes. Small log files (small aggregates) that are heavily updated may result in I/Os to sync metadata to reclaim log file pages resulting in additional I/O waits. Note that this number is *NOT* DASD response time. It's affected by it but it's not the same. If a thread does not have to wait for an I/O then it has no I/O wait, if a thread has to wait for an I/O but there are other I/Os being processed it might actually wait for more than 1 I/O (the time in queue plus the time for the I/O).

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

```
LOCK:
                                    Locking Statistics


Untimed sleeps:         22   Timed Sleeps:          0   Wakeups:         21

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

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

                 Top 15 Most Higly 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 5 Most Common Thread Sleeps
|  Thread Wait     Pct.      Description
|  -----------    -----     -----------
|           22    100.0%     Transaction allocation wait
|            0      0.0%     OSI cache item cleanup wait
|            0      0.0%     Directory Cache Buffer Wait
|            0      0.0%     User file cache Page Wait
|            0      0.0%     User file cache File Wait
```

This report is mainly for IBM service use to diagnose performance problems due to lock contention.

The locking report shows a detailed breakdown of how often zFS waits for locks and which locks cause the most contention. It also monitors how often a thread sleeps waiting for an event. The lock waits and lock wait time and sleep waits and sleep wait time can be used in conjunction with the KN report to break down zFS response time into what percentage of the time zFS is waiting on internal locks or events to occur. See the following example:

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

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

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

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

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

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

```
STOR:
                  zFS Primary Address Space Storage Usage
                  ---------------------------------------

Total Bytes Allocated: 75907620 (74128K) (72M)
Total Pieces Allocated: 131853
Total Allocation Requests: 16783
Total Free Requests: 385

             Storage Usage By Component
             --------------------------
Bytes               No. of  No. of
Allocated   Pieces  Allocs  Frees   Component
----------  ------  ------  ------  ---------

      66019       5       0       0  USS Interface
      91956    4120    3750       0  Media Manager I/O driver
   33555876       4       0       0  Trace Facility
     281236       3       0       0  Message Service
      35636      34       3       0  Miscellaneous
       1064      13       0       0  Aggregate Management
```

```
|      111300     103       0       0  Filesystem Management
|       10753      16      18      17  Administration Command Handling
|       45976     280      50       2  Vnode Management
|      211600    2515    1677       0  Anode Management
|     2146508       9       0       0  Directory Management
|      378632    5418       0       0  Log File Management
|    36162280   25127       3       0  Metadata Cache
|      372504    4020       6       0  Transaction Management
|      328292     986     844       0  Asynchronous I/O Component
|       35340      63      14       0  Lock Facility
|        2172      51       3       0  Threading Services
|      104424    2645    1567     356  Cache Services
|       12832      11       0       0  Configuration parameters processing
|     1922308   86385    8848      10  User File Cache
|       30912      45       0       0  Storage Management
```

This report provides a breakdown of zFS storage usage. It can be used to determine how much storage zFS uses based on a configuration change (such as increasing or decreasing a zFS cache via the zfsadm config command).

Not shown here is the output of QUERY,STOR,DETAILS. That report breaks down each component and show how much storage is used for each data structure class and is intended primarily for IBM service.

```
| FILE:
| File System Name                              Aggr #  Flg  Operations
| --------------------------------------------  ------  ---  ----------
| OMVS.PRVROOT.OS390R10.DCEDFBLD.ROOT           100003  AM      1023618
| OMVS.R10.DCEDFBLD.ROOT.FEB2202                100003  A        631872
| OMVS.ZFS.REL14.DR2                            100003  AM       122208
| OMVS.ZFS.REL15.DR3                            100003  AM        20006
| OMVS.ZFS.REL15.DR1                            100003  AM        11967
| OMVS.DFS.REL15.DR1                            100003  AM           10
|
| USER.SBOX.G363634                             100000  AM       954875
| OMVS.ZFS.PROJ.DFS                             100000  AM       198768
| USER.SBOX.G861833                             100000  AM       135367
| USER.SBOX.G000140                             100000  AM        97696
| USER.HOME.G536657                             100000  AM         9758
| USER.HOME.G185211                             100000  AM         1931
| USER.HOME.G532755                             100000  AM          217
|
| OMVS.ZFS.REL15.DR2                            100002  AM       759108
| OMVS.ZFS.TMP                                  100002  AM        92620
| OMVS.SMB.REL16.DR1                            100002  AM        28357
| OMVS.ZFS.REL14.DR3                            100002  AM         1071
| OMVS.ZFS.REL14.DR1                            100002  AM          141
|
| USER.SBOX.G536657                             100001  AM       260296
| USER.HOME.G363634                             100001  AM        23093
| USER.SBOX.G185211                             100001  AM        21524
| USER.HOME.SUDFBLD                             100001  AM         6634
| USER.HOME.G861833                             100001  AM         3771
| USER.HOME.G000140                             100001  AM         2521
| USER.HOME.G528359                             100001  AM          144
| USER.HOME.G572781                             100001  AM           85
|
| OMVS.ZFS.DFBLD.DFSSRC                         100008  AM       274472
|
| OMVS.ZFS.LOCAL                                100009  AM       111722
|
| OMVS.ZFS.DCEDFBLD.DCES390.ETC.DCE             100010  AM        81632
|
| OMVS.ZFS.DCEDFBLD.DFSLOCAL                    100012  AM        52154
|
| OMVS.ZFS.DCEDFBLD.OS390R10.ETC                100004  AM        44108
|
```

```
| OMVS.ZFS.GPLTOOLS                          100006  AM         8458
|
| OMVS.ZFS.BLDTOOLS                          100007  AM         8120
|
| OMVS.ZFS.DCEDFBLD.VAR                      100005  AM          314
|
| OMVS.ZFS.USR.LOCAL                         100011  AM           54
```

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

The ″Operations″ columns are the count of USS vnode calls to that particular file system. It is not an I/O rate. With the RMF DASD reports and the LFS Aggregate I/O report and the FILE report you can balance your file systems and aggregates among disks to provide an even I/O spread if desired.

Debugging

If a problem occurs in zFS that requires IBM service support, it is important that appropriate problem determination information be obtained so that the problem can be resolved quickly.

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. The trace table can be reset (that is, set to empty) to minimize the amount of information generated if you are recreating a problem. This is accomplished with the operator **MODIFY ZFS,TRACE,RESET** command. The trace table can be formatted and sent to a trace output data set by using the operator **MODIFY ZFS,TRACE,PRINT** command. The trace output data set must be preallocated and its name must be specified in the **IOEFSPRM trace_dsn** option. A separate trace output data set is required for each member of a sysplex. (This requires separate **IOEFSPRM** files. Refer to Chapter 5, “Sysplex considerations” on page 15.) It should be allocated 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*. *nn* starts at 01 and is incremented 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 overlayed. You should not be accessing the trace output data set while a trace is being sent to the trace output data set. The space used by a particular trace depends on how large the **trace_table_size** is and how recently the trace was reset. As an example, a 32M **trace_table_size** can generate a trace output member of 100 cylinders of 3390. It is important that the trace output data set be large enough to hold the trace output since the complete trace will not be captured if it runs out of room sending the trace to the trace output data set.

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 dumped. ZFS also sends the trace to the trace output data set when a ZFS dump occurs. Note that when ZFS abends, other application failures may occur (since ZFS is unavailable during this time). For problem determination, these failures are not as important as the original ZFS failure and dump. ZFS attempts to restart after the dump is taken. If it is successful, the administrator must remount any zFS file systems.

If a failure of a zFS operation occurs (other than a user error), but zFS does not dump, you should get a trace of the failure, if possible. You can perform the following steps:

1. Issue the operator **MODIFY ZFS,TRACE,RESET** command
2. Recreate the failing scenario
3. Issue the operator **MODIFY ZFS,TRACE,PRINT** command
4. 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.

You can also obtain a dump of the ZFS address space by issuing the operator **MODIFY ZFS,DUMP** command.

IBM service may need more events to be traced. Additional tracing can be specified in two ways:

- The events that are traced can be added to by specifying 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.
- The events that are traced can be added dynamically by issuing operator **MODIFY ZFS,IOEDEBUG** commands. IBM specifies the exact statements needed.

If a hang condition occurs, the operator **MODIFY ZFS,QUERY,THREADS** command can be used to determine if any zFS threads are hanging and why. You can also issue the operator **MODIFY ZFS,ABORT** command to cause ZFS to send the trace to the trace output data set and to issue a dump. This also causes ZFS to terminate and attempt to restart.

If you have a ZFS dump but could not capture the trace, the trace can be obtained from the dump.

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

```
IOEZ00052I zFS kernel: Initializing z/OS    zSeries File System
Version 01.04.00 Service Level 0000000.
Created on Thu May  9 09:15:58 EDT 2002.
```

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

```
zfsadm -level
IOEZ00020I zfsadm: z/OS    zSeries File System
Version 01.04.00 Service Level 0000000.
Created on Thu May  9 09:15:58 EDT 2002.
```

The **IOEFSPRM msg_output_dsn** option can be specified that names a data set where messages from the ZFS PFS are written. This may be helpful for debugging since this data set can be sent to IBM service if needed. The **msg_output_dsn** is optional. If it is not specified, ZFS PFS messages go only to the system log. If it is specified, the data set should be preallocated as a sequential data set with a RECFM=VB and LRECL=248 and should be large enough to contain all ZFS PFS messages between restarts. The space used depends on how often ZFS is restarted and how much administration activity occurs. A suggested primary allocation is 25 cylinders with a secondary allocation of 25 cylinders. If the data set fills up, no more messages will be written to the data set. (They will still go to the system log.) After ZFS restart, the **msg_output_dsn** data set specified is overwritten.

Part 2. zFS administration reference

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

Chapter 10. z/OS system commands

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

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

These commands may be invoked from the operator console or from a Spool Display and Search Facility (SDSF) screen.

modify zfs process

Purpose

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

Format

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

```
modify procname,query,{all | settings | storage | threads}

modify procname,reset,{all | storage}

modify procname,trace,{reset | print}

modify procname,abort

modify procname,dump
```

Parameters

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

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

query Displays ZFS counters or values.

all Displays all the ZFS counters.

settings
Displays the ZFS configuration settings. These are based on the **IOEFSPRM** file and defaults.

storage
Displays the ZFS storage values.

threads
Displays the threads running in the ZFS address space.

Refer to “Performance Monitoring” on page 34. Performance and debugging for other query options available.

reset Resets ZFS counters to zero.

all Resets all the ZFS counters to zero.

storage
Resets the ZFS storage counters to zero.

trace Resets or prints the internal ZFS trace table.

reset Resets the internal (wrap around) trace table to empty.

print Formats and sends the current trace table to the data set specified in the **IOEFSPRM** file **trace_dsn** entry. This data set must be preallocated as a PDSE with RECFM VB and LRECL 133. It must be large enough to hold the formatted trace table. Refer to Chapter 9, “Performance and debugging” on page 31 for more information of the trace output data set.

abort Causes the ZFS PFS to abnormally terminate and dump. The internal trace table is also printed to the data set specified in the **IOEFSPRM** file **trace_dsn** entry.

dump Causes the ZFS PFS to dump. The internal trace table is also printed to the data set specified in the **IOEFSPRM** file **trace_dsn** entry.

Usage

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

Privilege Required

This command is a z/OS system command.

Examples

The following example queries all the ZFS counters:

```
modify zfs,query,all
```

The following example resets the ZFS storage counters:

```
modify zfs,reset,storage
```

The following example formats and sends the trace table to the data set specified in the **IOEFSPRM** file **trace_dsn** entry:

```
modify zfs,trace,print
```

The following example causes the ZFS PFS to dump and terminate:

```
modify zfs,abort
```

Related Information

File:

IOEFSPRM

setomvs reset

Purpose

Can be used to start the ZFS PFS if it has not been started at IPL or to restart 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

stop zfs

Purpose

Stops the ZFS PFS. All zFS file systems are unmounted and all zFS aggregates are detached. After the ZFS PFS stops, z/OS UNIX displays the message BPXF032D. A response to this message can restart the ZFS PFS (**r nn,r**) or remove the message (**r nn,i**) and leave the ZFS PFS terminated.

Format

stop *procname*

Parameters

procname The name of the ZFS PROC. The default procname is **ZFS**.

Usage

The STOP command stops the ZFS PFS (**ioefscm**).

Privilege Required

This command is a z/OS system command.

Examples

The following command stops the ZFS PFS:

```
stop zfs
```

Related Information

File:

IOEFSPRM

Chapter 11. zFS commands

This chapter provides a description of the relevant zFS commands.

ioeagfmt

Purpose

Creates an HFS compatibility mode aggregate or a multi-file system aggregate.

Format

```
ioeagfmt -aggregate name [-initialempty blocks]
[-size blocks] [-logsize blocks] [-overwrite] [-compat]
[-owner {uid|name}][-group {gid|name}] [-perms {number}]
[-grow blocks] [-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 upper case. The following characters can be included in the name of an aggregate:

- All uppercase and lowercase alphabetic characters (a to z, A to Z)
- All numerals (0 to 9)
- The . (period)
- The - (dash)
- The _ (underscore).

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

-initialempty *blocks*

Specifies the number of 8K blocks that will be left empty at the beginning of the aggregate. The default is 1. This option is not normally specified.that will be left empty at the beginning of the aggregate. The default is 1. This option is not normally specified.

-size *blocks* Specifies the number of 8K blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM LDS. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM LDS is attempted after the primary allocation is formatted unless the **-grow** option is specified. In that case, multiple extensions of the amount specified in the **-grow** option will be attempted until the **-size** is satisfied. The size may be rounded up to a control area (CA) boundary by DFSMS. It is not necessary to specify a secondary allocation size on the DEFINE of the VSAM LDS for this extension to occur. Space must be available on the volume(s).

-logsize *blocks*

Specifies the size in 8K blocks of the log. The default is 1% of the aggregate size (which is sufficient).

-overwrite Required if you are reformatting an existing aggregate. Use this option with caution, since it destroys any existing data. This option is not usually specified.

-compat Indicates that a compatibility mode aggregate should be created. This means that in addition to formatting the VSAM Linear Data Set (LDS) as a zFS aggregate, a zFS file system by the same name (the aggregate name) is created and its quota is set to the size of the available blocks on the aggregate. This option should normally be specified unless you want to create a multi-file system aggregate. Refer to Chapter 8, “Multi-file system aggregates” on page 25 for more information on multi-file system aggregates.

-owner *uid* | *name*
Specifies the owner for the root directory of the file system. This is used with the **-compat** option, otherwise it is ignored. It may be specified as a z/OS user ID or as a *uid*. The default is the *uid* of the issuer of **ioeagfmt**.

-group *gid* | *name*
Specifies the group owner for the root directory of the file system. This is used with the **-compat** option, otherwise it is ignored. It may 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** is specified, the group is that owner's default group.

-perms *number*
Specifies the permissions for the root directory of the file system. This is used with the **-compat** option, otherwise it is ignored. The number can be specified as octal (for example, o755), as hexadecimal (for example, x1ED), or as decimal (for example, 493). The default is o755 (owner read/write/execute, group read/execute, other read/execute).

-grow *blocks* Specifies the number of 8K blocks that zFS will use as the increment for extension when the **-size** option specifies a size greater than the primary allocation.

-level Prints the level of the **ioeagfmt** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **ioeagfmt** utility is used to format an existing VSAM LDS as a zFS aggregate. All zFS aggregates must be formatted before use (including HFS compatibility mode aggregates). **ioeagfmt** can be run even if the ZFS PFS is not active on the system. The size of the aggregate is as many 8K blocks as fits in the primary allocation of the VSAM LDS or as specified in the **-size** option. The **-size** option can cause one additional extension to occur during formatting. To extend it further, use the **zfsadm grow** command. If **-overwrite** is specified, all existing primary and secondary allocations are formatted and the size includes all of that space.

Privilege Required

The user must have ALTER authority to the VSAM LDS and must be UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

Examples

Figure 7 on page 58 shows an example of a job that creates a compatibility mode aggregate and file system.

ioeagfmt

```
//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(2))
/*
//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 7. Job to create a compatibility mode aggregate and file system

ioeagslv

Purpose

Scans an aggregate and reports inconsistencies. Aggregates can be verified, recovered (that is, the log is replayed), or salvaged (that is, the aggregate is repaired). This utility is known as the Salvager.

Note: This utility is not normally needed. If a system failure occurs, the aggregate log is replayed automatically, the next time the aggregate is attached (or for compatibility mode aggregates, the next time the file system is mounted). This normally brings the aggregate (and all the file systems) back to a consistent state.

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.

-recoveronly Directs the Salvager to recover the specified aggregate. The Salvager replays the log of metadata changes that resides on the aggregate. Refer to “Usage” for information about using and combining the command’s options.

-verifyonly Directs the Salvager to verify the specified aggregate. The Salvager examines the structure of the aggregate to determine if it contains any inconsistencies, reporting any that it finds. Refer to “Usage” for information about using and combining the command’s options.

-salvageonly Directs the Salvager to salvage the specified aggregate. The Salvager attempts to repair any inconsistencies it finds on the aggregate. Refer to “Usage” for information about using and combining the command’s options.

-verbose Directs the Salvager to produce detailed information about the aggregate as it executes. The information is useful primarily for debugging purposes. It is displayed on standard output (which can be redirected). Use this option alone or with any combination of the available options.

-level Prints the level of the **ioeagslv** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **ioeagslv** utility invokes the Salvager on the zFS aggregate specified with the **-aggregate** option. Following a system restart, the Salvager employs the zFS file system log mechanism to return consistency to a file system by running recovery on the aggregate on which the file system resides. Recovery is the replaying of the log on the aggregate; the log records all changes made to metadata as a result of operations such as file creation and deletion. If problems are detected in the basic structure of the aggregate, if the log mechanism is damaged, or if the storage medium of the aggregate is suspect, the **ioeagslv** utility must be used to verify or repair the structure of the aggregate.

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

- Specify the **-recoveronly** and **-verifyonly** options

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

- Specify the **-salvageonly** option

  To attempt to repair any inconsistencies found in the structure of the aggregate without first running recovery on it. Use this option if you believe the log is damaged or replaying the log does not return consistency to the aggregate and may in fact further damage it. In most cases, you do not salvage an aggregate without first recovering it.

- Omit the **-recoveronly**, **-verifyonly**, and **-salvageonly** options

  To run recovery on the aggregate and then attempt to repair any inconsistencies found in the structure of the aggregate. Because recovery eliminates inconsistencies in an undamaged file system, an aggregate is typically recovered before it is salvaged. In general, it is good first to recover and then to salvage an aggregate if a system goes down or experiences a hardware failure.

  Omit these three options if you believe the log should be replayed before attempts are made to repair any inconsistencies found on the aggregate. (Omitting the three options is equivalent to specifying the **-recoveronly** and **-salvageonly** options.)

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; once one of these options is specified, you must explicitly request any operation you want the Salvager to perform on the aggregate.

The basic function of the Salvager is similar to that of the z/OS UNIX **fsck** program. The Salvager recovers a zFS aggregate and repairs problems it detects in the structure of the aggregate. It does not verify or repair the format of user data contained in files on the aggregate. If it makes changes, the Salvager displays the pathnames of the files affected by the modifications, when the pathnames can be determined. The owners of the files can then verify the files' contents, and the files can be restored from backups if necessary

The Salvager verifies the structure of an aggregate by examining all of the anodes, directories, and other metadata in each file system on the aggregate. An ***anode*** is an area on the disk that provides information used to locate data such as files, directories, ACLs, and other types of file system objects. Each file system contains an arbitrary number of anodes, all of which must reside on the same aggregate. By following the links between the various types of anodes, the Salvager can determine whether the organization of an aggregate and the file systems it contains is correct and make repairs if necessary.

Not all aggregates can be salvaged. In cases of extensive damage to the structure of the metadata on an aggregate or damage to the physical disk that houses an aggregate, the Salvager cannot repair

inconsistencies. Also, the Salvager cannot verify or repair damage to user data on an aggregate. The Salvager cannot detect problems that modified the contents of a file but did not damage the structure of an aggregate or change the metadata of the aggregate.

Like the z/OS UNIX **fsck** command, the Salvager analyzes the consistency of an aggregate by making successive passes through the aggregate. With each successive pass, the Salvager examines and extracts a different type of information from the blocks and anodes on the aggregate. Later passes of the Salvager use information found in earlier passes to help in the analysis.

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

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

Privilege Required

The privileges required depend on whether the **-recoveronly**, **-verifyonly**, or **-salvageonly** option is specified with the command:

- If just the **-verifyonly** option is included, the issuer needs only the READ authority for the specified VSAM LDS (aggregate).
- If the **-recoveronly** or **-salvageonly** option is included, or if all three of these options are omitted, the issuer must have ALTER authority for the specified VSAM LDS.

In addition, the user must be uid 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

Examples

Figure 8 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')
//SYSPRINT DD     SYSOUT=H
//STDOUT   DD     SYSOUT=H
//STDERR   DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 8. Job to verify a zFS aggregate

MOUNT

Purpose

Mounts a file system into the z/OS UNIX hierarchy. This section only documents MOUNT options that are unique to zFS. For additional information on this command, refer to the *z/OS: UNIX System Services Command Reference*.

Format

```
MOUNT TYPE(file_system_type) [PARM(parameter_sting)]
```

Options

TYPE (*file_system_type***)**
Specifies the file system type. For zFS, this must be specified as **ZFS**.

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 in quotes.

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

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

READAHEAD | NOREADAHEAD
Specifies whether this file system will be accessed sequentially or not and whether the zFS read ahead processing normally done should be enabled or disabled. NOREADAHEAD should be used for file systems that have random access patterns (for example, for file systems that are used like a database). The default is to do read ahead processing.

The following parameters apply to compatibility mode aggregates:

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

AGGRGROW | NOAGGRGROW
Specifies whether the aggregate is eligible to be dynamically grown. The growth will be based on the secondary allocation of the aggregate and will occur when the aggregate becomes full. The default is the **aggrgrow** specification in the **IOEFSPRM** file. This parameter only applies to compatibility mode aggregates/file systems.

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

The following parameters apply to multi-file system aggregates:

AGGREGATE(*aggregate_name***)**
Specifies the name of the aggregate that the file system resides in. This is

normally used when the zFS file system you are mounting has the same name as a zFS file system in another aggregate. See also the **FILESYSTEM** parameter. This parameter only applies to multi-file system aggregates/file systems.

FILESYSTEM(*zFS_filesystem_name***)**
Specifies the name of the zFS file system that you are mounting. This is normally used when the zFS file system you are mounting has the same name as a zFS file system in another aggregate. If this is not specified, zFS assumes that the zFS file system name is the same as the UNIX file system name (specified in the **MOUNT FILESYSTEM** option). This parameter only applies to multi-file system aggregates/file systems.

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

Usage

The **MOUNT** command mounts a zFS file system.

MOUNT of a compatibility mode aggregate is serialized with other **zfsadm commands** (since **MOUNT** of a compatibility mode aggregate does an implicit attach).

