z/OS

UNIX System Services
APAR OA12251

z/OS

UNIX System Services
APAR OA12251

About this document

This document supports APAR OA12251 for z/OS® UNIX System Services. This document is only available on the z/OS UNIX Web site at:
http://www.ibm.com/servers/eserver/zseries/zos/unix/release/apar.html

Contents

Chapter 1. Changes for z/OS UNIX System Services Planning (Version 1 Release 4)

Notice for APAR OA12251

Throughout the document, all descriptions of the AUTOMOVE function as it relates to the behavior that occurs when an owning system becomes unavailable (for instance, by shutting down, crashing, or leaving the sysplex) should instead refer to the description below in "Customizing BPXPRMxx for shared HFS" on page 13.

Shared HFS in a sysplex

Overview

This chapter describes shared HFS capability available as of OS/390® UNIX® V2R9 for those who participate in a multi-system sysplex. It assumes that you already have a sysplex up. It defines what shared HFS is, introduces you to HFS data sets that exist in a sysplex, and helps you establish that environment. The topics in this chapter reflect the tasks you must do.

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Establishing the root data set	"Steps in creating the sysplex root HFS data set" on page 7
Establishing the HFS data set	"Steps in creating the system-specific HFS data sets" on page 8
Mounting the version HFS	"Steps in mounting the version HFS" on page 9
Creating the OMVS couple data set	"Steps in creating an OMVS couple data set (CDS)" on page 11
Updating the COUPLExx data set	"Steps in updating COUPLExx to define the OMVS CDS to XCF" on page 13
Keeping the automount policy consistent on all systems in the sysplex	"Steps in keeping your automount policy consistent on all systems" on page 27

Although IBM® recommends that you exploit shared HFS support, you are not required to. If you choose not to, you will continue to share HFS data sets as you have before OS/390 UNIX V2R9. To see how your file system structure differs in OS/390 UNIX V2R9 from V2R8, see "Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or later environments" on page 3.

z/OS Parallel Sysplex Test Report describes how IBM's integration test team implemented shared HFS.

What does shared HFS mean?

Sysplex users can access data throughout the file hierarchy.

The best way to describe the benefit of this function is by comparing what was the file system sharing capability prior to OS/390 UNIX V2R9 with the capability that exists now. Consider a sysplex that consists of two systems, SY1 and SY2:

- Users logged onto SY1 can write to the directories on SY1. For users on SY1 to make a change to file systems mounted on SY2's **/u** directory, they would have to log onto SY2.
- The system programmer who makes configuration changes for the sysplex needs to change the entries in the **/etc** file systems for SY1 and SY2. To make the changes for both systems, the system programmer must log onto each system.

With shared HFS, all file systems that are mounted by a system participating in shared HFS are available to all participating systems. In other words, once a file system is mounted by a participating system, that file system is accessible by any other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. Consider a OS/390 UNIX V2R9 sysplex that consists of two systems, SY1 and SY2:

- A user logged onto any system can make changes to file systems mounted on **/u**, and those changes are visible to all systems.
- The system programmer who manages maintenance for the sysplex can change entries in both **/etc** file systems from either system.

In this chapter, the term *participating group* is used to identify those systems that belong to the same SYSBPX XCF sysplex group and have followed the required installation and migration activities to participate in shared HFS. To be in the participating group, the system level must be at OS/390 UNIX V2R9 or later. Systems earlier than OS/390 UNIX V2R9 can coexist in the sysplex with systems using shared HFS support, but the earlier systems will only be able to share file systems mounted on other systems in read-only mode, and not in read/write mode.

With shared HFS, there is greater availability of data in case of system outage. There is also greater flexibility for data placement and the ability for a single BPXPRMxx member to define all the file systems in the sysplex.