If you attempt to mount a compatibility mode aggregate/file system read-only and it fails because it needs to run recovery (return code EROFS (141) and reason code EFxx6271), you should temporarily mount it read-write (so it can complete the recovery process) and then mount it read-only.

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('FSFULL(90,5)')
```

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

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

Related Information

Command:

UNMOUNT (For information on this command, refer to the *z/OS: UNIX System Services Command Reference*.)

MOUNT

File:
IOEFSPRM

zfsadm

Purpose

Introduction to the **zfsadm** command suite.

Command Syntax

The **zfsadm** commands have the same general structure:

command {-option1 *argument...* | **-option2 {***argument1* | *argument2***}...} [-optional_information]**

The following example illustrates the elements of a **zfsadm** command:

zfsadm detach {-all | **-aggregate** *name***} [-help]**

The following list summarizes the elements of the **zfsadm** command:

- **Command** - A command consists of the command suite (**zfsadm** in the previous example) and the command name (**detach**). The command suite and the command name must be separated by a space. The command suite specifies the group of related commands.
- **Options** - Command options always appear in bold type in the text, are always preceded by a **-** (dash), and are often followed by arguments. In the previous example, **-aggregate** is an option, with *name* as its argument. An option and its arguments tell the program which entities to manipulate when executing the command (for example, which aggregate, or which file system). In general, the issuer should provide the options for a command in the order detailed in the documentation. The { | } (braces separated by a vertical bar) indicate that the issuer must enter either one option or the other (**-all** or **-aggregate** in the previous example).
- **Arguments** - Arguments for options always appear in italic type in the text. The { | } indicate that the issuer must enter either one argument or the other (**-all** or **-aggregate** in the preceding example). The ... (ellipsis) indicates that the issuer can enter multiple arguments.
- **Optional** information - Some commands have optional, as well as required, options and arguments. Optional information is enclosed in [ ] (brackets). All options except **-all** or **-aggregate** in the previous example are optional.

Options

The following options are used with many **zfsadm** commands. They are also listed with the commands that use them.

-filesystem *name*
: Specifies the file system to use with the command.

-aggregate *name*
: Specifies the aggregate name of the aggregate to use with the command.

-size *kbytes*
: Specifies the size in K-bytes for the *kbytes* argument.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored. For complete details about receiving help, refer to “Receiving Help” on page 66.

Usage

Most **zfsadm** commands are administrative-level commands used by system administrators to manage file systems and aggregates. They apply to multi-file system aggregates although several apply to compatibility mode aggregates, too (for example, **zfsadm grow** and **zfsadm quiesce**/**unquiesce**). They can be issued from OMVS or as a batch job. The descriptions of the **zfsadm aggrinfo** and the **zfsadm attach** commands show examples of issuing them as a batch job. The other **zfsadm** commands can be run as a batch job in a similar manner.

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 (since 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.

When supplying an argument to a **zfsadm** command, the option (for example **-aggregate**) associated with the argument (for example, OMVS.PRV.AGGR001.LDS0001) can be omitted if:

- All arguments supplied with the command are entered in the order in which they appear in the command's syntax. (The syntax for each command appears with its description in this chapter.)
- Arguments are supplied for all options that precede the option to be omitted.
- All options that precede the option to be omitted accept only a single argument.
- No options, either those that accept an argument or those that do not, are supplied before the option to be omitted.

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 since it has an **-aggrfull** option.)

It is also valid to abbreviate a command name to the shortest form that still distinguishes it from the other command names in the suite. For example, it is acceptable to shorten the **zfsadm grow** command to **zfsadm g** because no other command names in the **zfsadm** command suite begin with the letter **g**. However, there are three **zfsadm** commands that begin with **l**: **zfsadm lsaggr**, **zfsadm lsfs**, and **zfsadm lsquota**. To remain unambiguous, they can be abbreviated to **zfsadm lsa**, **zfsadm lsf**, and **zfsadm lsq**.

The following examples illustrate three acceptable ways to enter the same **zfsadm grow** command:

Complete command:

```
zfsadm grow -aggregate omvs.prv.aggr001.lds0001 -size 50000
```

Abbreviated command name and abbreviated options:

```
zfsadm g -a omvs.prv.aggr001.lds0001 -s 50000
```

Abbreviated command name and omitted options:

```
zfsadm g omvs.prv.aggr001.lds0001 50000
```

Note: The ability to abbreviate or omit options is intended for interactive use. If you imbed commands in a shell script, you should not omit options nor abbreviate them. If an option is added to a command in the future, it may increase the minimum unique abbreviation required for an existing option or change the order of options.

Receiving Help

There are several different ways to receive help about **zfsadm** commands. The following examples summarize the syntax for the different help options available:

$ zfsadm help

Displays a list of commands in a command suite.

$ zfsadm help -topic *command*
Displays the syntax for one or more commands.

$ zfsadm apropos -topic *string*
Displays a short description of any commands that match the specified *string*.

Privilege Required

zfsadm commands that query information (for example, **lsfs**, **aggrinfo**) can be issued by any user that has READ authority to the data set that contains the **IOEFSPRM** file. **zfsadm** commands that modify (for example, **setquota**, **create**) additionally require that the issuer be logged in as **root** or must have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class. Specific privilege information is listed with each command's description.

Related Information

Commands:

zfsadm aggrinfo
zfsadm apropos
zfsadm attach
zfsadm clone
zfsadm clonesys
zfsadm config
zfsadm configquery
zfsadm create
zfsadm define
zfsadm delete
zfsadm detach
zfsadm format
zfsadm grow
zfsadm help
zfsadm lsaggr
zfsadm lsfs
zfsadm lsquota
zfsadm quiesce
zfsadm rename
zfsadm setquota
zfsadm unquiesce

Files:

IOEFSPRM

zfsadm aggrinfo

Purpose

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

Format

zfsadm aggrinfo [-aggregate *name***] [-level] [-help]**

Options

-aggregate *name*
: Specifies the name of an aggregate about which information is to be displayed. The aggregate must be attached. The aggregate name is not case sensitive. It is translated to upper case. If this option is omitted, information is provided about all of the attached aggregates on the system. Compatibility mode aggregates are implicitly attached when they are mounted.

-level
: Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

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 on the system is displayed. 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) and whether it is a compatibility mode aggregate (COMP) or a multi-file system aggregate (MULT).
- The amount of space available in K-bytes.
- The total amount of space in the aggregate in K-bytes. (To grow an aggregate, you would specify a number larger than the sum of this number and the next number.)
- The amount of space reserved for other processing in K-bytes.

Privilege Required

READ authority to the data set that contains the **IOEFSPRM** file is required.

Examples

The following OMVS example displays information about the disk space available on all aggregates attached on the system:

```
zfsadm aggrinfo

OMVS.PRV.AGGR004.LDS0004 (R/W COMP): 1149 K free out of total 1296 (136 reserved)
OMVS.PRV.AGGR003.LDS0001 (R/W MULT): 2344983 K free out of total 2369344 (23928 reserved)
OMVS.PRV.AGGR002.LDS0002 (R/W MULT): 1159 K free out of total 1296 (136 reserved)
OMVS.PRV.AGGR001.LDS0001 (R/W MULT): 176119 K free out of total 177992 (2000 reserved)
```

Figure 9 on page 69 shows the same example as a job that invokes **zfsadm aggrinfo**.

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

Figure 9. Job to display aggregate information

Related Information

Command:

zfsadm lsaggr

File:

IOEFSPRM

zfsadm apropos

Purpose

Shows each help entry containing a specified string.

Format

zfsadm apropos -topic *string* **[-level] [-help]**

Options

-topic Specifies the keyword string for which to search. If it is more than a single word, surround it with double quotes (″″) or other delimiters. Type all strings for **zfsadm** commands in all lowercase letters.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

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

READ authority to the data set that contains the **IOEFSPRM** file is required.

Results

The first line of an online help entry for a command lists the command and briefly describes its function. This command displays the first line for any **zfsadm** command where the string specified by **-topic** is part of the command name or first line.

Examples

The following command lists all **zfsadm** commands that have the word **list** in their names or short descriptions:

```
zfsadm apropos list

lsaggr: list aggregates
lsfs: list filesystem information
lsquota: list filesystem and aggregate space usage
```

Related Information

Command:

zfsadm help

zfsadm attach

Purpose

Attaches an aggregate to zFS as a multi-file system aggregate.

Note: Do not explicitly attach compatibility mode aggregates. They are implicitly attached when the file system is mounted.

Format

```
zfsadm attach {-all | -aggregate name} [-aggrfull threshold,increment ] [-R/O] [-nbs] [-nonbs]
[-aggrgrow] [-noaggrgrow] [-level] [-help]
```

Options

-all Specifies that all aggregates listed in the **IOEFSPRM** file that are not currently attached are to be attached. **-all** honors the **define_aggr** options. Use this option or use **-aggregate**.

-aggregate *name*
Specifies the name of the aggregate to be attached. The aggregate name is not case sensitive. It is translated to upper case. This aggregate does not need an entry in the **IOEFSPRM** file. If the aggregate is not contained in the **IOEFSPRM** file, it needs to be attached again if it is a multi-file system aggregate and the ZFS PFS is restarted.

Note: Compatibility mode aggregates do not need to be attached with the **zfsadm attach** command, nor do they need to be contained in the **IOEFSPRM** file. Compatibility mode aggregates are automatically attached on **MOUNT** of the compatibility mode file system.

-aggrfull *threshold,increment*
Specifies the threshold and increment for reporting aggregate full error messages to the operator. Both numbers must be specified. The first number is the threshold percentage and the second number is the increment percentage. For example, if 90,5 were specified, the operator would be notified when the aggregate became 90% full, then again at 95% full and again at 100% full. This overrides the **aggrfull** option in the **define_aggr** entry for this aggregate in the **IOEFSPRM** file and the global **aggrfull** entry in the **IOEFSPRM**. The default is the global **aggrfull** entry of the **IOEFSPRM** file.

-R/O Specifies that the aggregate should be opened in read-only mode. A read-only aggregate means that all file systems are read-only and can only be mounted as read-only. The default is read-write unless **R/O** is specified.

-nbs Specifies whether New Block Security is used for file systems in this aggregate. New block security refers to the guarantee made when a system crashes. If **-nbs** is specified then we guarantee that at the time of a crash, if a file was being extended or new blocks were being allocated for the file, but the user data has not yet made it to the disk when the crash occurred then we show the newly allocated blocks as all binary 0's and not whatever was on disk in those blocks at time of failure. The default for this is the global **nbs** entry in the **IOEFSPRM** file. **-nbs** degrades performance. It is rare that you would have a case where the metadata updates are committed but the corresponding user updates are not on disk. However, in the case of high security requirements, this specification provides this guarantee.

-nonbs Specifies that the New Block Security guarantee is not required. This is the default. Refer to the explanation of **-nbs** for a description of the New Block Security guarantee.

-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(s). The default is the **aggrgrow** option of the **IOEFSPRM** file.

-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.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm attach** command attaches zFS aggregates on this system. File systems on attached aggregates are available to be mounted on the system.

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

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

The **zfsadm lsaggr** command can be used to display a current list of all aggregates attached on this system.

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

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

Privilege Required

The issuer must have READ authority to the data set that contains the **IOEFSPRM** file and is required to be logged in as **root** or to have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

Examples

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

```
zfsadm attach -all
```

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

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

Figure 10 on page 73 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 10. Job to attach an aggregate

Note: If you want to specify the **R/O** option, you must specify a leading slash. Otherwise, LE will treat the characters before the slash as LE parameters. That is, you must use **PARM=('/attach OMVS.PRV.AGGR001.LDS0001 -R/O')**

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

Related Information

Commands:

zfsadm create
zfsadm lsaggr

File:

IOEFSPRM

zfsadm clone

Purpose

Creates a backup version of a specific file system.

Format

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

Options

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

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

-aggregate *name*
Specifies the name of the aggregate where the zFS file system name resides. It is specified to qualify the zFS file system name (**-filesystem**) when there are multiple zFS file systems with the same name in different aggregates. The aggregate name is not case sensitive. It is always folded to upper case.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

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

If a backup version already exists, the new clone replaces it. If the read-write file system name is longer than 40 characters, the clone fails.

Privilege Required

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

Examples

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

```
$ zfsadm clone OMVS.PRV.FS1
IOEZ00225I File system OMVS.PRV.FS1 successfully cloned.
```

Related Information

Command:
zfsadm clonesys

zfsadm clonesys

Purpose

Creates backup versions of all indicated file systems.

Format

zfsadm clonesys [-prefix *string*] **[-aggregate** *name*] **[-level] [-help]**

Options

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

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

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

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

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

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

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

Use the **zfsadm clone** command to back up a single read-write zFS file system.

Privilege Required

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

Examples

The following example creates a backup version of each zFS file system on the system in aggregate OMVS.PRV.AGGR001.LDS0001:

```
$ zfsadm clonesys -aggregate omvs.prv.aggr001.lds0001

IOEZ00026I Clonesys starting for aggregate id 100000, file system * all filesystems *
IOEZ00031I Clonesys ending for aggregate id 100000 (Total: 3, Failed: 0, Time 0.148)
```

Related Information

Command:

zfsadm clone

File:

IOEFSPRM

zfsadm config

Purpose

Changes the value of zFS configuration (IOEFSPRM) options in memory.

Format

```
zfsadm config [-admin_threads number] [-user_cache_size number] [-meta_cache_size number]
              [-log_cache_size number] [-sync_interval number] [-vnode_cache_size number] [-nbs {on|off}]
              [-fsfull threshold,increment] [-aggrfull threshold,increment] [-trace_dsn PDSE_dataset_name]
              [-tran_cache_size number] [-msg_output_dsn Seq_dataset_name]
              [-user_cache_readahead {on|off}] [-metaback_cache_size number] [-fsgrow increment,times]
              [-aggrgrow {on|off} ] [-level] [-help]
```

Options

-admin_threads *number*
Specifies the number of threads defined to handle pfsctl or mount requests.

-user_cache_size *number*
Specifies the size, in bytes, of the cache used to contain file data.

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

-log_cache_size *number*
Specifies the size, in bytes, of the cache used to contain buffers for log file pages.

-sync_interval *number*
Specifies the number of seconds between syncs.

-vnode_cache_size *number*
Specifies the size of the internal vnode cache.

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

-fsfull *threshold,increment*
Specifies the threshold and increment for reporting file system quota full error messages to the operator.

-aggrfull *threshold,increment*
Specifies the threshold and increment for reporting aggregate full error messages to the operator.

-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.

-msg_output_dsn *Seq_dataset_name*
Specifies the name of a data set that contains any output messages that come from the ZFS PFS.

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

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

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

-aggrgrow on | off
Specifies whether an aggregate should be dynamically extended when it runs out of physical space.

-allow_dup_fs on | off
Specifies whether multiple file systems with the same name can be created in different aggregates.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

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 need to modify the IOEFSPRM file since ZFS reads the IOEFSPRM file to determine the configuration values the next time ZFS is started. The values that can be specified for each option are the same as the values that can be specifed for that option in the IOEFSPRM file.

Privilege Required

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

Examples

The following example changes the size of the user cache:

```
$ zfsadm config -user_cache_size 64M

IOEZ00300I Successfully set -user_cache_size to 64M
```

Related Information

Command:
zfsadm configquery

File:
IOEFSPRM

zfsadm configquery

Purpose

Queries the current value of zFS configuration options.

Format

```
zfsadm configquery [-adm_threads] [-aggrfull] [-aggrgrow] [-all] [-allow_dup_fs] [-auto_attach]
                   [-cmd_trace] [-debug_dsn] [-fsfull] [-fsgrow] [-log_cache_size]
                   [-meta_cache_size] [-metaback_cache_size] [-msg_input_dsn] [-msg_output_dsn]
                   [-nbs] [-storage_details] [-sync_interval] [-trace_dsn] [-trace_table_size]
                   [-tran_cache_size] [-user_cache_readahead] [-user_cache_size] [-usercancel]
                   [-vnode_cache_size] [-level] [-help]
```

Options

-adm_threads
Displays the number of threads defined to handle pfsctl or mount requests.

-aggrfull Displays the threshold and increment for reporting aggregate full error messages to the operator.

-aggrgrow Displays whether an aggregate should be dynamically extended when it runs out of physical space.

-all Displays the full set of configuration options.

-allow_dup_fs
Displays whether multiple file systems with the same name can be created in different aggregates.

-auto_attach Displays whether aggregates defined and listed in the IOEFSPRM file are attached when ZFS is started.

-cmd_trace Displays whether command tracing is active.

-debug_dsn Displays the name of the debug input parameters data set.

-fsfull Displays the threshold and increment for reporting file system quota full error messages to the operator.

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

-log_cache_size
Displays the size, in bytes, of the cache used to contain buffers for log file pages.

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

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

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

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

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

-sync_interval
Displays the number of seconds between syncs.

-trace_dsn Displays the name of the data set that contains the output of any operator MODIFY ZFS,TRACE,PRINT commands or the trace output if ZFS abends.

-trace_table_size
Displays the size, in bytes, of the internal trace table.

-tran_cache_size
Displays the number of transactions in the transaction cache.

-user_cache_readahead
Displays whether zFS should attempt to read ahead or not.

-user_cache_size
Displays the size, in bytes, of the cache used to contain file data.

-vnode_cache_size
Displays the size of the internal vnode cache.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

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.

Privilege Required

The issuer must have READ authority to the data set that contains the **IOEFSPRM** file.

Examples

The following example displays the 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.
```

Related Information

Command:
zfsadm config

File:
IOEFSPRM

zfsadm create

Purpose

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

Format

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

Options

-filesystem *name*

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

- All uppercase and lowercase alphabetic characters (a to z, A to Z)
- All numerals (0 to 9)
- The . (period)
- The - (dash)
- The _ (underscore).

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

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

However, you can create file systems with the same name in different multi-file system aggregates (if you have enabled this by using the **allow_duplicate_filesystems** option in the **IOEFSPRM**). The **zfsadm** commands and the **MOUNT** command can specify a zFS file system name that is qualified by its aggregate name.

-aggregate *name*

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

-size *kbytes* Specifies the initial maximum quota for the file system in K-bytes.

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

-group *name* | *gid*
Specifies the group of the root directory of the file system. This can be specified as a z/OS group ID or a numeric *gid*. The default is the group of the issuer of the command. If only **-owner** is specified, the group is the owner's default group.

-perms *permbits*
Specifies the permissions for the root directory of the file system. The number can be specified as octal (for example, o755), as hexadecimal (for example, x1ED), or as decimal (for example, 493). The default is o755 (owner read/write/execute, group read/execute, other read/execute).

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

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

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

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

Privilege Required

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

Examples

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

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

IOEZ00099I File system OMVS.USER.PAT created successfully
```

Related Information

Commands:
- **zfsadm delete**
- **zfsadm lsfs**

File:
- **IOEFSPRM**

zfsadm define

Purpose

Defines a VSAM Linear Data Set (VSAM LDS) in preparation to be formatted as a zFS aggregate.

Format

```
zfsadm define -aggregate name [-dataclass SMS_data_class] [-managementclass SMS_management_class]
                              [-storeageclass SMS_storage_class] [-model model [catalog]]
                              [-catalog 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 LDS that is defined. The aggregate name is not case sensitive. It is translated to upper case.

-dataclass *SMS_data_class*
Specifies the name of the data class to be used when the VSAM LDS is defined.

-managementclass *SMS_management_class*
Specifies the name of the management class to be use when the VSAM LDS is defined.

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

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

-catalog *catalog*
Specifies the name of the catalog in which the VSAM Linear Data Set is to be defined.

-volumes *volume*
Specifies the volume(s) on which the VSAM LDS may have space.

-cylinders *primary* **[***secondary***]**
Specifies the primary and optionally, the secondary allocation size for the VSAM LDS in cylinders. When **records** is specified, the record size is assumed to be 4089 bytes.

-kilobytes *primary* **[***secondary***]**
Specifies the primary and optionally, the secondary allocation size for the VSAM LDS in kilobytes.

-megabytes *primary* **[***secondary***]**
Specifies the primary and optionally, the secondary allocation size for the VSAM LDS in megabytes.

-records *primary* **[***secondary***]**
Specifies the primary and optionally, the secondary allocation size for the VSAM LDS in records. When **records** is specified, the record size is assumed to be 4089 bytes.

-tracks *primary* **[***secondary***]**
Specifies the primary and optionally, the secondary allocation size for the VSAM LDS in tracks.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm define** command defines a VSAM LDS. The VSAM LDS is available to be formatted as a zFS aggregate. The command creates a DEFINE CLUSTER command string for a VSAM LDS with SHAREOPTIONS(2) and passes it to the IDCAMS utility. If a failure occurs, the **zfsadm define** command may display additional messages from IDCAMS indicating the reason for the failure.

Privilege Required

The issuer of the **zfsadm define** command requires sufficient authority to create the VSAM LDS.

Examples

The following command defines a VSAM LDS.

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

Related Information

Commands:

zfsadm format

zfsadm delete

Purpose

Removes a file system.

Format

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

Options

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

-aggregate *name*
: Specifies the name of the aggregate where the zFS file system name resides. It is specified to qualify the zFS file system name (**-filesystem**) when there are multiple zFS file systems with the same name in different aggregates. The aggregate name is not case sensitive. It is always folded to upper case.

-level
: Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

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

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

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

Privilege Required

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

Examples

The following command deletes the read-write file system named **OMVS.USER.PAT** and its backup version (if it exists) from its aggregate:

```
zfsadm delete OMVS.USER.PAT

IOEZ00105I File system OMVS.USER.PAT deleted successfully
```

Related Information

Commands:

zfsadm create

zfsadm delete

zfsadm lsfs

File:

IOEFSPRM

zfsadm detach

Purpose

Detaches one or more aggregates from zFS. This makes any file systems contained in the aggregate unavailable to zFS.

Format

zfsadm detach {-all | -aggregate *name***} [-level] [-help]**

Options

-all Specifies that all attached aggregates are to be detached. Use this option or use **-aggregate** but not both.

-aggregate *name*
Specifies the aggregate name of the aggregate to be detached. Use this option or use **-all**, but not both. The aggregate name is not case sensitive. It is always translated to upper case.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

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.

Before detaching an aggregate, all file systems in the aggregate must be unmounted. Therefore, **zfsadm detach -all** will not detach 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 profile in the UNIXPRIV class.

Examples

The following is an example of a **zfsadm detach** command that detaches the aggregate OMVS.PRV.AGGR001.LDS0001.

```
zfsadm detach -aggregate omvs.prv.aggr001.lds0001

IOEZ00122I Aggregate OMVS.PRV.AGGR001.LDS0001 detached successfully
```

Related Information

Commands:
zfsadm attach

Files:
IOEFSPRM

zfsadm format

Purpose

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

Format

```
zfsadm format -aggregate name [-initialempty blocks] [-size blocks] [-logsize blocks]
                              [-overwrite][-compact] [-owner {uid | name}] [-group {gid | name}]
                              [-perms decimal | octal | hex_number] [-grow blocks] [-level] [-help]
```

Options

-aggregate *name*

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

-initialempty *blocks*

Specifies the number of 8K blocks that will be left empty at the beginning of the aggregate. The default is 1. This option is not normally specified.

-size *blocks* Specifies the number of 8K blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM LDS. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM LDS is attempted after the primary allocation is formatted unless the **-grow** option is specified. In that case, multiple extensions of the amount specified in the **-grow** option will be attempted until the **-size** is satisfied. Space must be available on the volume(s).

-logsize *blocks*

Specifies the number of 8K blocks reserved for the aggregate log. The default is 1 percent of the size of the aggregate. This is normally sufficient.

-overwrite Specifies that an existing zFS aggregate should be overlaid. All existing data will be lost. Use this option with caution. This option is not usually specified.

-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.

-owner {*uid* **|** *name***}**

Specifies the owner of the root directory of the file system. This is used with the **-compat** option, otherwise it is ignored. It may 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.

-group {*gid* **|** *name***}**

Specifies the group owner of the root directory of the file system. This is used with the **-compat** option, otherwise it is ignored. It may 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.

-perms *number*

Specifies the permissions of the root directory of the file system. This is used with the **-compat** option, otherwise it is ignored. It may 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.

-grow *blocks* Specifies the number of 8K blocks that zFS will use as the increment for extension when the **-size** option specifies a size greater than the primary allocation.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm format** command formats a VSAM LDS as a zFS aggregate. All zFS aggregates must be formatted before use (including HFS compatibility mode aggregates). The **zfsadm format** command requires the ZFS PFS to be active on the system. The size of the aggregate is as many 8K blocks as fits in the primary allocation of the VSAM LDS or as specified in the **-size** option. To extend it, use the **zfsadm grow** command. If **-overwrite** is specified, all existing primary and secondary allocations are formatted and the size includes all of that space.

Privilege Required

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

Examples

The following command formats the VSAM LDS as a compatibility mode aggregate.

```
zfsadm format -aggregate omvs.prev.aggr001.lds0001 -compat -owner usera -group audit -perms o750
```

Related Information

Commands:

zfsadm define

Files:

IOEFSPRM

zfsadm grow

Purpose

Makes the physical size of an aggregate larger.

Format

zfsadm grow -aggregate *name* **-size** ***kbytes*** **[-level] [-help]**

Options

-aggregate *name*
: Specifies the aggregate name of the aggregate to be grown. The aggregate name is not case sensitive. It is always translated to upper case.

-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)[3] boundary. If zero is specified, the secondary allocation size will be used.

-level
: Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm grow** command increases the size of an aggregate. The aggregate must be attached. If a non-zero size is specified, a secondary allocation on the VSAM LDS is not required. The aggregate is quiesced before the grow and is unquiesced after the grow completes. This means that any current operations against the aggregate are completed and no new operations begin until after the grow completes. Any mounted file systems are quiesced. An aggregate cannot be made smaller than its current size. The current size of an aggregate can be determined by using the **zfsadm aggrinfo** command. A grow operation fails if no space is available on the volume(s) or when zero is specified for the size and no secondary allocation is specified for the VSAM Linear Data Set.

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

Privilege Required

The issuer must have READ authority to the data set that contains the **IOEFSPRM** file and must be logged in as **root** or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the 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]
```

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

Related Information

Command:

zfsadm aggrinfo

zfsadm help

Purpose

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

Format

zfsadm help [-topic *command*...**] [-level] [-help]**

Options

-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.

-level
: Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

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

READ authority to the data set that contains the **IOEFSPRM** file is required.

Examples

The following command displays the online help entry for the **zfsadm lsfs** command and the **zfsadm lsaggr** command:

```
zfsadm help -topic lsfs lsaggr

zfsadm lsfs: list filesystem information
Usage: zfsadm lsfs [-aggregate <aggregate name>] [{-fast | -long}] [-level] [-help]
zfsadm lsaggr: list aggregates
Usage: zfsadm lsaggr [-level] [-help]
```

Related Information

Command:

zfsadm apropos

zfsadm lsaggr

Purpose

Lists all currently attached aggregates for zFS.

Format

```
zfsadm lsaggr [-level] [-help]
```

Options

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

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

This command displays a separate line for each aggregate. Each line displays the following information:

- The aggregate name
- The aggregate ID number.

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

READ authority to the data set that contains the **IOEFSPRM** file is required.

Examples

The following example shows that five aggregates are attached to the system:

```
zfsadm lsaggr
```

```
OMVS.PRV.AGGR004.LDS0004                    (id=100004)
OMVS.PRV.AGGR003.LDS0002                    (id=100003)
OMVS.PRV.AGGR003.LDS0001                    (id=100002)
OMVS.PRV.AGGR002.LDS0002                    (id=100001)
OMVS.PRV.AGGR001.LDS0001                    (id=100000)
```

Related Information

Command:

zfsadm aggrinfo

File:

IOEFSPRM

zfsadm lsfs

Purpose

Lists all the file systems on a given aggregate or all attached aggregates.

Format

zfsadm lsfs [-aggregate *name***] [-fast | -long] [-level] [-help]**

Options

-aggregate *name*
: Specifies an aggregate name that is used to retrieve file system information. The aggregate name is not case sensitive. It is always translated to upper case. If this option is not specified, the command displays information for all attached aggregates.

-fast
: Causes the output of the command to be shortened to display only the aggregate name if it contains one or more file systems or a message indicating that there are no file systems contained in the aggregate.

-long
: Causes the output of the command to be extended to five lines for each file system found in an aggregate.

-level
: Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm lsfs** command displays information about file systems in an aggregate. The file systems do not need to be mounted to use this command.

The **zfsadm lsfs** command displays the following information for a specified aggregate or all attached aggregates on a system:

- The total number of file systems contained in the aggregate.
- The file system's name (with a **.bak** extension, if appropriate).
- The type (RW for read-write, or BK for backup).
- If it is mounted or not.
- The allocation usage and the quota usage, in kilobytes.
- If the file system is on-line or not.
- The total number of file systems on-line, off-line, busy, and mounted appear at the end of the output for all file systems.

If **-fast** is specified, it only displays the file system names.

If **-long** is specified, the following is displayed:

- The total number of file systems contained in the aggregate.
- The file system's name.
- The file system's ID.
- The type (RW for read-write, or BK for backup).
- If it is mounted or not.
- The state vector of the file system.

- If the file system is on-line or not.
- The allocation limit and allocation usage.
- The quota limit and quota usage.
- The day, date, and time when the file system was created (backed up for a backup file system).
- The day, date, and time when the contents of the file system were last updated (same as the creation time for a backup file system).
- The total number of file systems on-line, off-line, busy and mounted appears at the end of the output for all file systems.

Privilege Required

READ authority to the data set that contains the **IOEFSPRM** file is required.

Examples

The following example displays information for the aggregate **OMVS.PRV.AGGR001.LDS0001**:

```
zfsadm lsfs -aggregate omvs.prv.aggr001.lds0001 -long
```

```
IOEZ00129I Total of 2 file systems found for aggregate OMVS.PRV.AGGR001.LDS0001
OMVS.PRV.FS1 100000,,5 RW (Not Mounted)      states 0x10010005 On-line
    4294967232 K alloc limit;          9 K alloc usage
         25000 K quota limit;          9 K quota usage
    Creation Thu Aug  9 17:17:03 2001
    Last Update Thu Aug  9 17:17:03 2001

OMVS.PRV.FS2 100000,,6 RW (Not Mounted)      states 0x10010005 On-line
    4294967232 K alloc limit;          9 K alloc usage
         45000 K quota limit;          9 K quota usage
    Creation Thu Aug  9 17:26:54 2001
    Last Update Thu Aug  9 17:26:54 2001

Total file systems on-line 2; total off-line 0; total busy 0; total mounted 0
```

Related Information

Commands:

zfsadm create
zfsadm clone

zfsadm lsquota

Purpose

Shows quota information about file systems and aggregates.

Format

```
zfsadm lsquota {-filesystem name | -mfilesystem mount_name} [-aggregate name] [-level] [-help]
```

Options

-filesystem *name*
: Specifies the name of the zFS file system about which quota and usage information is to be displayed. The file system name is case sensitive. If it was specified in upper case when the file system was created, it must be specified in upper case here.

-mfilesystem *mount_name*
: Specifies the UNIX file system name of the file system about which quota and usage information is to be displayed. The file system name is case sensitive. If it was specified in upper case when the file system was mounted, it must be specified in upper case here.

-aggregate *name*
: Specifies the name of the aggregate where the zFS file system name resides. It is specified to qualify the zFS file system name (**-filesystem**) when there are multiple zFS file systems with the same name in different aggregates. The aggregate name is not case sensitive. It is always folded to upper case.

-level
: Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm lsquota** command displays quota and usage information about a file system. The command also provides usage information on the aggregate in which the file system resides. The file system does not need to be mounted to use this command. The aggregate containing the file system must be attached.

The **zfsadm lsquota** command displays the name of the file system, the quota and the quota used (in kilobytes) of the file system, and the percentage of the quota in use. It also displays the information about the percentage of the aggregate in use, the number of kilobytes in use on the aggregate and the number of available kilobytes on the aggregate in which the file system resides. It also reports that the file system is zFS.

The size of a compatibility mode file system is equal to the size of the aggregate on which it resides. Therefore, the size and usage information displayed for the aggregate in the output of the **zfsadm lsquota** command equals the quota and quota usage information of the file system in the aggregate.

This command displays the following information about each specified file system:

- The name of the file system.
- The quota, in kilobytes, of the file system.
- The number of kilobytes of the quota currently in use on the file system.
- The percentage of the quota currently in use on the file system.
- The percentage of available disk space currently in use on the aggregate on which the file system resides.

- The number of kilobytes of disk space in use on the aggregate and the total number of kilobytes on the aggregate on which the file system resides.
- The file system type of the aggregate (zFS).

If the file system quota usage rises above 90% or the aggregate usage rises above 97%, the appropriate percentage is indicated with << and the message <<**WARNING** is displayed after the aggregate usage information at the end of the output line. (The 90% and the 97% are not related to the **FSFULL** and **AGGRFULL** options on **MOUNT** and in the **IOEFSPRM** file. Those are used to determine when to report to the operator.)

Note: Because each compatibility mode aggregate contains a single file system, the information displayed for a compatibility mode aggregate applies to the single file system it houses.

The **zfsadm aggrinfo** command can be used to display the total disk space on an aggregate and the amount currently available.

Every newly created zFS file system has a quota specification. The **zfsadm setquota** command can be used to increase or decrease the quota of a zFS file system. Because the quota of a zFS file system does not represent the amount of physical data space allocated to the file system, it can be larger than the size of the aggregate on which the file system resides. Similarly, the combined quotas of all file systems on an aggregate can be larger than the size of the aggregate. It cannot be changed to smaller than the usage of the file system.

Privilege Required

READ authority to the data set that contains the **IOEFSPRM** file is required.

Examples

The command that follows lists quota and usage information for the file system **OMVS.PRV.FS1**. It also displays the size and usage information for the aggregate that contains this file system.

```
zfsadm lsq OMVS.PRV.FS1

Filesys Name          Quota    Used  Percent Used  Aggregate
OMVS.PRV.FS1          25000       9     0     1 = 1891/177992 (zFS)
```

The following command lists quota and usage information for the zFS file system named **OMVS.PRV.AGGR004.LDS0004**, and size and usage information for the aggregate on which the file system resides. The <<**WARNING** message directs the issuer's attention to the fact that the percentage of the quota in use on the indicated file system is above the warning level of 90% or the aggregate usage is above 97%.

```
zfsadm lsq -f OMVS.PRV.AGGR004.LDS0004

Filesys Name              Quota    Used  Percent Used  Aggregate
OMVS.PRV.AGGR004.LDS0004   1300    1266    97<< 100<< = 1412/1412 (zFS)    <<WARNING
```

Related Information

Commands:

zfsadm aggrinfo
zfsadm lsfs
zfsadm setquota

zfsadm quiesce

Purpose

Specifies that an aggregate and all the file systems contained in it should be quiesced.

Format

zfsadm quiesce {-all | -aggregate *name***} [-level] [-help]**

Options

-all Specifies that all attached aggregates are to be quiesced. Use this option or use **-aggregate**.

-aggregate *name*
Specifies the name of the aggregate that is to be quiesced. The aggregate name is not case sensitive. It is always translated to upper case. An aggregate must be attached to be quiesced. All current activity against the aggregate is allowed to complete but no new activity is started. Any mounted file systems are quiesced.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm quiesce** command is used to temporarily drain activity to the aggregate. During this time:

- No file systems in the aggregate can be created, deleted, renamed, or cloned
- No quotas for file systems contained in the aggregate can be modified
- The aggregate cannot be detached, or grown
- No activity can occur against mounted file systems.

The aggregate can be the target of **lsaggr**, **aggrinfo**, **lsfs** (file systems are indicated as busy).

The aggregate would normally be quiesced prior to backing up the aggregate. After the backup is complete, the aggregate can be unquiesced.

Privilege Required

The issuer must have READ authority to the data set that contains the **IOEFSPRM** file and must be logged in as **root** or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the 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 unquiesce

zfsadm rename

Purpose

Renames a file system.

Format

```
zfsadm rename -oldname oldname -newname newname [-aggregate name][-level] [-help]
```

Options

-oldname *oldname*

Specifies the current zFS file system name of the read-write file system. It is case sensitive.

-newname *newname*

Specifies the new zFS file system name for the read-write file system. The name must be unique within the sysplex (or system, if not in a sysplex), or unique within the aggregate if **allow_duplicate_filesystems** is **on** and it should be indicative of the file system's contents. The following characters can be included in the name of a file system:

- All uppercase and lowercase alphabetic characters (a to z, A to Z)
- All numerals (0 to 9)
- The . (period)
- The - (dash)
- The _ (underscore).

The name can be no longer than 44 characters. This length includes the **.bak** extension, which is added automatically when a read-only or backup version of the file system is created. If you intend to clone this file system, you may want to limit the file system name to 40 characters. Note that the **.bak** extensions are reserved for use with backup zFS file systems, so you cannot specify a file system name that ends with that extension.

Note: The file system name is case sensitive. That is, if you specify the file system name in lower case as the **-newname** on the **zfsadm rename** command, you must specify the file system name in lower case when you mount it. The TSO/E **MOUNT** command translates the file system name to upper case even if it is within quotes. It is not translated to upper case if you specify the file system name on the TSO/E **MOUNT** command within triple quotes. For example, you can specify FILESYSTEM('''lower.case.example''') and the file system name is not translated to upper case. However, you may find it simpler to specify the file system name in upper case on the **zfsadm rename** command.

-aggregate *name*

Specifies the name of the aggregate where the zFS file system name resides. It is specified to qualify the zFS file system name (**-oldname**) when there are multiple zFS file systems with the same name in different aggregates. The aggregate name is not case sensitive. It is always folded to upper case.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

zfsadm rename

Usage

The **zfsadm rename** command changes the name of the read-write file system specified with **-oldname** to the name specified with **-newname**. The name of the read-write file system's backup copy, if any, automatically changes to match. The aggregate that the file system is contained in must be attached. The file system cannot be mounted.

Privilege Required

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

Examples

The following command changes the file system name **OMVS.PRV.FS2** to the file system name **OMVS.PRV.FS9**:

```
zfsadm rename -oldname OMVS.PRV.FS2 -newname OMVS.PRV.FS9

IOEZ00108I File system OMVS.PRV.FS2 renamed to OMVS.PRV.FS9
IOEZ00108I File system OMVS.PRV.FS2.bak renamed to OMVS.PRV.FS9.bak
```

Related Information

Commands:

zfsadm create
zfsadm clone

zfsadm setquota

Purpose

Sets the quota for a filesystem.

Format

```
zfsadm setquota {-filesystem name | -mfilesystem mount_name} -size kbytes [ -aggregate name]
[-level] [-help]
```

Options

-filesystem *name*
Specifies the file system name of the read-write file system whose quota is to be set. The file system name is case sensitive.

-mfilesystem *mount_name*
Specifies the UNIX file system name of the file system which the quota is to be set. The file system name is case sensitive. If it was specified in upper case when the file system was created, it must be specified in upper case here.

-size *kbytes*
Specifies the maximum amount of disk space that all of the files and directories in the read-write file system can occupy. This includes files and directories in the read-write version of the file system that are actually pointers to disk blocks in the backup version of the file system. Specify the value in 1-kilobyte blocks. (A value of 1024 kilobytes is 1 megabyte.) The minimum specification is 128 (that is, 128K bytes).

-aggregate *name*
Specifies the name of the aggregate where the zFS file system name resides. It is specified to qualify the zFS file system name (**-filesystem**) when there are multiple zFS file systems with the same name in different aggregates. The aggregate name is not case sensitive. It is always folded to upper case.

-level
Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help
Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm setquota** command sets the quota limit for a read-write zFS file system. (It cannot be used to set the quota for a non-zFS file system or for a backup zFS file system.) The file system whose quota is to be set is indicated by specifying the file system name with the **-filesystem** option.

Quota refers to the amount of disk space occupied by all of the files and directories in the read-write version of the file system. This includes files and directories in the read-write version of the file system that are actually pointers to disk blocks in the backup version of the file system. Do not confuse quota with allocation; the latter identifies the amount of disk space occupied by the data that a file system actually houses; excluding those files and directories that are pointers to disk blocks in the backup version of the file system.

This command increases or decreases a file system's quota to be the number of kilobytes specified with the **-size** option. Because it does not represent the amount of physical data the file system contains, a file system's quota can be larger than the size of the aggregate on which it resides. Similarly, the sum of the quotas of all file systems on an aggregate can exceed the size of the aggregate.

The **zfsadm lsfs** and **zfsadm lsquota** commands display, among other things, the current quota for a file system.

Privilege Required

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

Examples

The following command sets the quota for the file system named **OMVS.PRV.FS1** to be 15,000 kilobytes:

```
zfsadm setquota -filesystem OMVS.PRV.FS1 -size 15000
```

```
zfsadm lsquota OMVS.PRV.FS1

Filesys Name          Quota    Used  Percent Used  Aggregate
OMVS.PRV.FS1          15000       9     0     1 = 1907/177992 (zFS)
```

Related Information

Commands:

- **zfsadm lsfs**
- **zfsadm lsquota**

zfsadm unquiesce

Purpose

Makes an aggregate (and all the file systems contained in the aggregate) available to be accessed.

Format

zfsadm unquiesce {-all | -aggregate *name***} [-level] [-help]**

Options

-all Specifies that all attached aggregates are to be unquiesced. Use this option or use **-aggregate**.

-aggregate *name*
Specifies the name of the aggregate that is to be unquiesced. The aggregate name is not case sensitive. It is always translated to upper case. An aggregate must be attached to be unquiesced. All current activity against the aggregate is allowed to resume. Any mounted file systems is unquiesced.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

Usage

The **zfsadm unquiesce** command allows activity that has been suspended by **zfsadm quiesce**, to be resumed.

The aggregate would normally be quiesced prior to backing up the aggregate. After the backup is complete, the aggregate can be unquiesced.

Privilege Required

The issuer must have READ authority to the data set that contains the **IOEFSPRM** file and must be logged in as **root** or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the 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

Command:
zfsadm quiesce

Chapter 12. zFS data sets

The following data sets are used during zFS processing.

IOEFSPRM

Purpose

This file lists the processing options for the ZFS PFS and the definitions of the multi-file system aggregates. There is no mandatory information in this file, therefore it is not required. The options all have defaults. Aggregates can all be compatibility mode aggregates (which do not need definitions). Multi-file system aggregates can be attached by using the **zfsadm attach** command. They do not need definitions in IOEFSPRM to be attached using **zfsadm attach**. However, if you need to specify any options (for tuning purposes, for example) or if you want to have any multi-file system aggregates automatically attached when ZFS is started, you need to have an IOEFSPRM file.

The location of the IOEFSPRM file is specified by the IOEZPRM DD statement in the ZFS PROC. The IOEFSPRM file is normally a PDS member, so the IOEZPRM DD might look like the following:

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

If you need to have separate IOEFSPRM files and you want to share the ZFS PROC in a sysplex, you can use a system variable in the ZFS PROC so that it points to different IOEFSPRM files. The IOEZPRM DD might look like the following:

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

Your IOEFSPRM file might reside in SYS4.PVT.SY1.PARMLIB(IOEFSPRM) on system SY1; in SYS4.PVT.SY2.PARMLIB(IOEFSPRM) on system SY2; etc.

| 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** would result in
| USERA.SY1.ZFS.MSGOUT on system SY1 and **define_aggr cluster(USERA.&SYSNAME..AGGR001)**
| would result in **define_aggr cluster(USERA.SY1.AGGR001)** on system SY1. Each system has a single
| (possibly shared) **IOEFSPRM** file.

Any line beginning with # or * is considered a comment. The text in the IOEFSPRM file is case insensitive. Any option or value can be upper or lower case. Blank lines are allowed. You should not have any sequence numbers in the IOEFSPRM file.

Usage

The following options are used as processing options for the ZFS PFS:

adm_threads

Specifies the number of threads defined to handle pfsctl or mount requests.

Default Value	10
Expected Value	A number between 1 and 256.
Example	`adm_threads=20`

auto_attach

Controls whether aggregates defined and listed in the **IOEFSPRM** file are attached by default when ZFS is started (or restarted). When the value is on, you can add new multi-file system aggregates (with the define_aggr option) to the IOEFSPRM file and they are attached automatically the next time ZFS is started.

Default Value	On
Expected Value	On or off.
Example	`auto_attach=off`

| **user_cache_readahead**

| Specifies whether ZFS does read ahead for sequential access. Normally, this should be left on.
| Readahead can be disabled for a particular file system that has mostly random access by
| specifying NOREADAHEAD in the MOUNT PARM.

| **Default Value** on
| **Expected Value** on or off.
| **Example** `user_cache_readahead=off`

user_cache_size
Specifies the size, in bytes, of the cache used to contain file data. You can also specify a **fixed** option which indicates that the pages are permanently fixed for performance. Note, the **fixed** option reserves real storage for usage by ZFS only.
Default Value 256M
Expected Value A number between 10M and 65536M (64G). A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively.
Example `user_cache_size=64M,fixed`

meta_cache_size
Specifies the size of the cache used to contain metadata. You can also specify a fixed option which indicates that the pages are permanently fixed for performance. Note, the **fixed** option reserves real storage for usage by ZFS only.
Default Value 32M
Expected Value A number between 1M and 1024M. A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively.
Example `meta_cache_size=64M,fixed`

| **metaback_cache_size**
| Specifies the size of the backing cache used to contain metadata. This resides in a data space and can optionally be used to extend the size of the metadata cache. You can also specify a fixed option which indicates that the pages are permanently fixed for performance. Note, the **fixed** option reserves real storage for usage by ZFS only.
| **Default Value** None
| **Expected Value** A number between 1M and 2048M. A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively.
| **Example** `metaback_cache_size=64M,fixed`

log_cache_size
Specifies the size of the cache used to contain buffers for log file pages. You can also specify a fixed option which indicates that the pages are permanently fixed for performance. Note, the **fixed** option reserves real storage for usage by ZFS only.
Default Value 64M
Expected Value A number between 10M and 1536M. A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively.
Example `log_cache_size=32M,fixed`

sync_interval
Specifies the number of seconds between syncs.
Default Value 30
Expected Value A number between 11 and 21474836.
Example `sync_interval=60`

vnode_cache_size
Specifies the size of the internal zFS vnode cache.
Default Value 8192 (will grow if z/OS UNIX needs more than this number)
Expected Value A number between 32 and 1048576.
Example `vnode_cache_size=16384`

nbs Controls whether new block security is globally on by default or off by default for any aggregate. New block security refers to the guarantee made when a system crashes. Refer to "zfsadm attach" on page 71 for an explanation of the **nbs** option.
Default Value Off
Expected Value On or off.

Example `nbs=on`

aggrfull
Specifies the threshold and increment for reporting aggregate full error messages to the operator.
Default Value Off
Expected Value Two numbers between 1 and 99 within parentheses separated by a comma.
Example `aggrfull(85,5)`

fsfull Specifies the threshold and increment for reporting file system quota full error messages to the operator.
Default Value Off
Expected Value Two numbers between 1 and 99 within parentheses separated by a comma.
Example `fsfull(85,5)`

tran_cache_size
Specifies the initial number of transactions in the transaction cache.
Default Value 2000
Expected Value A number between 200 and 10485760.
Example `tran_cache_size=4000`

msg_input_dsn
Specifies the name of a data set containing translated zFS messages. It is specified when the installation uses non-English messages. (When you use English messages, you should not specify this option.) It is read when zFS is started (or restarted). Currently, Japanese messages are supported.
Default Value None
Expected Value The name of a data set containing translated zFS messages.
Example `msg_input_dsn=USERA.SIOEMJPN`

aggrgrow
Specifies whether aggregates can be dynamically extended when they become full. The aggregate (that is, the VSAM Linear Data Set) must have a secondary allocation specified to be dynamically extended and there must be space on the volume(s). This global value can be overridden in the define_aggr option or the zfsadm attach command for multi-file system aggregates and on the MOUNT command for compatibility mode aggregates.
Default Value Off
Expected Value On or off.
Example `aggrgrow=on`

fsgrow
Specifies whether file systems in multi-file system aggregates can have their quota be dynamically extended when they reach their quota limit. The first number specifies the number of k-bytes that the file system quota should be increased. The second number specifies the number of times the quota should be extended. This global value can be overridden on the MOUNT command for multi-file system aggregates.
Default Value Off
Expected Value Two numbers between 0 and 2147483648 within parentheses and separated by a comma.
Example `fsgrow=(100,16)`

allow_duplicate_filesystems
Specifies whether zFS allows file systems in different aggregates with the same file system name.
Default Value Off
Expected Value On or off.
Example `allow_duplicate_filesystems=on`

The following option is used to define multi-file system aggregates so that they are attached at ZFS start (or restart):

define_aggr

Defines a multi-file system aggregate, its corresponding data set name (which is the same as the aggregate name), and any processing suboptions for that aggregate. The **define_aggr** option can be contained on multiple lines and is complete when the next option is encountered or the end of the file is reached. Suboptions include:

R/O or R/W	**R/O** specifies that the aggregate should be opened in read-only mode. A **R/O** aggregate means that all file systems in the aggregate are read-only and can only be mounted read-only. This allows sharing of an aggregate among multiple systems. **R/W** specifies that the aggregate should be opened in read-write mode. **R/W** is the default.
attach or noattach	Indicates whether this aggregate is attached automatically when ZFS is started (or restarted). Aggregates that are not automatically attached must be attached with the **zfsadm attach** command. The default is the global **auto_attach** option (refer to page 106).
nbs or nonbs	Indicates if new block security algorithms should be used for this aggregate. The default is the global **nbs** option (refer to page 107).
aggrfull	Specifies the threshold and increment for reporting aggregate full messages for this aggregate to the operator. The default is the global **aggrfull** option (refer to page 108).
aggrgrow or noaggrgrow	Indicates whether the multi-file system aggregate should dynamically grow when it runs out of physical space. The aggregate (that is, the VSAM Linear Data Set) must have secondary allocation specified and there must be space on the volume(s). The default is the global **aggrgrow** option (refer to page 108).
cluster	Specifies the VSAM Linear Data Set Cluster name. This is also the aggregate name. This option is required.
Example	`define_aggr R/W attach nonbs aggrfull(85,5) aggrgrow cluster(OMVS.PRV.AGGR0001.LDS0001)`

The following options are used during debugging of the ZFS PFS:

debug_settings_dsn

Specifies the name of a data set containing debug classes to enable when zFS starts up. It is read when zFS is started (or restarted).

Default Value	None
Expected Value	The name of a data set containing debug classes to enable.
Example	`debug_settings_dsn=USERA.ZFS.DEBUG.INPUT(FILE1)`

trace_dsn

Contains the output of any operator **MODIFY ZFS,TRACE,PRINT** commands or the trace output if the ZFS PFS abends. Each trace output creates a member in the PDSE. Traces that come from the ZFS PFS kernel have member names of ZFSKNTnn. nn starts with 01 and increments for each trace output. nn is reset to 01 when ZFS is started (or restarted). Refer to Chapter 9, “Performance and debugging” on page 31. This is not a required parameter.

Default Value	None
Expected Value	The name of a PDSE data set.
Example	`trace_dsn=USERA.ZFS.TRACE.OUT`

IOEFSPRM

trace_table_size

Specifies the size, in bytes, of the internal trace table. This is the size of the wrap-around trace table in the ZFS address space that is used for internal tracing that is always on. The trace can be sent to the **trace_dsn** by using the operator **MODIFY ZFS,TRACE,PRINT** command.

Default Value	512K
Expected Value	A positive number. A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively.
Example	`trace_table_size=1M`

msg_output_dsn

Specifies the name of a data set that contains any output messages that come from the ZFS PFS. Refer to Chapter 9, "Performance and debugging" on page 31. 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`

The next two options are obsolete in z/OS Version 1 Release 3 and later and are ignored. **storage_details** is always on and output from the MODIFY ZFS,QUERY, STORAGE,DETAILS goes into the **msg_output_dsn** data set if **msg_output_dsn** is specified. **storage_details_dsn** is not used.

storage_details

Indicates whether or not the ZFS internal storage tracking mechanisms are active. The results can be sent to the **storage_details_dsn** with the operator **MODIFY ZFS,QUERY,STORAGE,DETAILS** command.

Default Value	Off
Expected Value	On or off.
Example	`storage_details=on`

storage _details_dsn

Indicates where the storage map is written if **storage_details** is on and the operator **MODIFY ZFS,QUERY,STORAGE,DETAILS** command is run.

Default Value	None
Expected Value	The name of a data set.
Example	`storage_details_dsn=usera.zfs.storage.output(file1)`

Examples

The following IOEFSPRM sample file lists every program option and includes some sample multi-file system aggregate definitions.

```
| + + + + Beginning of sample file + + + +
|
| ********************************************************************
| * zSeries File System (zFS) Sample Parameter File:  ioefsprm
| * For a description of these and other zFS parameters, refer to the
| * zSeries File System Administration, SC24-5989.
| * 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.
| *  2. An asterisk in column 1 identifies a comment line.
| *  3. A parameter specification must begin in column 1.
| ********************************************************************
| * The following msg_input_dsn parameter is ONLY required if the optional
| * NLS feature (e.g J0H232J) is installed. The parameter specifies the
| * message input data set containing the NLS message text which is
| * supplied by the NLS feature. If this parameter is not specified or if
| * the data set is not found, English language messages will be generated
| * by zFS. You must delete the * from a line to activate the parameter.
| ********************************************************************
| * msg_input_dsn=data.set.name
```

```
|  ********************************************************************
|  * 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
|  *auto_attach=ON
|  *user_cache_size=256M
|  *log_cache_size=12M
|  *sync_interval=45
|  *vnode_cache_size=5000
|  *nbs=off
|  *fsfull(85,5)
|  *aggrfull(90,5)
|  *aggrgrow
|  ********************************************************************
|  * 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.
|  ********************************************************************
|  *trace_dsn=data.set.name
|  *debug_settings_dsn=data.set.name(membername)
|  *trace_table_size=32M
|  *msg_output_dsn=data.set.name
|  *storage_details=ON
|  *storage_details_dsn=data.set.name
```

Chapter 13. zFS application programming interfaces

Note: This chapter contains programming interface information.

This chapter describes the ZFS Application Programming Interface (API), **pfsctl** (BPX1PCT). It describes the ZFS commands: ZFSCALL_AGGR (0x40000005), ZFSCALL_FILESYS (0x40000004) and ZFSCALL_CONFIG (0x40000006) and their subcommands. These APIs are used to manage zFS aggregates and file systems and to query and set configuration options.

pfsctl (BPX1PCT)

Purpose

The **pfsctl** (BPX1PCT) application programming interface is used to send physical file system specific requests to a physical file system. It is documented in a general manner in the *z/OS: UNIX System Services Assembler Callable Services Reference*. ZFS is a physical file system and supports several (ZFS specific) pfsctl functions. These are documented in this section.

Format

```
BPX1PCT (File_system_type,
         Command,
         Argument_Length,
         Argument,
         Return_value,
         Return_code,
         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 three major ZFS commands:

- ZFSCALL_AGGR (0x40000005)
- ZFSCALL_FILESYS (0x40000004)
- ZFSCALL_CONFIG (0x40000006)

Each of these commands has a set of subcommands. The general format of the **Argument** for all subcommands is:

```
Subcommand operation code     int
Parameter0                    int
Parameter1                    int
Parameter2                    int
Parameter3                    int
Parameter4                    int
Parameter5                    int
Parameter6                    int
Buffer[n]                     char[n]
```

where *n* depends on the particular subcommand.

Argument_Length
An integer that contains the length of the argument.

Argument
A structure that has the pfsctl parameters followed by the subcommand parameters.

The definitions of the FS_ID, the FS_STATUS and the FILESYS_DATA have padding characters generated by the compiler. These have now been made explicit in the examples.

Return_value
An integer that contains 0 if the request is successful or -1 if it is not successful.

Return_Code
An integer in which the return code is stored. Refer to the *z/OS: UNIX System Services Messages and Codes* document for these codes.

Reason_Code
An integer in which the reason code is stored. If this code is of the form 0xEFnnxxxx, refer to the

z/OS: Distributed File Service Messages and Codes document. Otherwise, refer to the *z/OS: UNIX System Services Messages and Codes* document.

Usage

There are three major commands: ZFSCALL_AGGR (0x40000005) and its subcommands, ZFSCALL_FILESYS (0x40000004) and its subcommands and ZFSCALL_CONFIG (0x40000006) and its subcommands.

Aggregate operations

The Aggregate operation command code is ZFSCALL_AGGR (0x40000005). The following subcommands and their subcommand codes are supported:

- Attach Aggregate (105)
- Create File System (131)
- Define Aggregate (139)
- Delete File System (136)
- Detach Aggregate (104)
- Format Aggregate (134)
- Grow Aggregate (129)
- List Aggregate Status (137)
- List Attached Aggregate Names (135)
- List File System Names (138)
- List File System Names (Version 2) (144)
- Quiesce Aggregate (132)
- Unquiesce Aggregate (133).

File System operations

The File System operation command code is ZFSCALL_FILESYS (0x40000004). The following subcommands and their subcommand codes are supported:

- Clone File System (143)
- List File System Status (142)
- Rename File System (140)
- Set File System Quota (141).

Configuration operations

The File System operation command code is ZFSCALL_CONFIG (0x40000006). The following subcommands and their subcommand codes are supported:

- Query adm_threads setting (180)
- Query aggrfull setting (181)
- Query aggrfull setting (182)
- Query allow_duplicate_filesystems setting (203)
- Query auto_attach setting (183)
- Query debug_settings_dsn setting (186)
- Query fsfull setting (187)
- Query fsgrow setting (188)
- Query log_cache_size setting (193)
- Query meta_cache_size setting (198)
- Query metaback_cache_size setting (199)
- Query msg_input_dsn setting (200)
- Query msg_output_dsn setting (201)
- Query nbs setting (202)
- Query sync_interval setting (205)
- Query trace_dsn setting (206)
- Query trace_table_size setting (207)
- Query tran_cache_size setting (208)
- Query user_cache_readahead setting (209)
- Query user_cache_size setting (210)

- Query vnode_cache_size setting (212)
- Set adm_threads (150)
- Set aggrfull (158)
- Set aggrgrow (171)
- Set allow_duplicate_filesystem (173)
- Set fsfull (157)
- Set fsgrow (172)
- Set log_cache_size (153)
- Set meta_cache_size (152)
- Set metaback_cache_size (163)
- Set msg_output_dsn (161)
- Set nbs (156)
- Set sync_interval (154)
- Set trace_dsn (159)
- Set tran_cache_size (160)
- Set user_cache_readahead (162)
- Set user_cache_size (151)
- Set vnode_cache_size (155).

Attach Aggregate

Purpose

The Attach Aggregate subcommand call is an aggregate operation that attaches a multi-file system aggregate to a system. This makes the aggregate and all its file systems known to the ZFS Physical File System running on that system.

Format

```
syscall_parmlist
   opcode                         105      AGOP_ATTACH_PARMDATA
   parms[0]                       offset to AGGR_ID
   parms[1]                       offset to AGGR_ATTACH
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
AGGR_ID
   aid_eye                        char[4]    "AGID"
   aid_len                        char       sizeof(AGGR_ID)
   aid_ver                        char       1
   aid_name                       char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                   char[33]   0
AGGR_ATTACH
   at_eye                         char[4]    "AGAT"
   at_len                         short      sizeof(AGGR_ATTACH)
   at_ver                         char       1
   at_res1                        int        0
   at_threshold                   char       90
   at_increment                   char       5
   at_flags                       char       0x80
     ATT_MONITOR                    0x80    Monitor aggregate full
     ATT_RO                         0x40    Attach aggregate as read-only
     ATT_NBS                        0x20    Use New Block Security
     ATT_NONBS                      0x10    Do not use new block security
     ATT_GROW                       0x04    Allow dynamic grow
     ATT_NOGROW                     0x02    Disallow dynamic grow
   at_res2                        char      0
   at_reserved                    int[64]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EEXIST        Aggregate already attached
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  EPERM         Permission denied to perform request

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

This function is used to attach multi-file system aggregates. Compatibility mode aggregates are attached during mount, so a separate attach is not necessary.

ATT_NBS and ATT_NONBS are mutually exclusive. If neither is specified, the default is the **nbs** setting in the **IOEFSPRM** file. Refer to the “zfsadm attach” on page 71 for a description of the **nbs** parameter.

Attach Aggregate

| 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 13 and “Dynamically growing a multi-file system aggregate” on page 28 for a description of dynamic
| grow.

`at_threshold` and `at_increment` are ignored unless ATT_MONITOR is set.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

Delete 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_ATTACH_PARMDATA  105

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform            */
  int parms[7];             /* Specific to type of operation,       */
                            /* provides access to the parms         */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
  char aid_eye[4];                    /* Eye Catcher               */
#define AID_EYE "AGID"
  char aid_len;                       /* Length of this structure  */
  char aid_ver;                       /* Version                   */
#define AID_VER_INITIAL 1             /* Initial version           */
  char aid_name[ZFS_MAX_AGGRNAME+1];  /* aggr name, null terminated */
  char aid_reserved[33];              /* Reserved for the future   */
} AGGR_ID;

typedef struct aggr_attach_t
{
  char  at_eye[4];                    /* Eye catcher               */
#define AT_EYE "AGAT"
  short at_len;                       /* Length of structure       */
  char  at_ver;                       /* Structure version         */
#define AT_VER_INITIAL  1             /* Version 1                 */
  char  at_res1;                      /* Reserved for internal use  */
  char  at_threshold;                 /* Threshold for monitoring  */
  char  at_increment;                 /* Increment                 */
  char  at_flags;                     /* Processing flags          */
#define ATT_MONITOR   0x80            /* aggrfull monitoring should */
                                      /* be used                   */
#define ATT_RO        0x40            /* aggr should be attached ro */
```

```
#define ATT_NBS       0x20            /* aggr should be attached    */
                                      /* with full NBS              */
#define ATT_NONBS     0x10            /* aggr should be attached    */
                                      /* with no NBS                */
#define ATT_GROW      0x04            /* allow dynamic growing      */
#define ATT_NOGROW    0x02            /* disallow dynamic growing   */

  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;
} ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";   /* aggregate name to attach */

struct parmstruct myparmstruct;

AGGR_ID *idp = &(myparmstruct.aggr_id);
AGGR_ATTACH *atp = &(myparmstruct.myaggr);

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;
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(atp,0,sizeof(AGGR_ATTACH));             /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.aggr_id,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);
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         */

     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\n", aggrname);
     printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
```

```
     return bpxrc;
     }
   else /* Return from attach was successful  */
     {
     printf("Aggregate %s attached successfully\n",aggrname);
     }
return 0;
}
```

Clone File System

Purpose

The Clone File System subcommand call is a file system operation that creates (or replaces) a backup file system from the specified read-write file system. This is referred to as cloning a file system. The backup file system is stored in the same aggregate as the read-write file system.

| You can use an FS_ID or an FS_ID2 as input.

Format

```
syscall_parmlist
   opcode                        143      FSOP_CLONE_PARMDATA
   parms[0]                      offset to FS_ID or FS_ID2
   parms[1]                      0
   parms[2]                      0
   parms[3]                      0
   parms[4]                      0
   parms[5]                      0
   parms[6]                      0
FS_ID or FS_ID2
   fsid_eye                      char[4]    "FSID"
   fsid_len                      short      sizeof(FS_ID)
   fsid_ver                      char       1
   fsid_res1                     char       0
   fsid_res2                     char       0
   fsid_id
     high                        unsigned long  0
     low                         unsigned long  0
   fsid_aggrname                 char[45]   0
   fsid_name                     char[45]   "OMVS.PRV.FS3"
   fsid_reserved                 char[32]   0
   fsid_reserved2                char[2]    0
FS_ID2 or FS_ID
   fsid_eye                      char[4]    "FSID"
   fsid_len                      short      sizeof(FS_ID2)
   fsid_ver                      char       2
   fsid_res1                     char       0
   fsid_res2                     char       0
   fsid_id
     high                        unsigned long  0
     low                         unsigned long  0
   fsid_aggrname                 char[45]   0
   fsid_name                     char[45]   "OMVS.PRV.FS3"
   fsid_mtname                   char[45]   0
   fsid_reserved                 char[49]   0


Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate containing file system is quiesced
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request
  EROFS         Aggregate is attached as read only

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The aggregate containing the read-write file system to be cloned must be attached. The backup file system name is the same as the read-write file system's name with **.bak** appended. After the clone operation, the backup file system can be mounted read-only.

After the backup file system is mounted read-only, users can access this point-in-time copy of the data until the backup file system is deleted or the read-write file system is recloned.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

Delete File System

Restrictions

The aggregate cannot be attached as read-only. The file system name of the read-write file system to be cloned must be less than or equal to 40 characters. If the backup file system already exists, it cannot be mounted. The aggregate containing the read-write file system cannot be quiesced.

Examples

Example 1 - Using FS_ID

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS   0x40000004
#define FSOP_CLONE_PARMDATA  143

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                   /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                      /* Length of this structure */
  char fsid_ver;                      /* Version */
#define FSID_VER_INITIAL 1            /* Initial version            */
  char fsid_res1;                     /* Reserved.                  */
  char fsid_res2;                     /* Reserved.                  */
  hyper fsid_id;                      /* Internal identifier        */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
  char fsid_reserved[32];             /* Reserved for the future  */
```

```
  char fsid_reserved2[2];              /* Reserved for the future  */
} FS_ID;

struct parmstruct
{
 syscall_parmlist myparms;
 FS_ID fsid;
 } ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char filesystemname[45] = "OMVS.PRV.FS3";

struct parmstruct myparmstruct;

FS_ID *idp = &(myparmstruct.fsid);

myparmstruct.myparms.opcode = FSOP_CLONE_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idp,0,sizeof(FS_ID));                  /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
myparmstruct.fsid.fsid_len = sizeof(FS_ID);
myparmstruct.fsid.fsid_ver = FSID_VER_INITIAL;
strcpy(myparmstruct.fsid.fsid_name,filesystemname);

     BPX1PCT("ZFS     ",
             ZFSCALL_FILESYS,          /* Aggregate operation     */
             sizeof(myparmstruct),     /* Length of Argument      */
             (char *) &myparmstruct,   /* Pointer to Argument     */
             &bpxrv,                   /* Pointer to Return_value */
             &bpxrc,                   /* Pointer to Return_code  */
             &bpxrs);                  /* Pointer to Reason_code  */

if (bpxrv < 0)
     {
     printf("Error cloning file system %s\n",filesystemname);
     printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
     return bpxrc;
     }
   else /* Return from clone file system was successful  */
     {
     printf("File system %s cloned successfully\n",filesystemname);
     }
return 0;
}
```

Example 2 - Using FS_ID2

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS   0x40000004
#define FSOP_CLONE_PARMDATA  143
```

Clone File System

```
| typedef struct syscall_parmlist_t {
|   int opcode;               /* Operation code to perform             */
|   int parms[7];             /* Specific to type of operation,        */
|                             /* provides access to the parms          */
|                             /* parms[4]-parms[6] are currently unused*/
| } syscall_parmlist;
|
| typedef struct hyper { /* unsigned 64 bit integers */
|     unsigned long high;
|     unsigned long low;
| } hyper;
|
| #define ZFS_MAX_AGGRNAME 44
| #define ZFS_MAX_FSYSNAME 44
|
| typedef struct fs_id_t {
|   char fsid_eye[4];                    /* Eye catcher */
| #define FSID_EYE "FSID"
|   char fsid_len;                       /* Length of this structure */
|   char fsid_ver;                       /* Version */
| #define FSID_VER_INITIAL 1             /* Initial version            */
|   char fsid_res1;                      /* Reserved.                  */
|   char fsid_res2;                      /* Reserved.                  */
|   hyper fsid_id;                       /* Internal identifier        */
|   char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
|   char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
|   char fsid_reserved[32];              /* Reserved for the future  */
|   char fsid_reserved2[2];              /* Reserved for the future  */
| } FS_ID;
|
| typedef struct fs_id2_t {
|   char fsid_eye[4];                    /* Eye catcher */
| #define FSID_EYE "FSID"
|   char fsid_len;                       /* Length of this structure */
|   char fsid_ver;                       /* Version */
|   char fsid_res1;                      /* Reserved.                  */
|   char fsid_res2;                      /* Reserved.                  */
|   hyper fsid_id;                       /* Internal identifier        */
| #define FSID_VER_2       2              /* Second version  */
|   char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */ /*@D10997MN*/
|   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;
|  FS_ID2 fsid;
|  } ;
|
| int main(int argc, char **argv)
| {
| int bpxrv;
| int bpxrc;
| int bpxrs;
| char filesystemname[45] = "OMVS.PRV.FS3";
|
| struct parmstruct myparmstruct;
|
| FS_ID2 *idp = &(myparmstruct.fsid);
|
| myparmstruct.myparms.opcode = FSOP_CLONE_PARMDATA;
| myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
| myparmstruct.myparms.parms[1] = 0;
| myparmstruct.myparms.parms[2] = 0;
| myparmstruct.myparms.parms[3] = 0;
```