How the end user views the HFS

This chapter describes the kinds of file systems and data sets that support the shared HFS capability in the sysplex. Figure 1 shows that, to the end users, the logical view of the HFS does not change for OS/390 UNIX V2R9. From their point of view, accessing files and directories in the system is just the same. That is true for all end users, whether they are in a sysplex or not.

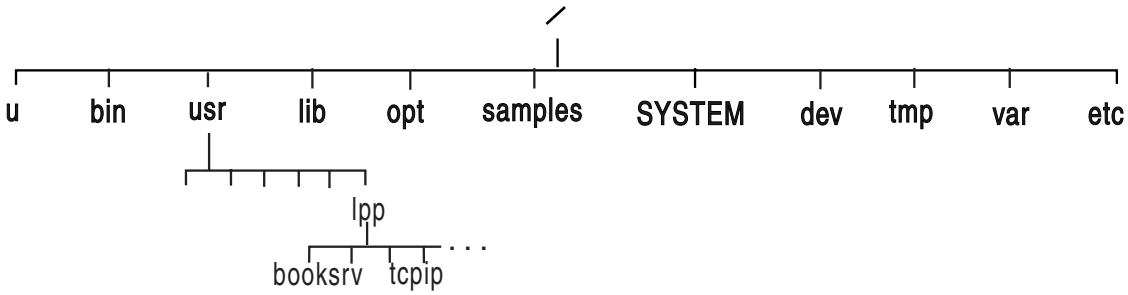

Figure 1. Logical view of shared HFS for the end user

This logical view applies to the end user only. However, system programmers need to know that the illustration of directories found in Figure 1 does not reflect the

physical view of file systems. Starting in OS/390 UNIX V2R9, some of the directories are actually symbolic links, as is described in the following information.

Summary of new HFS data sets

This chapter introduces HFS data sets and terms needed to use shared HFS. Table 1 summarizes the HFS data sets that are needed in a sysplex that has shared HFS. As you study the illustrations of file system configurations in this chapter, you can refer back to this table.

Table 1. HFS data sets that exist in a sysplex

Name	Characteristics	Purpose	How Created
Sysplex root	It contains directories and symbolic links that allow redirection of directories. Only one sysplex root HFS is allowed for all systems participating in shared HFS.	The sysplex root is used by the system to redirect addressing to other directories. It is very small and is mounted read-write. See "Procedures for establishing shared HFS in a sysplex" on page 7for a more complete description of the sysplex root HFS.	The user runs the BPXISYSR job.
System-specific System specific	It contains data specific to each system, including the **/dev**, **/tmp**, **/var**, and **/etc** directories for one system. There is one system-specific HFS data set for each system participating in the shared HFS capability.	The system-specific HFS data set is used by the system to mount system-specific data. It contains the necessary mount points for system-specific data and the symbolic links to access sysplex-wide data, and should be mounted read-write. See "Steps in creating the system-specific HFS data sets" on page 8 for a complete description of the system-specific HFS.	The user runs the BPXISYSS job on each participating system.
Version In a sysplex, *version HFS* is the new name for the root HFS.	It contains system code and binaries, including the **/bin**, **/usr**, **/lib**, **/opt**, and **/samples** directories. IBM delivers only one version root; you might define more as you add new system levels and new maintenance levels.	The version HFS has the same purpose as the root HFS in the non-sysplex world. It should be mounted read-only. See "Steps in mounting the version HFS" on page 9 for a complete description of the version HFS.	IBM supplies this HFS in the ServerPac. CBPDO users create the HFS by following steps defined in the Program Directory.

Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or later environments

The illustrations in this section show you how the file system structures that existed before OS/390 UNIX V2R9 compare with the structures in OS/390 UNIX V2R9 and

later. IBM's recommendations for several releases prior to OS/390 UNIX V2R9 has been that you separate the system setup parameters from the file system parameters so that each system in the sysplex has two BPXPRMxx members: a system limits member and a file system member. In the shared HFS environment, that separation of system limit parameters from file system parameters is even more important. In the shared HFS environment, each system will continue to have a system limits BPXPMRxx member. As you will see in sections that follow, with shared HFS, you might have a file system BPXPRMxx member for each participating system or you might replace those individual file system BPXPRMxx members with a single file system BPXPRMxx member for all participating systems.