```
|  myparmstruct.myparms.parms[4] = 0;
|  myparmstruct.myparms.parms[5] = 0;
|  myparmstruct.myparms.parms[6] = 0;
|
|  memset(idp,0,sizeof(FS_ID2));                  /* Ensure reserved fields are 0 */
|
|  memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
|  myparmstruct.fsid.fsid_len = sizeof(FS_ID2);
|  myparmstruct.fsid.fsid_ver = FSID_VER_2;
|  strcpy(myparmstruct.fsid.fsid_name,filesystemname);
|
|       BPX1PCT("ZFS     ",
|               ZFSCALL_FILESYS,          /* Aggregate operation     */
|               sizeof(myparmstruct),     /* Length of Argument      */
|               (char *) &myparmstruct,   /* Pointer to Argument     */
|               &bpxrv,                   /* Pointer to Return_value */
|               &bpxrc,                   /* Pointer to Return_code  */
|               &bpxrs);                  /* Pointer to Reason_code  */
|
|  if (bpxrv < 0)
|       {
|       printf("Error cloning file system %s\n",filesystemname);
|       printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
|       return bpxrc;
|       }
|     else /* Return from clone file system was successful  */
|       {
|       printf("File system %s cloned successfully\n",filesystemname);
|       }
|  return 0;
|  }
```

Create File System

Purpose

The Create File System subcommand call is an aggregate operation that creates a new read-write file system in a multi-file system aggregate on a system.

| You can use an FS_ID or an FS_ID2 as input.

Format

```
syscall_parmlist
   opcode                         131      AGOP_CREATEFILESYS_PARMDATA
   parms[0]                       offset to FS_ID or FS_ID2
   parms[1]                       offset to FILESYS_DATA
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
FS_ID or FS_ID2
   fsid_eye                       char[4]    "FSID"
   fsid_len                       char       sizeof(FS_ID)
   fsid_ver                       char       1
   fsid_res1                      char       0
   fsid_res2                      char       0
   fsid_id                        hyper
      high                          long     0
      low                           long     0
   fsid_aggrname                  char[45]   "OMVS.PRV.AGGR001.LDS0001"
   fsid_name                      char[45]   "OMVS.PRV.FS3"
   fsid_reserved                  char[32]   0
   fsid_reserved2                 char[2]    0
FS_ID2 or FS_ID
   fsid_eye                       char[4]    "FSID"
   fsid_len                       short      sizeof(FS_ID2)
   fsid_ver                       char       2
   fsid_res1                      char       0
   fsid_res2                      char       0
   fsid_id
     high                         unsigned long  0
     low                          unsigned long  0
   fsid_aggrname                  char[45]   0
   fsid_name                      char[45]   "OMVS.PRV.FS3"
   fsid_mtname                    char[45]   0
   fsid_reserved                  char[49]   0
FILESYS_DATA
   fd_eye                         char[4]    "FSDT"
   fd_len                         short      sizeof(FILESYS_DATA)
   fd_ver                         char       1
   fd_flags                       char
     FD_OWNER_SPECIFIED             0x80
     FD_PERMS_SPECIFIED             0x40
   fd_owner                       int        612
   fd_group                       int        10
   fd_perms                       int        0755
   fd_quotah                      short      0
   fd_reserved1                   char[2]    0
   fd_quota                       long       5000
   fd_reserved                    char[64]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code
```

```
  EBUSY         Aggregate containing file system is quiesced
  EXIST         File system already exists
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  EPERM         Permission denied to perform request
  EROFS         Aggregate is attached as read only

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The aggregate that is to contain the new read-write file system must be attached. The file system name can be no longer than 44 characters. If this file system is to be cloned, a file system name extension of **.bak** will be added to the end of the read-write file system name to create the backup file system name. If you intend to clone this read-write file system, you may want to limit the read-write file system name to 40 characters.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

Clone File System
Delete File System

Restrictions

The aggregate cannot be quiesced or attached as read-only. You cannot create a file system that already exists. You cannot create a file system that ends with **.bak.**

Examples

Example 1 - Using FS_ID

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR   0x40000005
#define AGOP_CREATEFILESYS_PARMDATA  131

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                    /* Eye catcher */
```

Create File System

```
#define FSID_EYE "FSID"
  char fsid_len;                        /* Length of this structure */
  char fsid_ver;                        /* Version */
#define FSID_VER_INITIAL 1              /* Initial version           */
  char fsid_res1;                       /* Reserved.                 */
  char fsid_res2;                       /* Reserved.                 */
  hyper fsid_id;                        /* Internal identifier       */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
  char fsid_reserved[32];               /* Reserved for the future  */
  char fsid_reserved2[2];               /* Reserved for the future  */
} FS_ID;

typedef struct filesys_t {
  char fd_eye[4];                      /* eye catcher               */
#define FD_EYE "FSDT"
  short fd_len;                        /* Length                    */
  char  fd_ver;                        /*                           */
#define FD_VER_INITIAL 1               /* Initial version           */
  char  fd_flags;                      /* Flag bits                 */
#define FD_OWNER_SPECIFIED 0x80        /* Owner & group specified   */
#define FD_PERMS_SPECIFIED 0x40        /* Permissions specified     */
  int   fd_owner;                      /* Owner id for root         */
                                       /* filesystem                */
  int   fd_group;                      /* Group id for root         */
                                       /* filesystem                */
  int   fd_perms;                      /* Permissions for root      */
                                       /* filesystem                */
#define FD_DEFAULT_PERMS 0755          /* Default permissions if not specified  */
  short fd_quotah;                     /* High portion of quota, in  */
                                       /* K bytes                   */
  char  fd_reserved1[2];               /* Reserved bytes            */
  long  fd_quota;                      /* Low portion of quota in   */
                                       /* K bytes                   */
  char  fd_reserved[64];               /* More reserved bytes       */
} FILESYS_DATA;

struct parmstruct
{
 syscall_parmlist myparms;
 FS_ID fsid;
 FILESYS_DATA myfilesystem;
} ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char filesystemname[45] = "OMVS.PRV.FS3";
char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";

struct parmstruct myparmstruct;

FS_ID *idp = &(myparmstruct.fsid);
FILESYS_DATA *fdp = &(myparmstruct.myfilesystem);

myparmstruct.myparms.opcode = AGOP_CREATEFILESYS_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID);
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idp,0,sizeof(FS_ID));                  /* Ensure reserved fields are 0 */
```

```
memset(fdp,0,sizeof(FILESYS_DATA));           /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
myparmstruct.fsid.fsid_len = sizeof(FS_ID);
myparmstruct.fsid.fsid_ver = FSID_VER_INITIAL;
strcpy(myparmstruct.fsid.fsid_aggrname,aggrname);
strcpy(myparmstruct.fsid.fsid_name,filesystemname);

memcpy(&myparmstruct.myfilesystem.fd_eye[0], FD_EYE, 4);
myparmstruct.myfilesystem.fd_len = sizeof(FILESYS_DATA);
myparmstruct.myfilesystem.fd_ver = FD_VER_INITIAL;
myparmstruct.myfilesystem.fd_flags = FD_OWNER_SPECIFIED | FD_PERMS_SPECIFIED;
myparmstruct.myfilesystem.fd_owner = 612;
myparmstruct.myfilesystem.fd_group = 10;
myparmstruct.myfilesystem.fd_perms = 0755;        /* permissions (in octal)         */
myparmstruct.myfilesystem.fd_quotah = 0;
myparmstruct.myfilesystem.fd_quota = 5000;        /* Size of file system in K-bytes */

     BPX1PCT("ZFS     ",
             ZFSCALL_AGGR,             /* Aggregate operation     */
             sizeof(myparmstruct),     /* Length of Argument      */
             (char *) &myparmstruct,   /* Pointer to Argument     */
             &bpxrv,                   /* Pointer to Return_value */
             &bpxrc,                   /* Pointer to Return_code  */
             &bpxrs);                  /* Pointer to Reason_code  */

if (bpxrv < 0)
     {
     printf("Error creating file system %s in aggregate %s\n",filesystemname,aggrname);
     printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
     return bpxrc;
     }
   else /* Return from create file system was successful  */
     {
     printf("File system %s in Aggregate %s created successfully\n",filesystemname,aggrname);
     }
return 0;
}
```

Example 2 - Using FS_ID2

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR   0x40000005
#define AGOP_CREATEFILESYS_PARMDATA  131

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                   /* Eye catcher */
#define FSID_EYE "FSID"
```

Create File System

```
|    char fsid_len;                       /* Length of this structure */
|    char fsid_ver;                       /* Version */
|  #define FSID_VER_INITIAL 1             /* Initial version            */
|    char fsid_res1;                      /* Reserved.                  */
|    char fsid_res2;                      /* Reserved.                  */
|    hyper fsid_id;                       /* Internal identifier        */
|    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
|    char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
|    char fsid_reserved[32];              /* Reserved for the future  */
|    char fsid_reserved2[2];              /* Reserved for the future  */
|  } FS_ID;
|
|  typedef struct fs_id2_t {
|    char fsid_eye[4];                    /* Eye catcher */
|  #define FSID_EYE "FSID"
|    char fsid_len;                       /* Length of this structure */
|    char fsid_ver;                       /* Version */
|    char fsid_res1;                      /* Reserved.                  */
|    char fsid_res2;                      /* Reserved.                  */
|    hyper fsid_id;                       /* Internal identifier        */
|  #define FSID_VER_2       2             /* Second version  */
|    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */ /*@D10997MN*/
|    char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
|    char fsid_mtname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated     */
|    char fsid_reserved[49];              /* Reserved for the future  */
|  } FS_ID2;
|
|  typedef struct filesys_t {
|    char fd_eye[4];                     /* eye catcher               */
|  #define FD_EYE "FSDT"
|    short fd_len;                       /* Length                    */
|    char  fd_ver;                       /*                           */
|  #define FD_VER_INITIAL 1              /* Initial version           */
|    char  fd_flags;                     /* Flag bits                 */
|  #define FD_OWNER_SPECIFIED 0x80       /* Owner & group specified   */
|  #define FD_PERMS_SPECIFIED 0x40       /* Permissions specified     */
|    int   fd_owner;                     /* Owner id for root         */
|                                        /* filesystem                */
|    int   fd_group;                     /* Group id for root         */
|                                        /* filesystem                */
|    int   fd_perms;                     /* Permissions for root      */
|                                        /* filesystem                */
|  #define FD_DEFAULT_PERMS 0755         /* Default permissions if not specified  */
|    short fd_quotah;                    /* High portion of quota, in  */
|                                        /* K bytes                   */
|    char  fd_reserved1[2];              /* Reserved bytes            */
|    long  fd_quota;                     /* Low portion of quota in   */
|                                        /* K bytes                   */
|    char  fd_reserved[64];              /* More reserved bytes       */
|  } FILESYS_DATA;
|
|  struct parmstruct
|  {
|   syscall_parmlist myparms;
|   FS_ID2 fsid;
|   FILESYS_DATA myfilesystem;
|  } ;
|
|  int main(int argc, char **argv)
|  {
|  int bpxrv;
|  int bpxrc;
|  int bpxrs;
|  char filesystemname[45] = "OMVS.PRV.FS3";
|  char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";
|
|  struct parmstruct myparmstruct;
```

```
|
|  FS_ID2 *idp = &(myparmstruct.fsid);
|  FILESYS_DATA *fdp = &(myparmstruct.myfilesystem);
|
|  myparmstruct.myparms.opcode = AGOP_CREATEFILESYS_PARMDATA;
|  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
|  myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID2);
|  myparmstruct.myparms.parms[2] = 0;
|  myparmstruct.myparms.parms[3] = 0;
|  myparmstruct.myparms.parms[4] = 0;
|  myparmstruct.myparms.parms[5] = 0;
|  myparmstruct.myparms.parms[6] = 0;
|
|  memset(idp,0,sizeof(FS_ID2));                   /* Ensure reserved fields are 0 */
|  memset(fdp,0,sizeof(FILESYS_DATA));            /* Ensure reserved fields are 0 */
|
|  memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
|  myparmstruct.fsid.fsid_len = sizeof(FS_ID2);
|  myparmstruct.fsid.fsid_ver = FSID_VER_2;
|  strcpy(myparmstruct.fsid.fsid_aggrname,aggrname);
|  strcpy(myparmstruct.fsid.fsid_name,filesystemname);
|
|  memcpy(&myparmstruct.myfilesystem.fd_eye[0], FD_EYE, 4);
|  myparmstruct.myfilesystem.fd_len = sizeof(FILESYS_DATA);
|  myparmstruct.myfilesystem.fd_ver = FD_VER_INITIAL;
|  myparmstruct.myfilesystem.fd_flags = FD_OWNER_SPECIFIED | FD_PERMS_SPECIFIED;
|  myparmstruct.myfilesystem.fd_owner = 612;
|  myparmstruct.myfilesystem.fd_group = 10;
|  myparmstruct.myfilesystem.fd_perms = 0755;        /* permissions (in octal)         */
|  myparmstruct.myfilesystem.fd_quotah = 0;
|  myparmstruct.myfilesystem.fd_quota = 5000;        /* Size of file system in K-bytes */
|
|       BPX1PCT("ZFS     ",
|               ZFSCALL_AGGR,             /* Aggregate operation     */
|               sizeof(myparmstruct),     /* Length of Argument      */
|               (char *) &myparmstruct,   /* Pointer to Argument     */
|               &bpxrv,                   /* Pointer to Return_value */
|               &bpxrc,                   /* Pointer to Return_code  */
|               &bpxrs);                  /* Pointer to Reason_code  */
|
|  if (bpxrv < 0)
|       {
|       printf("Error creating file system %s in aggregate %s\n",filesystemname,aggrname);
|       printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
|       return bpxrc;
|       }
|     else /* Return from create file system was successful  */
|       {
|       printf("File system %s in Aggregate %s created successfully\n",filesystemname,aggrname);
|       }
|  return 0;
|  }
```

Define Aggregate

Purpose

The Define Aggregate subcommand call is an aggregate operation that defines (creates) a VSAM Linear Data Set (VSAM LDS). This VSAM LDS can then be formatted as a zFS aggregate.

Format

```
syscall_parmlist
   opcode                          139      AGOP_DEFINE_PARMDATA
   parms[0]                        offset to AGGR_DEFINE
   parms[1]                        size of Buffer
   parms[2]                        offset to Buffer
   parms[3]                        0
   parms[4]                        0
   parms[5]                        0
   parms[6]                        0
AGGR_DEFINE
   an_eye                         char[4]    "AGDF"
   an_len                         char       sizeof(AGGR_DEFINE)
   an_ver                         char       1
   an_aggrName                    char[45]   "OMVS.PRV.AGGR001.LDS0001"
   an_dataClass                   char[9]    0
   an_managementClass             char[9]    0
   an_storageClass                char[9]    0
   an_model                       char[45]   0
   an_modelCatalog                char[45]   0
   an_volumes[59]                 char[7]    "PRV000"
   an_reservedChars1              char       0
   an_numVolumes                  int        1
   an_spaceUnit                   int        1     /* 1 = cylinders */
   an spacePrimary                int        10    /* 10  cylinders */
   an_spaceSecondary              int        1     /*  1  cylinder  */
   an_reservedIntsl               char[32]   0

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        Aggregate is not attached
  EPERM         Permission denied to perform request

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

Related Services

Format Aggregate

Restrictions

The VSAM LDS to be defined cannot already exist

Examples

```
#pragma linkage(BPX1PCT, OS)
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 an_eye[4];                  /* Eye catcher */
#define ADEF_EYE "AGDF"
  short an_len;                    /* Length of this structure */
  char an_ver;                     /* Version */
#define ADEF_VER_INITIAL 1         /* Initial version */
  char an_aggrName[ZFS_MAX_AGGRNAME+1];
  char an_dataClass[ZFS_MAX_SMSID+1];
  char an_managementClass[ZFS_MAX_SMSID+1];
  char an_storageClass[ZFS_MAX_SMSID+1];
  char an_model[ZFS_MAX_AGGRNAME+1];
  char an_modelCatalog[ZFS_MAX_AGGRNAME+1];
  char an_catalog[ZFS_MAX_AGGRNAME+1];
  char an_volumes[59][ZFS_MAX_VOLID+1];
  char an_reservedChars1;
  int  an_numVolumes;
  int  an_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  an_spacePrimary;
  unsigned int  an_spaceSecondary;
  int  an_reservedInts1[32];
} AGGR_DEFINE;

struct parmstruct {
  syscall_parmlist myparms;
  AGGR_DEFINE aggdef;
  char Buffer[1024];
} ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";   /* aggregate name to define */
char dataclass[9] = "";
char managementclass[9] = "";
```

Define Aggregate

```
char storageclass[9] = "";
char model[45] = "";
char modelcatalog[45] = "";
char catalog[45] = "";
char volumes[7] = "PRV000";


struct parmstruct myparmstruct;

AGGR_DEFINE *agp  = &(myparmstruct.aggdef);
char        *bufp = &(myparmstruct.Buffer[0]);

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;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(agp,0,sizeof(*agp));
strcpy(agp->an_eye,ADEF_EYE);
agp->an_ver=ADEF_VER_INITIAL;
agp->an_len=sizeof(AGGR_DEFINE);

memset(bufp,0,sizeof(myparmstruct.Buffer));

strcpy(agp->an_aggrName,aggrname);

strcpy(agp->an_model,model);                          /* If included next 4 can be null */
strcpy(agp->an_dataClass,modelcatalog);
strcpy(agp->an_managementClass,managementclass);
strcpy(agp->an_storageClass,storageclass);
strcpy(agp->an_modelCatalog,modelcatalog);
strcpy(agp->an_volumes[0],(char *)volumes);
agp->an_numVolumes=1;
agp->an_spaceUnit=ZFS_SPACE_CYLS;
agp->an_spacePrimary=10;
agp->an_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;
}
```

Delete File System

Purpose

The Delete File System subcommand call is an aggregate operation that deletes an existing read-write file system from a multi-file system aggregate on a system. It can also be used to delete an existing backup file system.

| You can use an FS_ID or an FS_ID2 as input.

Format

```
syscall_parmlist
   opcode                         136      AGOP_DELETEFILESYS_PARMDATA
   parms[0]                       offset to FS_ID or FS_ID2
   parms[1]                       0
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
FS_ID or FS_ID2
   fsid_eye                       char[4]    "FSID"
   fsid_len                       char       sizeof(FS_ID)
   fsid_ver                       char       1
   fsid_res1                      char       0
   fsid_res2                      char       0
   fsid_id                        hyper
      high                          long     0
      low                           long     0
   fsid_aggrname                  char[45]   0
   fsid_name                      char[45]   "OMVS.PRV.FS3"
   fsid_reserved                  char[32]   0
   fsid_reserved2                 char[2]    0
FS_ID2 or FS_ID
   fsid_eye                       char[4]    "FSID"
   fsid_len                       short      sizeof(FS_ID2)
   fsid_ver                       char       2
   fsid_res1                      char       0
   fsid_res2                      char       0
   fsid_id
     high                         unsigned long  0
     low                          unsigned long  0
   fsid_aggrname                  char[45]   0
   fsid_name                      char[45]   "OMVS.PRV.FS3"
   fsid_mtname                    char[45]   0
   fsid_reserved                  char[49]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate containing file system is quiesced
  EXIST         File system does not exist
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  EPERM         Permission denied to perform request
  EROFS         Aggregate is attached as read only

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Delete File System

Usage

The aggregate that contains the file system to be deleted must be attached. Read-write file systems and backup file systems are related during removal as follows:

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

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

Clone File System
Create File System

Restrictions

The aggregate cannot be quiesced or attached as read-only. You cannot delete a file system that is mounted. If you are removing a read-write file system and it has a backup file system, neither the read-write nor the backup file systems can be mounted.

| When using an FS_ID2 as input, you cannot specify the file system with the USS file system name
| (fsid_mtname) since the file system cannot be mounted. You must use the zFS file system name
| (fsid_name).

Examples

Example 1 - Using FS_ID

```
#pragma linkage(BX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR   0x40000005
#define AGOP_DELETEFILESYS_PARMDATA  136

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                   /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                      /* Length of this structure */
  char fsid_ver;                      /* Version */
#define FSID_VER_INITIAL 1            /* Initial version           */
  char fsid_res1;                     /* Reserved.                 */
```

```
  char fsid_res2;                      /* Reserved.                 */
  hyper fsid_id;                       /* Internal identifier       */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];  /* Name, null terminated     */
  char fsid_reserved[32];              /* Reserved for the future   */
  char fsid_reserved2[2];              /* Reserved for the future   */
} FS_ID;

struct parmstruct
{
 syscall_parmlist myparms;
 FS_ID fsid;
 } ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char filesystemname[45] = "OMVS.PRV.FS3";

struct parmstruct myparmstruct;

FS_ID *idp = &(myparmstruct.fsid);

myparmstruct.myparms.opcode = AGOP_DELETEFILESYS_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idp,0,sizeof(FS_ID));                  /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
myparmstruct.fsid.fsid_len = sizeof(FS_ID);
myparmstruct.fsid.fsid_ver = FSID_VER_INITIAL;
strcpy(myparmstruct.fsid.fsid_name,filesystemname);

     BPX1PCT("ZFS     ",
             ZFSCALL_AGGR,            /* Aggregate operation     */
             sizeof(myparmstruct),    /* Length of Argument      */
             (char *) &myparmstruct,  /* Pointer to Argument     */
             &bpxrv,                  /* Pointer to Return_value */
             &bpxrc,                  /* Pointer to Return_code  */
             &bpxrs);                 /* Pointer to Reason_code  */

if (bpxrv < 0)
     {
     printf("Error deleting file system %s\n",filesystemname);
     printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
     return bpxrc;
     }
   else /* Return from delete file system was successful  */
     {
     printf("File system %s deleted successfully\n",filesystemname);
     }
return 0;
}
```

Example 2 - Using FS_ID2

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
```

Delete File System

```
| #include <stdio.h>
|
| #define ZFSCALL_AGGR   0x40000005
| #define AGOP_DELETEFILESYS_PARMDATA  136
|
| typedef struct syscall_parmlist_t {
|   int opcode;               /* Operation code to perform             */
|   int parms[7];             /* Specific to type of operation,        */
|                             /* provides access to the parms          */
|                             /* parms[4]-parms[6] are currently unused*/
| } syscall_parmlist;
|
| typedef struct hyper { /* unsigned 64 bit integers */
|     unsigned long high;
|     unsigned long low;
| } hyper;
|
| #define ZFS_MAX_AGGRNAME 44
| #define ZFS_MAX_FSYSNAME 44
|
| typedef struct fs_id_t {
|   char fsid_eye[4];                     /* Eye catcher */
| #define FSID_EYE "FSID"
|   char fsid_len;                        /* Length of this structure */
|   char fsid_ver;                        /* Version */
| #define FSID_VER_INITIAL 1              /* Initial version            */
|   char fsid_res1;                       /* Reserved.                  */
|   char fsid_res2;                       /* Reserved.                  */
|   hyper fsid_id;                        /* Internal identifier        */
|   char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
|   char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
|   char fsid_reserved[32];               /* Reserved for the future  */
|   char fsid_reserved2[2];               /* Reserved for the future  */
| } FS_ID;
|
| typedef struct fs_id2_t {
|   char fsid_eye[4];                     /* Eye catcher */
| #define FSID_EYE "FSID"
|   char fsid_len;                        /* Length of this structure */
|   char fsid_ver;                        /* Version */
|   char fsid_res1;                       /* Reserved.                  */
|   char fsid_res2;                       /* Reserved.                  */
|   hyper fsid_id;                        /* Internal identifier        */
| #define FSID_VER_2       2               /* Second version  */
|   char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
|   char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
|   char fsid_mtname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated     */
|   char fsid_reserved[49];               /* Reserved for the future  */
| } FS_ID2;
|
| struct parmstruct
| {
|  syscall_parmlist myparms;
|  FS_ID2 fsid;
|  } ;
|
| int main(int argc, char **argv)
| {
| int bpxrv;
| int bpxrc;
| int bpxrs;
| char filesystemname[45] = "OMVS.PRV.FS3";
|
| struct parmstruct myparmstruct;
|
| FS_ID2 *idp = &(myparmstruct.fsid);
|
```

```
|  myparmstruct.myparms.opcode = AGOP_DELETEFILESYS_PARMDATA;
|  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
|  myparmstruct.myparms.parms[1] = 0;
|  myparmstruct.myparms.parms[2] = 0;
|  myparmstruct.myparms.parms[3] = 0;
|  myparmstruct.myparms.parms[4] = 0;
|  myparmstruct.myparms.parms[5] = 0;
|  myparmstruct.myparms.parms[6] = 0;
|
|  memset(idp,0,sizeof(FS_ID2));                  /* Ensure reserved fields are 0 */
|
|  memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
|  myparmstruct.fsid.fsid_len = sizeof(FS_ID2);
|  myparmstruct.fsid.fsid_ver = FSID_VER_2;
|  strcpy(myparmstruct.fsid.fsid_name,filesystemname);
|
|       BPX1PCT("ZFS     ",
|               ZFSCALL_AGGR,             /* Aggregate operation     */
|               sizeof(myparmstruct),     /* Length of Argument      */
|               (char *) &myparmstruct,   /* Pointer to Argument     */
|               &bpxrv,                   /* Pointer to Return_value */
|               &bpxrc,                   /* Pointer to Return_code  */
|               &bpxrs);                  /* Pointer to Reason_code  */
|
|  if (bpxrv < 0)
|       {
|       printf("Error deleting file system %s\n",filesystemname);
|       printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
|       return bpxrc;
|       }
|     else /* Return from delete file system was successful  */
|       {
|       printf("File system %s deleted successfully\n",filesystemname);
|       }
|  return 0;
|  }
```

Detach Aggregate

Purpose

The Detach Aggregate subcommand call is an aggregate operation that detaches a multi-file system aggregate from a system. This makes the aggregate and all its file systems unavailable to the ZFS Physical File System running on that system.

Format

```
syscall_parmlist
   opcode                         104      AGOP_DETACH_PARMDATA
   parms[0]                       offset to AGGR_ID
   parms[1]                       0
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
AGGR_ID
   aid_eye                        char[4]    "AGID"
   aid_len                        char       sizeof(AGGR_ID)
   aid_ver                        char       1
   aid_name                       char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                   char[33]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate could not be detached due to mounted file system
  EINTR         ZFS is shutting down
  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

This function is used to detach multi-file system aggregates. Compatibility mode aggregates are detached during unmount.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the 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)
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] = "OMVS.PRV.AGGR001.LDS0001";

struct parmstruct myparmstruct;

myparmstruct.myparms.opcode = AGOP_DETACH_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID));     /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.aggr_id,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);

     BPX1PCT("ZFS     ",
             ZFSCALL_AGGR,             /* Aggregate operation     */

            sizeof(myparmstruct),     /* Length of Argument      */
             (char *) &myparmstruct,   /* Pointer to Argument     */
             &bpxrv,                   /* Pointer to Return_value */
             &bpxrc,                   /* Pointer to Return_code  */
```

```
             &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

The Format Aggregate subcommand call is an aggregate operation that formats a VSAM Linear Data Set (VSAM LDS) as a zFS aggregate.

Format

```
syscall_parmlist
   opcode                         134      AGOP_FORMAT_PARMDATA
   parms[0]                       offset to AGGR_ID
   parms[1]                       offset to AGGR_FORMAT
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
AGGR_ID
   aid_eye                        char[4]    "AGID"
   aid_len                        char       sizeof(AGGR_ID)
   aid_ver                        char       1
   aid_name                       char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                   char[33]   0
AGGR_FORMAT
   af_eye                         char[4]    "AGFM"
   af_len                         short      sizeof(AGGR_FORMAT
   af_ver                         char       1
   af_res1                        char       0
   af_size                        long       0
   af_logsize                     long       0
   af_initialempty                long       0
   af_overwrite                   int        0     /* Use caution if you specify 1 */
   af_compat                      int        1     /* compat aggr desired          */
   af_owner                       int        0     /* no uid specified             */
   af_ownerSpecified              int        0     /* use uid of issuer            */
   af_group                       int        0     /* no guid specified            */
   af_groupSpecified              int        0     /* gid set to issuer default group */
   af_perms                       int        0     /* no perms specified           */
   af_reserved                    char[64]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate is busy or otherwise unavailable
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameters
  EMVSERR       Internal error using an osi service
  ENOENT        No aggregate by this name is found
  EPERM         Permission denied to perform request

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

Format Aggregate

Related Services

Define Aggregate

Restrictions

The VSAM LDS to be formatted cannot be attached.

Examples

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR   0x40000005
#define AGOP_FORMAT_PARMDATA  134

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
  char aid_eye[4];                      /* Eye catcher */
#define AID_EYE "AGID"
  char aid_len;                         /* Length of this structure */
  char aid_ver;                         /* Version */
#define AID_VER_INITIAL 1               /* Initial version */
  char aid_name[ZFS_MAX_AGGRNAME+1];    /* Name, null terminated    */
  char aid_reserved[33];                /* Reserved for the future  */
} AGGR_ID;

typedef struct aggr_format_t
{
  char  af_eye[4];                      /* Eye catcher               */
#define AF_EYE "AGFM"
  short af_len;                         /* Length of structure       */
  char  af_ver;                         /* Version of cb */
#define AF_VER_INITIAL 1
  char  af_res1;                        /* For future use           */
  long  af_size;                        /* Amount to format of aggr */
#define AF_DEFAULT_SIZE    0          /* If set, we use default of entire primary partition of LDS */
  long  af_logsize;                     /* Size of logfile in aggr  */
#define AF_DEFAULT_LOGSIZE 0          /* If set, we use default of 1% of aggr size */
  long  af_initialempty;                /* Initial empty blocks     */
#define AF_DEFAULT_INITIALEMPTY  1     /* This is the default & minimum too */
  int   af_overwrite;                   /* Overwrite aggr if its not empty */
#define AF_OVERWRITE_OFF 0             /* Overwrite off, if aggr not empty it will NOT be formatted */
#define AF_OVERWRITE_ON  1             /* Overwrite in effect */
  int   af_compat;                      /* HFS-compat aggr desired */
#define AF_MULT    0                   /* Multi-file sys 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 gets set to pfsctl issuer uid */
#define AF_OWNER_SPECIFIED 1           /* Use owner uid set in af_owner */
  int   af_group;                       /* Group for HFS-compat */
  int   af_groupSpecified;              /* Indicates if group specified */
#define AF_GROUP_USECALLER 0           /* Group gets set to pfsctl issuer default group */
#define AF_GROUP_SPECIFIED 1           /* Use group gid set in af_group */
  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 */
  char  af_reserved[64];              /* For future use */
} AGGR_FORMAT;

struct parmstruct {
  syscall_parmlist myparms;
  AGGR_ID aid;
  AGGR_FORMAT aggformat;
} myparmstruct;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";   /* aggregate name to format */

AGGR_FORMAT *aggptr = &(myparmstruct.aggformat);
AGGR_ID *idp = &(myparmstruct.aid);

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;
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;       /* I want an HFS compatibility mode aggregate */
aggptr->af_ownerSpecified = AF_OWNER_USECALLER;
/* aggptr->af_owner = owner;  */
aggptr->af_groupSpecified=AF_GROUP_USECALLER;
/* aggptr->af_group = group;  */
aggptr->af_permsSpecified=AF_PERMS_DEFAULT;
/* aggptr->af_perms = perms;  */

     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

The Grow Aggregate subcommand call is an aggregate operation that extends the physical size of an aggregate. It can also be used to extend compatibility mode aggregates and multi-file system aggregates.

Format

```
syscall_parmlist
   opcode                         129      AGOP_GROW_PARMDATA
   parms[0]                       offset to AGGR_ID
   parms[1]                       new size of aggregate
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
AGGR_ID
   aid_eye                        char[4]    "AGID"
   aid_len                        char       sizeof(AGGR_ID)
   aid_ver                        char       1
   aid_name                       char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                   char[33]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  8             DFSMS did not extend the aggregate
  EBUSY         Aggregate containing file system is quiesced
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request
  EROFS         Aggregate is attached as read only

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The aggregate to be grown must be attached. The size specified is the new total size (in 1 K-byte blocks) being requested. The size may be rounded up by DFSMS. If a zero is specified for the new size, the aggregate is grown by a secondary allocation. The determination of whether to extend to another volume is made by DFSMS. The aggregate is quiesced by the grow operation before the grow occurs and is unquiesced after the grow completes. All mounted file system activity is quiesced during the grow operation.

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 UNIXPRIV class.

Related Services

List Aggregate Status

Restrictions

The aggregate to be grown cannot already be quiesced and cannot be attached as read-only. An aggregate cannot be made smaller.

Examples

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
/* #include <stdlib.h>  */

#define ZFSCALL_AGGR   0x40000005

define AGOP_GROW_PARMDATA  129

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
  char aid_eye[4];                    /* Eye catcher */
#define AID_EYE "AGID"
  char aid_len;                       /* Length of this structure */
  char aid_ver;                       /* Version */
#define AID_VER_INITIAL 1             /* Initial version */
  char aid_name[ZFS_MAX_AGGRNAME+1];  /* Name, null terminated    */
  char aid_reserved[33];              /* Reserved for the future  */
} AGGR_ID;

struct parmstruct
{
 syscall_parmlist myparms;
 AGGR_ID aggr_id;
} ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";

struct parmstruct myparmstruct;

memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID));    /* Ensure reserved fields are 0 */

myparmstruct.myparms.opcode = AGOP_GROW_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 70000;           /* New size of aggregate in K-bytes */
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);
```

Grow Aggregate

```
     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

Purpose

The List Aggregate Status subcommand call is an aggregate operation that returns information about a specified attached aggregate on this system.

Format

```
syscall_parmlist
   opcode                         137      AGOP_GETSTATUS_PARMDATA
   parms[0]                       offset to AGGR_ID
   parms[1]                       offset to AGGR_STATUS
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
AGGR_ID
   aid_eye                        char[4]    "AGID"
   aid_len                        char       sizeof(AGGR_ID)
   aid_ver                        char       1
   aid_name                       char[45]   "OMVS.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                      long       Aggregate ID
   as_nFileSystems                long       Number of File Systems
   as_threshold                   char       Aggrfull threshold
   as_increment                   char       Aggrfull increment
   as_flags                       char
     AS_MONITOR                     0x80
     AS_RO                          0x40
     AS_NBS                         0x20
     AS_COMPAT                      0x10
     AS_GROW                        0x08
   as_res2                        char       0
   as_blocks                      unsigned long
   as_fragSize                    long
   as_blockSize                   long
   as_totalUsable                 unsigned long
   as_realFree                    unsigned long
   as_minFree                     unsigned long
   as_reserved                    char[128]

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

This call returns information about a specified aggregate. The aggregate must be attached.

List Aggregate Status

To grow an aggregate, you would need to specify a number larger than the sum of as_totalUsable and as_minFree.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

None.

Related Services

List Attached Aggregate Names

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR   0x40000005
#define AGOP_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 unsigned long   u_long;

typedef struct aggr_status_t {
  char  as_eye[4];                      /* Eye catcher               */
#define AS_EYE "AGST"
  short as_len;                         /* Length of structure       */
  char  as_ver;
#define AS_VER_INITIAL 1                /* Initial version */
  char  as_res1;                        /* Reserved.                 */
  long  as_aggrId;                      /* Internal identifier       */
  long  as_nFileSystems;                /* Number of filesystems in aggregate */
  char  as_threshold;                   /* Threshold for aggrfull monitoring   */
  char  as_increment;                   /* Increment for aggrfull monitoring   */
  char  as_flags;                       /* Aggregate flags */
#define AS_MONITOR 0x80                 /* Aggr monitored for aggr full */
#define AS_RO      0x40                 /* Aggr attached Read-only */
#define AS_NBS     0x20                 /* Aggr should guarantee NBS */
#define AS_COMPAT  0x10                 /* Aggr is HFS compatible */
#define AS_GROW    0x08                 /* Aggr can be dynamically grown */
  char  as_res2;                        /* Reserved                  */
```

```
  u_long  as_blocks;                  /* Number of fragments in aggregate */
  long    as_fragSize;                /* Size of fragment in aggregate (normally 1K) */
  long    as_blockSize;               /* Size of blocks on aggregate (normally 8K) */
  u_long  as_totalUsable;             /* Total available blocks on aggregate (normally 8K) */
  u_long  as_realFree;                /* Total kilobytes free */
  u_long  as_minFree;                  /* Minimum kilobytes free */
  char  as_reserved[128];             /* Reserved for future */
} AGGR_STATUS;

struct parmstruct
{
 syscall_parmlist myparms;
 AGGR_ID aggr_id;
 AGGR_STATUS aggr_status;
} ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";   /* aggregate name to getstatus */

struct parmstruct myparmstruct;

AGGR_ID *idp = &(myparmstruct.aggr_id);
AGGR_STATUS *asp = &(myparmstruct.aggr_status);

myparmstruct.myparms.opcode = AGOP_GETSTATUS_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idp,0,sizeof(AGGR_ID));                  /* Ensure reserved fields are 0 */
memset(asp,0,sizeof(AGGR_STATUS));              /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.aggr_status.as_eye[0], AS_EYE, 4);
myparmstruct.aggr_status.as_len = sizeof(AGGR_STATUS);
myparmstruct.aggr_status.as_ver = AS_VER_INITIAL;

memcpy(&myparmstruct.aggr_id,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);

     BPX1PCT("ZFS     ",
             ZFSCALL_AGGR,             /* Aggregate operation     */
             sizeof(myparmstruct),     /* Length of Argument      */
             (char *) &myparmstruct,   /* Pointer to Argument     */
             &bpxrv,                   /* Pointer to Return_value */
             &bpxrc,                   /* Pointer to Return_code  */
             &bpxrs);                  /* Pointer to Reason_code  */

if (bpxrv < 0)
     {
     printf("Error getstatus aggregate %s\n", aggrname);
     printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
     return bpxrc;
     }
   else /* Return from getstatus was successful  */
     {
     printf("Aggregate %s getstatus successful\n",aggrname);
     printf("getstatus: aggr_id=%d, no_of_filesystems=%d, aggr_flags=%x\n",
```

```
                     myparmstruct.aggr_status.as_aggrId,
                                myparmstruct.aggr_status.as_nFileSystems,
                                                    myparmstruct.aggr_status.as_flags);
     printf("getstatus: threshold=%d, increment=%d\n",
                     myparmstruct.aggr_status.as_threshold,
                                  myparmstruct.aggr_status.as_increment);
     printf("getstatus: blocks=%d, frag_size=%d, block_size=%d\n",
                     myparmstruct.aggr_status.as_blocks,
                               myparmstruct.aggr_status.as_fragSize,
                                          myparmstruct.aggr_status.as_blockSize);
     printf("getstatus: total_usable=%d, real_free=%d, min_free=%d\n",
                     myparmstruct.aggr_status.as_totalUsable,
                                    myparmstruct.aggr_status.as_realFree,
                                                 myparmstruct.aggr_status.as_minFree);
     }
return 0;
}
```

List Attached Aggregate Names

Purpose

The List Attached Aggregate Names subcommand call is an aggregate operation that returns a list of the names of all attached aggregates on a system.

Format

```
syscall_parmlist
   opcode                          135      AGOP_LISTAGGRNAMES_PARMDATA
   parms[0]                        buffer length or 0
   parms[1]                        offset to AGGR_ID or 0
   parms[2]                        offset to size
   parms[3]                        0
   parms[4]                        0
   parms[5]                        0
   parms[6]                        0
AGGR_ID[n]                         Array of AGGR_IDs (n can be 0)
   aid_eye                         char[4]    "AGID"
   aid_len                         char       sizeof(AGGR_ID)
   aid_ver                         char       1
   aid_name                        char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                    char[33]   0
size needed                        long       0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  E2BIG         List is too big for buffer supplied

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

This call returns an array of AGGR_IDs - one for each attached aggregate on the system. Each AGGR_ID structure is 84 bytes. You can specify a buffer that you think might hold all of them or you can specify a buffer length and offset of zero. If you get a return code of E2BIG, the required size for the buffer is contained in the size field.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

None.

Related Services

List Aggregate Status
List File System Names

Restrictions

None.

Examples

```
#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 */
 long size;
 } ;

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;
int count_aggrs, total_aggrs;

myparmstruct.myparms.opcode = AGOP_LISTAGGRNAMES_PARMDATA;
myparmstruct.myparms.parms[0] = 0;
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

BPX1PCT("ZFS     ",
        ZFSCALL_AGGR,             /* Aggregate operation     */
        sizeof(myparmstruct),     /* Length of Argument      */
        (char *) &myparmstruct,   /* Pointer to Argument     */
        &bpxrv,                   /* Pointer to Return_value */
        &bpxrc,                   /* Pointer to Return_code  */
        &bpxrs);                  /* Pointer to Reason_code  */
```

```
if (bpxrv < 0)
{
      if (bpxrc == E2BIG)
      {
          buflen = myp->size;          /* Get buffer size needed  */
          mypsize = buflen + sizeof(syscall_parmlist) + sizeof(long);
          myp = (struct parmstruct *) malloc ((long) mypsize);
          memset(myp, 0, mypsize);

          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;
          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++)
              {
                  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 File System Names

Purpose

The List File System Names subcommand call is an aggregate operation that returns the names of the file systems contained in a specified aggregate on this system.

Format

```
syscall_parmlist
   opcode                          138      AGOP_LISTFSNAMES_PARMDATA
   parms[0]                        offset to AGGR_ID
   parms[1]                        buffer length or 0
   parms[2]                        offset to buffer or 0
   parms[3]                        offset to size
   parms[4]                        0
   parms[5]                        0
   parms[6]                        0
AGGR_ID
   aid_eye                         char[4]    "AGID"
   aid_len                         char       sizeof(AGGR_ID)
   aid_ver                         char       1
   aid_name                        char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                    char[33]   0
FS_ID[n]                           Array of FS_IDs (n can be zero)
   fsid_eye                        char[4]    "FSID"
   fsid_len                        short      sizeof(FS_ID)
   fsid_ver                        char       1
   fsid_res1                       char       0
   fsid_res2                       char       0
   fsid_id
     high                          unsigned long
     low                           unsigned long
   fsid_aggrname                   char[45]
   fsid_name                       char[45]
   fsid_reserved                   char[32]
   fsid_reserved2                  char[2]
size                               long

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  E2BIG         List is too big for buffer supplied

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The aggregate specified must be attached.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

None.

Related Services

List Attached Aggregate Names
List File System Status

Restrictions

None.

Examples

```
#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 long high;
  unsigned long 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  */ /*@D10997MN*/
  char fsid_name[ZFS_MAX_FSYSNAME+1];  /* Name, null terminated    */
  char fsid_reserved[32];              /* Reserved for the future  */
  char fsid_reserved2[2];              /* Reserved for the future  */
} FS_ID;

struct parmstruct
{
 syscall_parmlist myparms;
 AGGR_ID aggr_id;
 /* Real malloc'd structure will have an array of FS_IDs here */
 long size;
 } ;
```

List File System Names

```
int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;

struct parmstruct myparmstruct;
AGGR_ID *aggPtr;
FS_ID *fsPtr;
int fsSize = sizeof(FS_ID);
int buflen = sizeof(FS_ID);
struct parmstruct *myp = &myparmstruct;
int mypsize;
int count_fs, total_fs;
char aggrname[45]="OMVS.PRV.AGGR001.LDS0001";

memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID));     /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);

myparmstruct.myparms.opcode = AGOP_LISTFSNAMES_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

BPX1PCT("ZFS     ",
        ZFSCALL_AGGR,             /* Aggregate operation     */
        sizeof(myparmstruct),     /* Length of Argument      */
        (char *) &myparmstruct,   /* Pointer to Argument     */
        &bpxrv,                   /* Pointer to Return_value */
        &bpxrc,                   /* Pointer to Return_code  */
        &bpxrs);                  /* Pointer to Reason_code  */

if (bpxrv < 0)
{
      if (bpxrc == E2BIG)
      {
          buflen = myp->size;          /* Get buffer size needed  */
          mypsize = buflen + sizeof(syscall_parmlist) + sizeof(AGGR_ID) + sizeof(long);
          myp = (struct parmstruct *) malloc ((long) mypsize);
          memset(myp, 0, mypsize);

          memcpy(myp->aggr_id.aid_eye,AID_EYE,4);
          myp->aggr_id.aid_len = sizeof(AGGR_ID);
          myp->aggr_id.aid_ver = AID_VER_INITIAL;
          strcpy(myp->aggr_id.aid_name,aggrname);

          myp->myparms.opcode = AGOP_LISTFSNAMES_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

The List File System Names (Version 2) subcommand call is an aggregate operation that returns the names of the zFS file systems contained in a specified aggregate on this system and their corresponding USS file system names (if they are mounted).