File systems in single system pre-OS/390 UNIX V2R9 Environments

The following example shows what BPXPRMxx file system parameters would look like in a single system environment (before OS/390 UNIX V2R9) with no regard to sysplex.

BPXPRMxx

```
FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.ETC.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/etc')
.
.
.
```

Figure 2. BPXPRMxx for a single system before OS/390 UNIX V2R9 or later environments

The root can be mounted either read-only or read-write.

Figure 3 on page 5 shows the recommended setup of the root HFS in a single system environment.

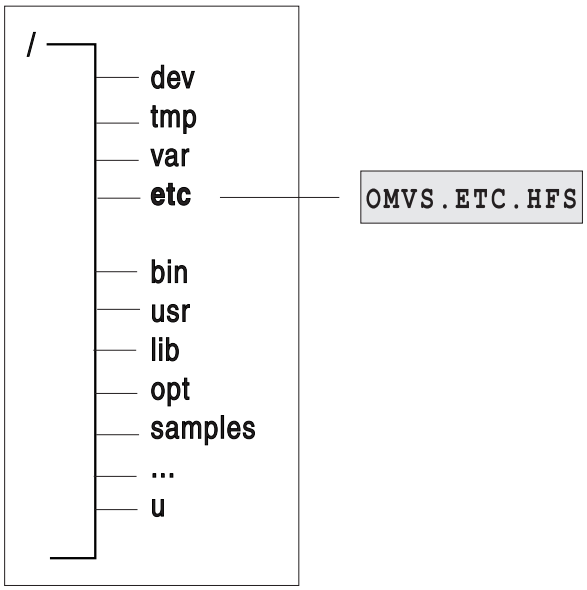

Figure 3. Single system before OS/390 UNIX V2R9

The directories in the root HFS represent “first-level” directories created by IBM. The **/etc**, **/dev**, **/var**, **/tmp**, and **/u** directories are used as mount points for other HFS data sets.

File systems in single system OS/390 UNIX V2R9 or later environments

Figure 4 on page 6 shows what BPXPRMxx file system parameters would look like in an OS/390 UNIX V2R9 (or later) single system environment, and Figure 5 on page 6 shows the corresponding single system image. SYSPLEX(NO) is specified (or the default taken), and the mount mode is read-write.

Note: The root can be mounted either read-only or read-write.

```
BPXPRMxx

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

SYSPLEX(NO)

ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.DEV.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/dev')

MOUNT
FILESYSTEM('OMVS.TMP.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/tmp')

MOUNT
FILESYSTEM('OMVS.VAR.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/var')

MOUNT
FILESYSTEM('OMVS.ETC.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/etc')
```