Format

```
syscall_parmlist
   opcode                         144      AGOP_LISTFSNAMES_PARMDATA2
   parms[0]                       offset to AGGR_ID
   parms[1]                       buffer length or 0
   parms[2]                       offset to buffer or 0
   parms[3]                       offset to size
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
AGGR_ID
   aid_eye                        char[4]    "AGID"
   aid_len                        char       sizeof(AGGR_ID)
   aid_ver                        char       1
   aid_name                       char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                   char[33]   0
FS_ID2[n]                         Array of FS_ID2s (n can be zero)
   fsid_eye                       char[4]    "FSID"
   fsid_len                       short      sizeof(FS_ID2)
   fsid_ver                       char       2
   fsid_res1                      char       0
   fsid_res2                      char       0
   fsid_id
     high                         unsigned long
     low                          unsigned long
   fsid_aggrname                  char[45]
   fsid_name                      char[45]
   fsid_mtname                    char[45]
   fsid_reserved                  char[49]
size                              long

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  E2BIG         List is too big for buffer supplied

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The version 2 List File System Names returns an array of FS_ID2s.

The aggregate specified must be attached.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

None.

Related Services

List Attached Aggregate Names
List File System Status

Restrictions

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_PARMDATA2  144
#define E2BIG  145

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct aggr_id_t {
  char aid_eye[4];                    /* Eye Catcher               */
#define AID_EYE "AGID"
  char aid_len;                       /* Length of this structure  */
  char aid_ver;                       /* Version                   */
#define AID_VER_INITIAL 1             /* Initial version           */
  char aid_name[ZFS_MAX_AGGRNAME+1];  /* aggr name, null terminated */
  char aid_reserved[33];              /* Reserved for the future   */
} AGGR_ID;

typedef struct hyper {             /* This is a 64 bit integer to zFS */
  unsigned long high;
  unsigned long low;
} hyper;

typedef struct fs_id2_t {
  char fsid_eye[4];                    /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                       /* Length of this structure */
  char fsid_ver;                       /* Version */
  char fsid_res1;                      /* Reserved.                 */
  char fsid_res2;                      /* Reserved.                 */
  hyper fsid_id;                       /* Internal identifier       */
#define FSID_VER_2       2             /* Second version  */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];  /* Name, null terminated    */
  char fsid_mtname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated     */
  char fsid_reserved[49];              /* Reserved for the future  */
} FS_ID2;

struct parmstruct
{
 syscall_parmlist myparms;
 AGGR_ID aggr_id;
```

List File System Names (Version 2)

```
|   /* Real malloc'd structure will have an array of FS_ID2s here */
|   long size;
|   } ;
|
|  int main(int argc, char **argv)
|  {
|  int bpxrv;
|  int bpxrc;
|  int bpxrs;
|
|  struct parmstruct myparmstruct;
|  AGGR_ID *aggPtr;
|  FS_ID2 *fsPtr;
|  int fsSize = sizeof(FS_ID2);
|  int buflen = sizeof(FS_ID2);
|  struct parmstruct *myp = &myparmstruct;
|  int mypsize;
|  int count_fs, total_fs;
|  char aggrname[45]="OMVS.PRV.AGGR001.LDS0001";
|
|  long *p;
|
|  memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID));     /* Ensure reserved fields are 0 */
|
|  memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
|  myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
|  myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
|  strcpy(myparmstruct.aggr_id.aid_name,aggrname);
|
|  myparmstruct.myparms.opcode = AGOP_LISTFSNAMES_PARMDATA2;
|  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
|  myparmstruct.myparms.parms[1] = 0;
|  myparmstruct.myparms.parms[2] = 0;
|  myparmstruct.myparms.parms[3] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
|  myparmstruct.myparms.parms[4] = 0;
|  myparmstruct.myparms.parms[5] = 0;
|  myparmstruct.myparms.parms[6] = 0;
|
|  BPX1PCT("ZFS     ",
|          ZFSCALL_AGGR,             /* Aggregate operation     */
|          sizeof(myparmstruct),     /* Length of Argument      */
|          (char *) &myparmstruct,   /* Pointer to Argument     */
|          &bpxrv,                   /* Pointer to Return_value */
|          &bpxrc,                   /* Pointer to Return_code  */
|          &bpxrs);                  /* Pointer to Reason_code  */
|
|  if (bpxrv < 0)
|  {
|        if (bpxrc == E2BIG)
|        {
|            buflen = myp->size;          /* Get buffer size needed  */
|            mypsize = buflen + sizeof(syscall_parmlist) + sizeof(AGGR_ID) +
|                sizeof(myparmstruct.size);
|            myp = (struct parmstruct *) malloc ((long) mypsize);
|            memset(myp, 0, mypsize);
|
|            memcpy(myp->aggr_id.aid_eye,AID_EYE,4);
|            myp->aggr_id.aid_len = sizeof(AGGR_ID);
|            myp->aggr_id.aid_ver = AID_VER_INITIAL;
|            strcpy(myp->aggr_id.aid_name,aggrname);
|
|            myp->myparms.opcode = AGOP_LISTFSNAMES_PARMDATA2;
|            myp->myparms.parms[0] = sizeof(syscall_parmlist);
|            myp->myparms.parms[1] = buflen;
|            myp->myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
|            myp->myparms.parms[3] = sizeof(syscall_parmlist) + sizeof(AGGR_ID) + buflen;
|            myp->myparms.parms[4] = 0;
```

```
|            myp->myparms.parms[5] = 0;
|            myp->myparms.parms[6] = 0;
|
|            BPX1PCT("ZFS     ",
|                    ZFSCALL_AGGR,             /* Aggregate operation     */
|                    mypsize,                  /* Length of Argument      */
|                    (char *) myp,             /* Pointer to Argument     */
|                    &bpxrv,                   /* Pointer to Return_value */
|                    &bpxrc,                   /* Pointer to Return_code  */
|                    &bpxrs);                  /* Pointer to Reason_code  */
|            if (bpxrv == 0)
|            {
|                total_fs = buflen/fsSize;
|                printf("total file systems = %d in aggregate %s\n",total_fs, aggrname);
|                count_fs = 1;
|                for(fsPtr = (FS_ID2 *) &(myp->size) ; count_fs <= total_fs ; fsPtr++, count_fs++)
|                {
|                    printf("\n");
|                    printf("zFS file system name [%s]\n",fsPtr->fsid_name);
|                    printf("USS 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");
|      }
|  }
|  return 0;
|  }
```

List File System Status

Purpose

The List File System Status subcommand call is a file system operation that lists the status information of a file system.

You can use an FS_ID as input or (if you want to specify the USS file system name (that is, the mount name)) you can use an FS_ID2 as input. Of course, if you use the USS file system name, the file system must be mounted using that file system name.

Format

```
syscall_parmlist
   opcode                        142      FSOP_GETSTAT_PARMDATA
   parms[0]                      offset to FS_ID
   parms[1]                      offset ro FS_STATUS
   parms[2]                      0
   parms[3]                      0
   parms[4]                      0
   parms[5]                      0
   parms[6]                      0
FS_ID or FS_ID2
   fsid_eye                      char[4]    "FSID"
   fsid_len                      short      sizeof(FS_ID)
   fsid_ver                      char       1
   fsid_res1                     char       0
   fsid_res2                     char       0
   fsid_id
     high                        unsigned long  0
     low                         unsigned long  0
   fsid_aggrname                 char[45]   0
   fsid_name                     char[45]   "OMVS.PRV.FS3"
   fsid_reserved                 char[32]   0
   fsid_reserved2                char[2]    0
FS_ID2 or FS_ID
   fsid_eye                      char[4]    "FSID"
   fsid_len                      short      sizeof(FS_ID2)
   fsid_ver                      char       2
   fsid_res1                     char       0
   fsid_res2                     char       0
   fsid_id
     high                        unsigned long  0
     low                         unsigned long  0
   fsid_aggrname                 char[45]   0
   fsid_name                     char[45]   0
   fsid_mtname                   char[45]   "OMVS.PRV.MNT.FS3"
   fsid_reserved                 char[49]   0
FS_STATUS
  fs_eye                         char[4]    "FSST"
  fs_len                         short      sizeof(FS_STATUS)
  fs_ver                         char       1
  fs_res1                        char       0
  fs_id
    high                         unsigned long   0
    low                          unsigned long   0
  fs_cloneTime                   timeval    0
  fs_createTime                  timeval    0
  fs_updateTime                  timeval    0
  fs_accessTime                  timeval    0
  fs_allocLimit                  unsigned long   0
  fs_allocUsage                  unsigned long   0
  fs_visQuotaLimit              unsigned long   0
  fs_visQuotaUsage              unsigned long   0
  fs_accError                   unsigned long   0
```

```
  fs_accStatus                  long       0
  fs_states                     long       0
  fs_nodeMax                    long       0
  fs_minQuota                   long       0
  fs_type                       long       0
  fs_threshold                  char       0
  fs_increment                  char       0
  fs_mountstate                 char       0
    FS_NOT_MOUNTED              0
    FS_MOUNTED_RW               1
    FS_MOUNTED_RO               2
  fs_msglen                     char       0
  fs_msg                        char[128]  0
  fs_aggrname                   char[45]   0
  fs_reserved                   char[128]  0
  fs_reserved2                  char[3]    0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate containing file system is quiesced
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The aggregate containing the file system to be listed must be attached.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

None.

Related Services

List Attached Aggregate Names
List File System Aggregate Names

Restrictions

The aggregate containing the file system to be listed cannot be quiesced.

When FS_ID2 is used, if you specify the USS file system name (fsid_mtname), you cannot specify the zFS file system name (fsid_name) nor the aggregate name (fsid_aggrname).

Examples

Example 1 - Using FS_ID

```
#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
```

List File System Status

```
typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform            */
  int parms[7];             /* Specific to type of operation,       */
                            /* provides access to the parms         */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper {             /* This is a 64 bit integer to zFS */
  unsigned long high;
  unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                    /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                       /* Length of this structure */
  char fsid_ver;                       /* Version */
  char fsid_res1;                      /* Reserved.                 */
  char fsid_res2;                      /* Reserved.                 */
  hyper fsid_id;                       /* Internal identifier       */
#define FSID_VER_INITIAL 1              /* Initial version */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
  char fsid_reserved[32];               /* Reserved for the future  */
  char fsid_reserved2[2];               /* Reserved for the future  */
} FS_ID;

typedef unsigned long   u_long;

struct timeval {
        long            tv_sec;         /* seconds */
        long            tv_usec;        /* microseconds */
};

typedef struct fs_status_t {
  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_res1;                       /* Reserved.                 */
  hyper fs_id;                         /* Internal identifier       */
  struct timeval fs_cloneTime;         /* Time when this filesys made via clone or when last recloned */
  struct timeval fs_createTime;        /* Time when this filesys was created */
  struct timeval fs_updateTime;        /* Time when this filesys was last updated */
  struct timeval fs_accessTime;        /* Time when this filesys was last accessed */
  u_long fs_allocLimit;                /* Allocation limit for filesys in kilobytes*/
  u_long fs_allocUsage;                /* Amount of allocation used in kilobytes*/
  u_long fs_visQuotaLimit;             /* Visible filesystem quota in kilobytes*/
  u_long fs_visQuotaUsage;             /* How much quota is used in kilobytes*/
  u_long fs_accError;                 /* error to return for incompatible vnode ops */
  long  fs_accStatus;                  /* Operations currently being performed on file system */
  long  fs_states;                     /* State bits */
#define  FS_TYPE_RW  0x10000  /* read-write (ordinary) */
#define  FS_TYPE_BK  0x30000  /* backup (.bak) */
  long  fs_nodeMax;                    /* Maximum inode number used */
  long  fs_minQuota;
  long  fs_type;
  char  fs_threshold;                  /* Threshold for fsfull monitoring   */
  char  fs_increment;                  /* Increment for fsfull monitoring   */
  char  fs_mountstate;                 /* Aggregate flags */
#define FS_NOT_MOUNTED 0               /* Filesys not mounted */
#define FS_MOUNTED_RW  1               /* Filesys mounted RW */
#define FS_MOUNTED_RO  2               /* Filesys mounted RO */
```

```
  char  fs_msglen;                    /* Length of status message   */
  char  fs_msg[128];                  /* Status message for filesystem */
  char  fs_aggrname[ZFS_MAX_AGGRNAME+1]; /* Name of aggregate I reside on */
  char  fs_reserved[128];             /* Reserved for future use */
  char  fs_reserved2[3];              /* Reserved for future use */
} FS_STATUS;

struct parmstruct
{
 syscall_parmlist myparms;
 FS_ID fs_id;
 FS_STATUS fs_status;
} ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char filesystemname[45] = "OMVS.PRV.FS3";           /* file system name to getstatus */

struct parmstruct myparmstruct;

FS_ID *idp = &(myparmstruct.fs_id);
FS_STATUS *fsp = &(myparmstruct.fs_status);

myparmstruct.myparms.opcode = FSOP_GETSTAT_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID);
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idp,0,sizeof(FS_ID));                  /* Ensure reserved fields are 0 */
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\n",
                       myparmstruct.fs_status.fs_id.high,
                       myparmstruct.fs_status.fs_id.low,
```

List File System Status

```
                                    ctime(&myparmstruct.fs_status.fs_cloneTime.tv_sec),
                                          ctime(&myparmstruct.fs_status.fs_createTime.tv_sec));
     printf("getstatus: update_time=%s, access_time=%s\n",
                        ctime(&myparmstruct.fs_status.fs_updateTime.tv_sec),
                                    ctime(&myparmstruct.fs_status.fs_accessTime.tv_sec));
     printf("getstatus: alloc_limit=%u, alloc_usage=%u, quota_limit=%u\n",
                        myparmstruct.fs_status.fs_allocLimit,
                                    myparmstruct.fs_status.fs_allocUsage,
                                       myparmstruct.fs_status.fs_visQuotaLimit);
     printf("getstatus: quota_usage=%u, accError=%u, accStatus=%x, states=%x\n",
                        myparmstruct.fs_status.fs_visQuotaUsage,
                                    myparmstruct.fs_status.fs_accError,
                                          myparmstruct.fs_status.fs_accStatus,
                                                myparmstruct.fs_status.fs_states);
     printf("getstatus: max_inode=%d, min_quota=%d, type=%d, 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: 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);
     }
return 0;
}
```

Example 2 - Using FS_ID2

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <time.h>       /* ctime  */

#define ZFSCALL_FILESYS   0x40000004
#define FSOP_GETSTAT_PARMDATA  142

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform            */
  int parms[7];             /* Specific to type of operation,       */
                            /* provides access to the parms         */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper {             /* This is a 64 bit integer to zFS */
  unsigned long high;
  unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id2_t {
  char fsid_eye[4];                    /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                       /* Length of this structure */
  char fsid_ver;                       /* Version */
  char fsid_res1;                      /* Reserved.                 */
  char fsid_res2;                      /* Reserved.                 */
  hyper fsid_id;                       /* Internal identifier       */
#define FSID_VER_2       2           /* version for FS_ID2 */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
  char fsid_mtname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated    */
```

```
|    char fsid_reserved[49];              /* Reserved for the future  */
|  } FS_ID2;
|  typedef unsigned long   u_long;
|
|  struct timeval {
|          long            tv_sec;         /* seconds */
|          long            tv_usec;        /* microseconds */
|  };
|
|  typedef struct fs_status_t {
|    char  fs_eye[4];                     /* Eye catcher               */
|  #define FS_EYE "FSST"
|    short fs_len;                        /* Length of structure       */
|    char  fs_ver;
|  #define FS_VER_INITIAL 1              /* Initial version */
|    char  fs_res1;                       /* Reserved.                 */
|    hyper fs_id;                         /* Internal identifier       */
|    struct timeval fs_cloneTime;         /* Time when this filesys made via clone or when last recloned */
|    struct timeval fs_createTime;        /* Time when this filesys was created */
|    struct timeval fs_updateTime;        /* Time when this filesys was last updated */
|    struct timeval fs_accessTime;        /* Time when this filesys was last accessed */
|    u_long fs_allocLimit;                /* Allocation limit for filesys in kilobytes*/
|    u_long fs_allocUsage;                /* Amount of allocation used in kilobytes*/
|    u_long fs_visQuotaLimit;             /* Visible filesystem quota in kilobytes*/
|    u_long fs_visQuotaUsage;             /* How much quota is used in kilobytes*/
|    u_long fs_accError;               /* error to return for incompatible vnode ops */
|    long  fs_accStatus;                  /* Operations currently being performed on file system */
|    long  fs_states;                     /* State bits */
|  #define  FS_TYPE_RW  0x10000  /* read-write (ordinary) */
|  #define  FS_TYPE_BK  0x30000  /* backup (.bak) */
|    long  fs_nodeMax;                    /* Maximum inode number used */
|    long  fs_minQuota;
|    long  fs_type;
|    char  fs_threshold;                  /* Threshold for fsfull monitoring   */
|    char  fs_increment;                  /* Increment for fsfull monitoring   */
|    char  fs_mountstate;                 /* Aggregate flags */
|  #define FS_NOT_MOUNTED 0              /* Filesys not mounted */
|  #define FS_MOUNTED_RW  1              /* Filesys mounted RW */
|  #define FS_MOUNTED_RO  2              /* Filesys mounted RO */
|    char  fs_msglen;                     /* Length of status message   */
|    char  fs_msg[128];                   /* Status message for filesystem */
|    char  fs_aggrname[ZFS_MAX_AGGRNAME+1]; /* Name of aggregate I reside on */
|    char  fs_reserved[128];              /* Reserved for future use */
|    char  fs_reserved2[3];               /* Reserved for future use */
|  } FS_STATUS;
|
|  struct parmstruct
|  {
|   syscall_parmlist myparms;
|   FS_ID2 fs_id2;
|   FS_STATUS fs_status;
|  } ;
|
|  int main(int argc, char **argv)
|  {
|  int bpxrv;
|  int bpxrc;
|  int bpxrs;
|  char filesystemname[45] = "OMVS.PRV.MNT.FS3";           /* file system name to getstatus */
|
|  struct parmstruct myparmstruct;
|
|  FS_ID2 *idp = &(myparmstruct.fs_id2);
|  FS_STATUS *fsp = &(myparmstruct.fs_status);
|
|  myparmstruct.myparms.opcode = FSOP_GETSTAT_PARMDATA;
|  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
```

List File System Status

```
| 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("USS file system %s getstatus successful\n",filesystemname);
|      printf("getstatus: fs_id=%d,,%d, clone_time=%s, create_time=%s\n",
|                       myparmstruct.fs_status.fs_id.high,
|                       myparmstruct.fs_status.fs_id.low,
|                                    ctime(&myparmstruct.fs_status.fs_cloneTime.tv_sec),
|                                          ctime(&myparmstruct.fs_status.fs_createTime.tv_sec));
|      printf("getstatus: update_time=%s, access_time=%s\n",
|                       ctime(&myparmstruct.fs_status.fs_updateTime.tv_sec),
|                                     ctime(&myparmstruct.fs_status.fs_accessTime.tv_sec));
|      printf("getstatus: alloc_limit=%u, alloc_usage=%u, quota_limit=%u\n",
|                       myparmstruct.fs_status.fs_allocLimit,
|                                  myparmstruct.fs_status.fs_allocUsage,
|                                           myparmstruct.fs_status.fs_visQuotaLimit);
|      printf("getstatus: quota_usage=%u, accError=%u, accStatus=%x, states=%x\n",
|                       myparmstruct.fs_status.fs_visQuotaUsage,
|                                     myparmstruct.fs_status.fs_accError,
|                                          myparmstruct.fs_status.fs_accStatus,
|                                                    myparmstruct.fs_status.fs_states);
|      printf("getstatus: max_inode=%d, min_quota=%d, type=%d, 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: 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);
|      }
| return 0;
| }
```

Query Config Option

Purpose

The Query Config Option is a set of subcommand calls (that are configuration operations) that retrieve the current value for a particular configuration setting. Each one returns the particular configuration setting as a character string.

The following Format and Example use the CFGOP_QUERY_ADM_THREADS subcommand. The other query subcommands (refer to “Configuration operations” on page 115) would operate in a similar manner. That is, each of them would return the configuration setting as a character string in the co_string field.

Format

```
syscall_parmlist
   opcode                          180      CFGOP_QUERY_ADM_THREADS
   parms[0]                        offset to CFG_OPTION
   parms[1]                        0
   parms[2]                        0
   parms[3]                        0
   parms[4]                        0
   parms[5]                        0
   parms[6]                        0
CFG_OPTION
   co_eye                          char[4]    "CFOP"
   co_len                          short      sizeof(CFG_OPTION)
   co_ver                          char       1
   co_string                       char[81]   0
   co_value_reserved               int        0
   co_reserved                     char[24]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate could not be quiesced
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

Query Config Option subcommands are used to retrieve the current value of a particular configuration option. Each subcommand retrieves one configuration as a character string.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

None.

Related Services

Set Config Option

Restrictions

None.

Examples

```
#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_reserved[4];   /* Reserved for future use */
  char  co_reserved[24];        /* Reserved for future use */
} CFG_OPTION;

struct parmstruct {
  syscall_parmlist myparms;
  CFG_OPTION co;
} myparmstruct;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;

CFG_OPTION *coptr = &(myparmstruct.co);

myparmstruct.myparms.opcode = CFGOP_QUERY_ADM_THREADS;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(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;
|  }
```

Quiesce Aggregate

Purpose

The Quiesce Aggregate subcommand call is an aggregate operation that quiesces a multi-file system aggregate on a system. This quiesces activity on the aggregate and all its file systems.

Format

```
syscall_parmlist
   opcode                          132      AGOP_QUIESCE_PARMDATA
   parms[0]                        offset to AGGR_ID
   parms[1]                        offset to handle returned by quiesce
   parms[2]                        0
   parms[3]                        0
   parms[4]                        0
   parms[5]                        0
   parms[6]                        0
AGGR_ID
   aid_eye                         char[4]    "AGID"
   aid_len                         char       sizeof(AGGR_ID)
   aid_ver                         char       1
   aid_name                        char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                    char[33]   0
quiesce_handle                     long

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate could not be quiesced
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

Quiesce Aggregate is used to suspend activity on an aggregate. All activity on file systems contained in the aggregate that are mounted is also suspended. This would normally be used prior to backing up an aggregate. The aggregate must be attached in order to be quiesced. The quiesce operation returns a quiesce handle that must be supplied on the unquiesce call.

Reserved fields and undefined flags must be set to binary zeros.

The Grow Aggregate subcommand does its own quiesce so you should not quiesce an aggregate before growing it.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

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_QUIESCE_PARMDATA  132

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
  char aid_eye[4];                    /* Eye catcher */
#define AID_EYE "AGID"
  char aid_len;                       /* Length of this structure */
  char aid_ver;                       /* Version */
#define AID_VER_INITIAL 1             /* Initial version */
  char aid_name[ZFS_MAX_AGGRNAME+1];  /* Name, null terminated    */
  char aid_reserved[33];              /* Reserved for the future  */
} AGGR_ID;

struct parmstruct
{
 syscall_parmlist myparms;
 AGGR_ID aggr_id;
 long quiesce_handle;
} ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";
long save_quiesce_handle;

struct parmstruct myparmstruct;

AGGR_ID *idp = &(myparmstruct.aggr_id);

myparmstruct.myparms.opcode = AGOP_QUIESCE_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID));   /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.aggr_id,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);
```

Quiesce Aggregate

```
     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;
}
```

Rename File System

Purpose

The Rename File System subcommand call is a file system operation that renames a file system.

| You can use an FS_ID or an FS_ID2 as input for the old zFS file system name or the new zFS file system
| name (that is, the fsid_name).

Format

```
syscall_parmlist
   opcode                      140      FSOP_RENAME_PARMDATA
   parms[0]                    offset to FS_ID (Old Name)
   parms[1]                    offset to FS_ID (New Nakme)
   parms[2]                    0
   parms[3]                    0
   parms[4]                    0
   parms[5]                    0
   parms[6]                    0
FS_ID or FS_ID2                /* Old File System Name */
   fsid_eye                    char[4]    "FSID"
   fsid_len                    short      sizeof(FS_ID)
   fsid_ver                    char       1
   fsid_res1                   char       0
   fsid_res2                   char       0
   fsid_id
     high                      unsigned long  0
     low                       unsigned long  0
   fsid_aggrname               char[45]   0
   fsid_name                   char[45]   "OMVS.PRV.FS3"
   fsid_reserved               char[32]   0
   fsid_reserved2              char[2]    0
FS_ID2 or FS_ID                /* Old File System Name */
   fsid_eye                    char[4]    "FSID"
   fsid_len                    short      sizeof(FS_ID2)
   fsid_ver                    char       2
   fsid_res1                   char       0
   fsid_res2                   char       0
   fsid_id
     high                      unsigned long  0
     low                       unsigned long  0
   fsid_aggrname               char[45]   0
   fsid_name                   char[45]   0
   fsid_mtname                 char[45]   "OMVS.PRV.FS3"
   fsid_reserved               char[49]   0
FS_ID or FS_ID2                         /* New File System Name */
   fsid_eye                    char[4]    "FSID"
   fsid_len                    short      sizeof(FS_ID)
   fsid_ver                    char       1
   fsid_res1                   char       0
   fsid_res2                   char       0
   fsid_id
     high                      unsigned long  0
     low                       unsigned long  0
   fsid_aggrname               char[45]   0
   fsid_name                   char[45]   "OMVS.PRV.FS4"
   fsid_reserved               char[32]   0
   fsid_reserved2              char[2]    0
FS_ID2 or FS_ID                /* New File System Name */
   fsid_eye                    char[4]    "FSID"
   fsid_len                    short      sizeof(FS_ID2)
   fsid_ver                    char       2
   fsid_res1                   char       0
   fsid_res2                   char       0
```

Rename File System

```
   fsid_id
     high                          unsigned long  0
     low                           unsigned long  0
   fsid_aggrname                   char[45]   0
   fsid_name                       char[45]   0
   fsid_mtname                     char[45]   "OMVS.PRV.FS4"
   fsid_reserved                   char[49]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate containing file system is quiesced
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request
  EROFS         Aggregate is attached as read only

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The aggregate that contains the file system to be renamed must be attached.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

Clone File System

Restrictions

A backup file system cannot be renamed by itself. You must rename the read-write file system (which renames both the read-write and the backup file systems). You cannot rename a read-write file system to an name that ends in **.bak**. This is reserved for backup file systems. If a backup file system exists, you cannot rename a read-write file system to a name that is longer than 40 characters. The file system(s) to be renamed cannot be mounted. The aggregate containing the file system to be renamed cannot be quiesced or attached as read-only. If you specify an aggregate name in the old file system name FS_ID and in the new file system name FS_ID, they must be the same.

| When using an FS_ID2 for the old file system name, you cannot specify the USS file system name
| (fsid_mtname) because the file system to be renamed cannot be mounted. When using the FS_ID2 for the
| new file system name, you cannot specify the USS file system name (fsid_mtname) because the API
| needs the new zFS file system name.

Examples

Example 1 - Using FS_ID

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS   0x40000004
```

```
#define FSOP_RENAME_PARMDATA  140

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform            */
  int parms[7];             /* Specific to type of operation,       */
                            /* provides access to the parms         */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                    /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                       /* Length of this structure */
  char fsid_ver;                       /* Version */
#define FSID_VER_INITIAL 1             /* Initial version           */
  char fsid_res1;                      /* Reserved.                 */
  char fsid_res2;                      /* Reserved.                 */
  hyper fsid_id;                       /* Internal identifier       */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
  char fsid_reserved[32];              /* Reserved for the future  */
  char fsid_reserved2[2];              /* Reserved for the future  */
} FS_ID;

struct parmstruct
{
 syscall_parmlist myparms;
 FS_ID fsid_old;
 FS_ID fsid_new;
 } ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char old_filesystemname[45] = "OMVS.PRV.FS3";
char new_filesystemname[45] = "OMVS.PRV.FS4";

struct parmstruct myparmstruct;

FS_ID *idop = &(myparmstruct.fsid_old);
FS_ID *idnp = &(myparmstruct.fsid_new);

myparmstruct.myparms.opcode = FSOP_RENAME_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID);
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idop,0,sizeof(FS_ID));                  /* Ensure reserved fields are 0 */
memset(idnp,0,sizeof(FS_ID));                  /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.fsid_old.fsid_eye, FSID_EYE, 4);
myparmstruct.fsid_old.fsid_len = sizeof(FS_ID);
myparmstruct.fsid_old.fsid_ver = FSID_VER_INITIAL;
```

Rename File System

```
strcpy(myparmstruct.fsid_old.fsid_name,old_filesystemname);

memcpy(&myparmstruct.fsid_new.fsid_eye, FSID_EYE, 4);
myparmstruct.fsid_new.fsid_len = sizeof(FS_ID);
myparmstruct.fsid_new.fsid_ver = FSID_VER_INITIAL;
strcpy(myparmstruct.fsid_new.fsid_name,new_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 renaming file system %s to %s\n",old_filesystemname, new_filesystemname);
     printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
     return bpxrc;
     }
   else /* Return from rename file system was successful  */
     {
     printf("File system %s was renamed to %s successfully\n",old_filesystemname, new_filesystemname);
     }
return 0;
}
```

Example 2 - Using FS_ID2

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS   0x40000004
#define FSOP_RENAME_PARMDATA  140

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                    /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                       /* Length of this structure */
  char fsid_ver;                       /* Version */
#define FSID_VER_INITIAL 1             /* Initial version            */
  char fsid_res1;                      /* Reserved.                  */
  char fsid_res2;                      /* Reserved.                  */
  hyper fsid_id;                       /* Internal identifier        */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
  char fsid_reserved[32];              /* Reserved for the future  */
  char fsid_reserved2[2];              /* Reserved for the future  */
} FS_ID;
```

```
|
|
|
|  typedef struct fs_id2_t {
|    char fsid_eye[4];                    /* Eye catcher */
|  #define FSID_EYE "FSID"
|    char fsid_len;                       /* Length of this structure */
|    char fsid_ver;                       /* Version */
|    char fsid_res1;                      /* Reserved.                */
|    char fsid_res2;                      /* Reserved.                */
|    hyper fsid_id;                       /* Internal identifier      */
|  #define FSID_VER_2       2            /* version for R14 */
|    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
|    char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
|    char fsid_mtname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated    */
|    char fsid_reserved[49];              /* Reserved for the future  */
|  } FS_ID2;
|
|  struct parmstruct
|  {
|   syscall_parmlist myparms;
|   FS_ID2 fsid_old;
|   FS_ID2 fsid_new;
|   } ;
|
|  int main(int argc, char **argv)
|  {
|  int bpxrv;
|  int bpxrc;
|  int bpxrs;
|  char old_filesystemname[45] = "OMVS.PRV.FS3";
|  char new_filesystemname[45] = "OMVS.PRV.FS4";
|
|  struct parmstruct myparmstruct;
|
|  FS_ID2 *idop = &(myparmstruct.fsid_old);
|  FS_ID2 *idnp = &(myparmstruct.fsid_new);
|
|  myparmstruct.myparms.opcode = FSOP_RENAME_PARMDATA;
|  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
|  myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID2);
|  myparmstruct.myparms.parms[2] = 0;
|  myparmstruct.myparms.parms[3] = 0;
|  myparmstruct.myparms.parms[4] = 0;
|  myparmstruct.myparms.parms[5] = 0;
|  myparmstruct.myparms.parms[6] = 0;
|
|  memset(idop,0,sizeof(FS_ID2));                  /* Ensure reserved fields are 0 */
|  memset(idnp,0,sizeof(FS_ID2));                  /* Ensure reserved fields are 0 */
|
|  memcpy(&myparmstruct.fsid_old.fsid_eye, FSID_EYE, 4);
|  myparmstruct.fsid_old.fsid_len = sizeof(FS_ID2);
|  myparmstruct.fsid_old.fsid_ver = FSID_VER_2;
|  strcpy(myparmstruct.fsid_old.fsid_name,old_filesystemname);
|
|  memcpy(&myparmstruct.fsid_new.fsid_eye, FSID_EYE, 4);
|  myparmstruct.fsid_new.fsid_len = sizeof(FS_ID2);
|  myparmstruct.fsid_new.fsid_ver = FSID_VER_2;
|  strcpy(myparmstruct.fsid_new.fsid_name,new_filesystemname);
|
|       BPX1PCT("ZFS     ",
|               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  */
```

Rename File System

```
|
|  if (bpxrv < 0)
|       {
|       printf("Error renaming file system %s to %s\n",old_filesystemname, new_filesystemname);
|       printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
|       return bpxrc;
|       }
|     else /* Return from rename file system was successful  */
|       {
|       printf("File system %s was renamed to %s successfully\n",old_filesystemname, new_filesystemname);
|       }
|  return 0;
|
|  }
```

Set Config Option

Purpose

The Set Config Option is a set of subcommand calls (that are configuration operations) that set the current value for a particular configuration setting. Each one sets the particular configuration setting from input specified as a character string.

The following Format and Example use the CFGOP_ADM_THREADS subcommand. The other set subcommands (refer to “Configuration operations” on page 115) would operate in a similar manner. That is, each of them would set the configuration setting from the character string in the co_string field.

Format

```
syscall_parmlist
   opcode                          150      CFGOP_ADM_THREADS
   parms[0]                        offset to CFG_OPTION
   parms[1]                        0
   parms[2]                        0
   parms[3]                        0
   parms[4]                        0
   parms[5]                        0
   parms[6]                        0
CFG_OPTION
   co_eye                          char[4]    "CFOP"
   co_len                          short      sizeof(AGGR_ID)
   co_ver                          char       1
   co_string                       char[81]   "15"   /* New value for adm_threads  */
   co_value_reserved               int        0
   co_reserved                     char[24]   0
quiesce_handle                     long

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate could not be quiesced
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

Set Config Option subcommands are used to set the current value of a particular configuration option. Each subcommand sets one configuration from a character string.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

Query Config Option

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_CONFIG  0x40000006
#define CFGOP_ADM_THREADS          150    /* Set number of admin threads */


typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct config_option_t {
  char co_eye[4];               /* Eye catcher */
#define CFGO_EYE "CFOP"
  short co_len;                 /* Length of structure */
  char  co_ver;                 /* Version of structure */
#define CO_VER_INITIAL 1        /* Initial version */
#define CO_SLEN 80              /* Sizeof string */
  char  co_string[CO_SLEN+1];   /* String value for option must be 0 terminated */
  int   co_value_reserved[4];   /* Reserved for future use */
  char  co_reserved[24];        /* Reserved for future use */
} CFG_OPTION;

struct parmstruct {
  syscall_parmlist myparms;
  CFG_OPTION co;
} myparmstruct;

char new_adm_threads[CO_SLEN+1]="15";           /* New adm_threads value   */

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;

CFG_OPTION *coptr = &(myparmstruct.co);

myparmstruct.myparms.opcode = CFGOP_ADM_THREADS;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(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;
|  }
```

Set File System Quota

Purpose

The Set File System Quota subcommand call is a file system operation that sets the quota for the file system.

You can use an FS_ID or an FS_ID2 as input.

Format

```
syscall_parmlist
   opcode                         141      FSOP_SETQUOTA_PARMDATA
   parms[0]                       offset to FS_ID or FS_ID2
   parms[1]                       7000     quota
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
FS_ID or FS_ID2
   fsid_eye                       char[4]    "FSID"
   fsid_len                       short      sizeof(FS_ID)
   fsid_ver                       char       1
   fsid_res1                      char       0
   fsid_res2                      char       0
   fsid_id
     high                         unsigned long  0
     low                          unsigned long  0
   fsid_aggrname                  char[45]   0
   fsid_name                      char[45]   "OMVS.PRV.FS3"
   fsid_reserved                  char[32]   0
   fsid_reserved2                 char[2]    0
FS_ID2 or FS_ID
   fsid_eye                       char[4]    "FSID"
   fsid_len                       short      sizeof(FS_ID2)
   fsid_ver                       char       2
   fsid_res1                      char       0
   fsid_res2                      char       0
   fsid_id
     high                         unsigned long  0
     low                          unsigned long  0
   fsid_aggrname                  char[45]   0
   fsid_name                      char[45]   "OMVS.PRV.FS3"
   fsid_mtname                    char[45]   0
   fsid_reserved                  char[49]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EBUSY         Aggregate containing file system is quiesced
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request
  EROFS         Aggregate is attached as read only

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The aggregate containing the file system to have its quota set must be attached. A quota can be decreased as long as the quota usage has not exceeded the new quota.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

List File System Status

Restrictions

You cannot set the quota of a backup file system. The aggregate containing the file system to have its quota set cannot be quiesced or attached as read-only.

Examples

Example 1 - Using FS_ID

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS   0x40000004
#define FSOP_SETQUOTA_PARMDATA  141

typedef struct syscall_parmlist_t {
  int opcode;               /* Operation code to perform             */
  int parms[7];             /* Specific to type of operation,        */
                            /* provides access to the parms          */
                            /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t {
  char fsid_eye[4];                     /* Eye catcher */
#define FSID_EYE "FSID"
  char fsid_len;                        /* Length of this structure */
  char fsid_ver;                        /* Version */
#define FSID_VER_INITIAL 1              /* Initial version           */
  char fsid_res1;                       /* Reserved.                 */
  char fsid_res2;                       /* Reserved.                 */
  hyper fsid_id;                        /* Internal identifier       */
  char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
  char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
  char fsid_reserved[32];               /* Reserved for the future  */
  char fsid_reserved2[2];               /* Reserved for the future  */
} FS_ID;

struct parmstruct
{
 syscall_parmlist myparms;
```

Set File System Quota

```
 FS_ID fsid;
 } ;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
char filesystemname[45] = "OMVS.PRV.FS3";
int quota;

struct parmstruct myparmstruct;

FS_ID *idp = &(myparmstruct.fsid);
quota = 7000;

myparmstruct.myparms.opcode = FSOP_SETQUOTA_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = quota;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idp,0,sizeof(FS_ID));                  /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
myparmstruct.fsid.fsid_len = sizeof(FS_ID);
myparmstruct.fsid.fsid_ver = FSID_VER_INITIAL;
strcpy(myparmstruct.fsid.fsid_name,filesystemname);

     BPX1PCT("ZFS     ",
             ZFSCALL_FILESYS,          /* File system operation   */
             sizeof(myparmstruct),     /* Length of Argument      */
             (char *) &myparmstruct,   /* Pointer to Argument     */
             &bpxrv,                   /* Pointer to Return_value */
             &bpxrc,                   /* Pointer to Return_code  */
             &bpxrs);                  /* Pointer to Reason_code  */

if (bpxrv < 0)
     {
     printf("Error setting quota of %d for file system %s\n",quota, filesystemname);
     printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
     return bpxrc;
     }
   else /* Return from set quota of file system was successful  */
     {
     printf("File system %s had its quota set to %d successfully\n",filesystemname, quota);
     }
return 0;
}
```

Example 2 - Using FS_ID2

```
| #pragma linkage(BPX1PCT, OS)
| extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
|
| #include <stdio.h>
|
| #define ZFSCALL_FILESYS   0x40000004
| #define FSOP_SETQUOTA_PARMDATA  141
|
| typedef struct syscall_parmlist_t {
|   int opcode;               /* Operation code to perform            */
|   int parms[7];             /* Specific to type of operation,       */
|                             /* provides access to the parms         */
```

```
                            /* parms[4]-parms[6] are currently unused*/
  } syscall_parmlist;

  typedef struct hyper { /* unsigned 64 bit integers */
      unsigned long high;
      unsigned long low;
  } hyper;

  #define ZFS_MAX_AGGRNAME 44
  #define ZFS_MAX_FSYSNAME 44

  typedef struct fs_id_t {
    char fsid_eye[4];                    /* Eye catcher */
  #define FSID_EYE "FSID"
    char fsid_len;                       /* Length of this structure */
    char fsid_ver;                       /* Version */
  #define FSID_VER_INITIAL 1             /* Initial version            */
    char fsid_res1;                      /* Reserved.                  */
    char fsid_res2;                      /* Reserved.                  */
    hyper fsid_id;                       /* Internal identifier        */
    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
    char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
    char fsid_reserved[32];              /* Reserved for the future  */
    char fsid_reserved2[2];              /* Reserved for the future  */
  } FS_ID;

  typedef struct fs_id2_t {
    char fsid_eye[4];                    /* Eye catcher */
  #define FSID_EYE "FSID"
    char fsid_len;                       /* Length of this structure */
    char fsid_ver;                       /* Version */
    char fsid_res1;                      /* Reserved.                  */
    char fsid_res2;                      /* Reserved.                  */
    hyper fsid_id;                       /* Internal identifier        */
  #define FSID_VER_2       2              /* Second version  */
    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string  */
    char fsid_name[ZFS_MAX_FSYSNAME+1];   /* Name, null terminated    */
    char fsid_mtname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated     */
    char fsid_reserved[49];              /* Reserved for the future  */
  } FS_ID2;

  struct parmstruct
  {
   syscall_parmlist myparms;
   FS_ID2 fsid;
   } ;

  int main(int argc, char **argv)
  {
  int bpxrv;
  int bpxrc;
  int bpxrs;
  char filesystemname[45] = "OMVS.PRV.FS3";
  int quota;

  struct parmstruct myparmstruct;

  FS_ID2 *idp = &(myparmstruct.fsid);
  quota = 7000;

  myparmstruct.myparms.opcode = FSOP_SETQUOTA_PARMDATA;
  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
  myparmstruct.myparms.parms[1] = quota;
  myparmstruct.myparms.parms[2] = 0;
  myparmstruct.myparms.parms[3] = 0;
  myparmstruct.myparms.parms[4] = 0;
  myparmstruct.myparms.parms[5] = 0;
```

Set File System Quota

```
| myparmstruct.myparms.parms[6] = 0;
|
| memset(idp,0,sizeof(FS_ID2));                  /* Ensure reserved fields are 0 */
|
| memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
| myparmstruct.fsid.fsid_len = sizeof(FS_ID2);
| myparmstruct.fsid.fsid_ver = FSID_VER_2;
| strcpy(myparmstruct.fsid.fsid_name,filesystemname);
|
|      BPX1PCT("ZFS     ",
|              ZFSCALL_FILESYS,          /* File system operation   */
|              sizeof(myparmstruct),     /* Length of Argument      */
|              (char *) &myparmstruct,   /* Pointer to Argument     */
|              &bpxrv,                   /* Pointer to Return_value */
|              &bpxrc,                   /* Pointer to Return_code  */
|              &bpxrs);                  /* Pointer to Reason_code  */
|
| if (bpxrv < 0)
|      {
|      printf("Error setting quota of %d for file system %s\n",quota, filesystemname);
|      printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
|      return bpxrc;
|      }
|    else /* Return from set quota of file system was successful  */
|      {
|      printf("File system %s had its quota set to %d successfully\n",filesystemname, quota);
|      }
| return 0;
| |}
```

Unquiesce Aggregate

Purpose

The Unquiesce Aggregate subcommand call is an aggregate operation that unquiesces a multi-file system aggregate on a system. This allows activity on the aggregate and all its file systems to resume.

Format

```
syscall_parmlist
   opcode                         133      AGOP_UNQUIESCE_PARMDATA
   parms[0]                       offset to AGGR_ID
   parms[1]                       quiesce handle
   parms[2]                       0
   parms[3]                       0
   parms[4]                       0
   parms[5]                       0
   parms[6]                       0
AGGR_ID
   aid_eye                        char[4]    "AGID"
   aid_len                        char       sizeof(AGGR_ID)
   aid_ver                        char       1
   aid_name                       char[45]   "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved                   char[33]   0

Return_value    0 if request is successful, -1 if it is not successful

Return_code

  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request

Reason_code

  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage

The unquiesce call must supply the quiesce handle that was returned by the quiesce call. The aggregate would normally be quiesced prior to backing up the aggregate. After the backup is complete, the aggregate can be unquiesced.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the UNIXPRIV class.

Related Services

Quiesce Aggregate

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
```

Unquiesce Aggregate

```
#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] = "OMVS.PRV.AGGR001.LDS0001";
long save_quiesce_handle;

struct parmstruct myparmstruct;

if (argc != 2)
   {
    printf("This unquiesce program requires a quiesce handle from the quiesce program as a parameter\n");
    return 1;
    }
save_quiesce_handle = atoi(argv[1]);

myparmstruct.myparms.opcode = AGOP_UNQUIESCE_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = save_quiesce_handle;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID));    /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);

     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;
}
```

Unquiesce Aggregate

Appendix A. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS: TSO/E Primer*, SA22-7787, *z/OS: TSO/E User's Guide*, SA22-7794, and *z/OS: ISPF User's Guide Volume*, SC34-4822, for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Appendix B. Notices

This information was developed for products and services offered in the USA. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road

Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM application programming interfaces.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This *z/OS Distributed File Service zSeries File System Administration* primarily documents information that is NOT intended to be used as Programming Interfaces of the Distributed File Service.

This *z/OS Distributed File Service zSeries File System Administration* also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of the Distributed File Service. This information is identified where it occurs by an introductory statement to a chapter.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

Table 1. Trademarks

BookManager	DFSMSdss	IBM
IBMLink	Library Reader	Lotus Notes
MVS	OS/390	Resource Link
RMF	z/OS	z/OS.e
z/Series		

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Bibliography

This bibliography lists and provides a brief description of each publication in the z/OS Distributed File Service library.

- *z/OS: Distributed File Service DFS Administration*, SC24-5915

This document introduces the Distributed File Service concepts to system and network administrators and provides an in-depth understanding of the Distributed File Service, it uses and benefits. This document also provides reference information for the commands and files used by system and network administrators to work with the Distributed File Service.

- *z/OS: Distributed File Service Customization*, SC24-5916

This document helps system and network administrators configure the Distributed File Service.

- *z/OS: Distributed File Service Messages and Codes*, SC24-5917

This document provides detailed explanations and recovery actions for the messages, status codes, and exception codes issued by the Distributed File Service.

- *z/OS: Distributed File Service SMB Administration*, SC24-5918

This document provides guidance and reference information for system and network administrators to use when they work with the Server Message Block (SMB) support of the Distributed File Service base element of z/OS. SMB is a protocol for remote file/print access used by Windows.

Index

Special characters

A

B

C

D

N

O

P

Q

R

S

T

U

V

Z

Program Number: 5694-A01, 5655-G52

Printed in U.S.A.

SC24-5989-02

Spine information:

z/OS

z/OS V1R4.0 Distributed File Service zFS Administration