Figure 4. BPXPRMxx parmlib member for single system: OS/390 UNIX V2R9

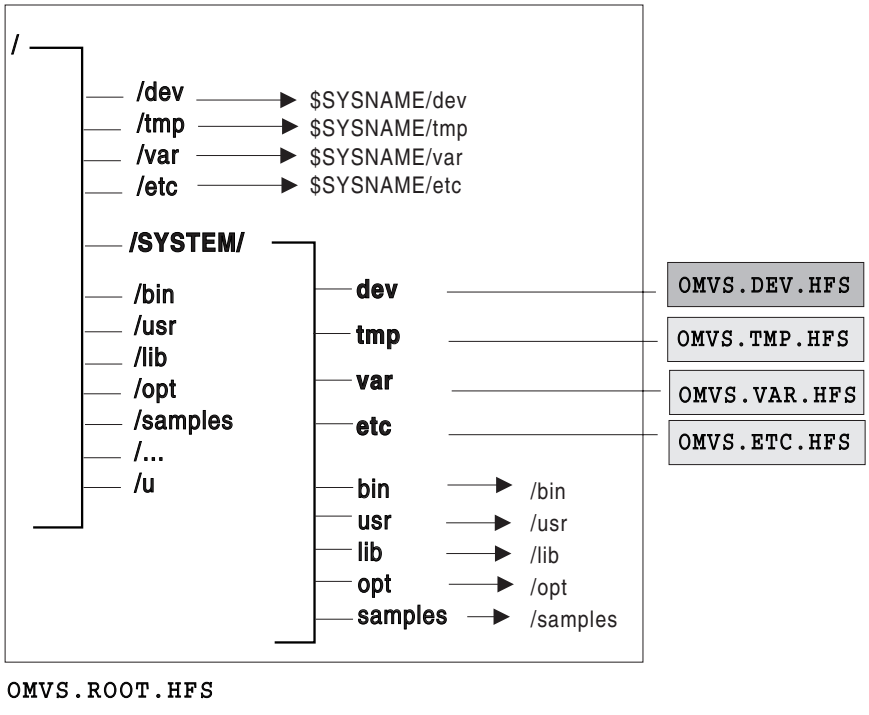

Figure 5. Single system: OS/390 UNIX V2R9

The presence of symbolic links is transparent to the user. In the illustrations used throughout this chapter, symbolic links are indicated with an arrow.

In Figure 5 on page 6, the root file system contains an additional directory, **/SYSTEM**; existing directories, **/etc, /dev, /tmp** and **/var** are converted into symbolic links. These changes, however, are transparent to the user who brings up a single system environment.

Note: If the content of the symbolic link begins with $SYSNAME and SYSPLEX is specified NO, then $SYSNAME is replaced with /SYSTEM when the symbolic link is resolved.

File systems in OS/390 UNIX V2R9 or later sysplex environments

This section describes file systems in sysplex environments (OS/390 UNIX V2R9 or later) and what you need to do to take advantage of shared HFS support, such as creating specific HFS data sets (also see Table 1 on page 3) and the OMVS couple data set, updating COUPLExx, and customizing BPXPRMxx.

You must not assume that with shared HFS, two systems can share a common HFS data set for **/etc**, **/tmp**, **/var**, and **/dev**. This is not the case. Even with shared HFS, each system must have specific HFS data sets for each of these file systems. The file systems are then mounted under the system-specific HFS (see Figure 14 on page 23). With shared HFS support, one system can access system-specific file systems on another system. (The existing security model remains the same.) For example, while logged onto SY2, you can gain read-write access to SY1's **/tmp** by specifying **/SY1/tmp/**.

You should also be aware that when SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating group must be defined for all systems participating in shared HFS. The easiest way to accomplish this is to create a single BPXPRMxx member that contains file system information for each system participating in shared HFS. If you decide to define a BPXPRMxx member for each system, the FILESYSTYPE statements must be identical on each system. To see the differences between having one BPXPRMxx member for all participating systems and having one member for each participating system, see the two examples in "Scenario 2: Multiple systems in the sysplex – using the same release level" on page 21.

In addition, facilities required for a particular file system must be initiated on all systems in the participating group. For example, NFS requires TCP/IP; if you specify a filesystype of NFS, you must also initialize TCP/IP when you initialize NFS, even if there is no network connection.

Procedures for establishing shared HFS in a sysplex

Steps in creating the sysplex root HFS data set

The sysplex root is an HFS data set that is used as the sysplex-wide root. This HFS data set must be mounted read-write and designated AUTOMOVE (see "Customizing BPXPRMxx for shared HFS" on page 13 for a description of the AUTOMOVE parameter in BPXPRMxx). Only one sysplex root is allowed for all systems participating in shared HFS. The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB. After the job runs, the sysplex root file system structure would look like Figure 6 on page 8:

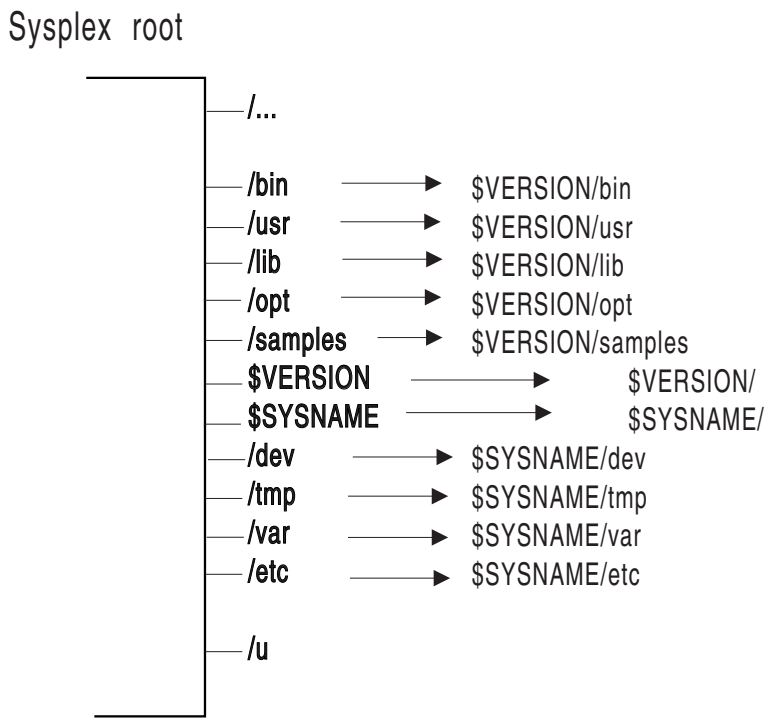

Figure 6. Sysplex root

No files or code reside in the sysplex root data set. It consists of directories and symbolic links only, and it is a small data set.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequent as possible.

Steps in creating the system-specific HFS data sets

Directories in the system-specific HFS data set are used as mount points, specifically for **/etc**, **/var**, **/tmp**, and **/dev**. To create the system-specific HFS, run the BPXISYSS sample job in SYS1.SAMPLIB on each participating system (in other words, you must run the sample job separately for each system that will participate in shared HFS). After you invoke the job, the system-specific file system structure would look like Figure 7 on page 9:

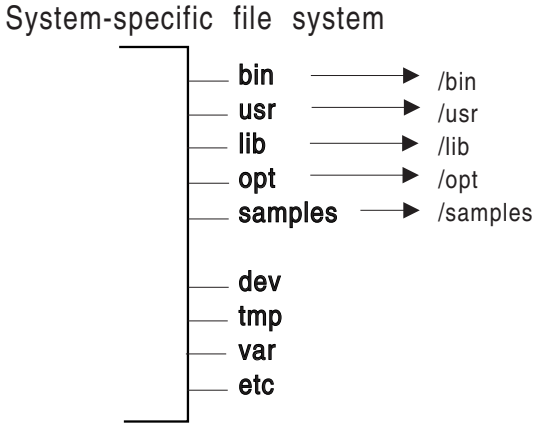

Figure 7. System HFS

The system-specific HFS data set should be mounted read-write, and should be designated NOAUTOMOVE (see “Customizing BPXPRMxx for shared HFS” on page 13 for a description of the NOAUTOMOVE parameter in BPXPRMxx). **/etc**, **/var**, **/tmp**, and **/dev** should also be mounted NOAUTOMOVE. In addition, IBM recommends that the name of the system-specific data set contain the system name as one of the qualifiers. This allows you to use the &SYSNAME symbolic (defined in IEASYMxx) in BPXPRMxx.

Note: If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at **/dev** for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Steps in mounting the version HFS

The version HFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root HFS data set, “root HFS” has been renamed to “version HFS”.

Figure 8 on page 10 shows a version HFS.

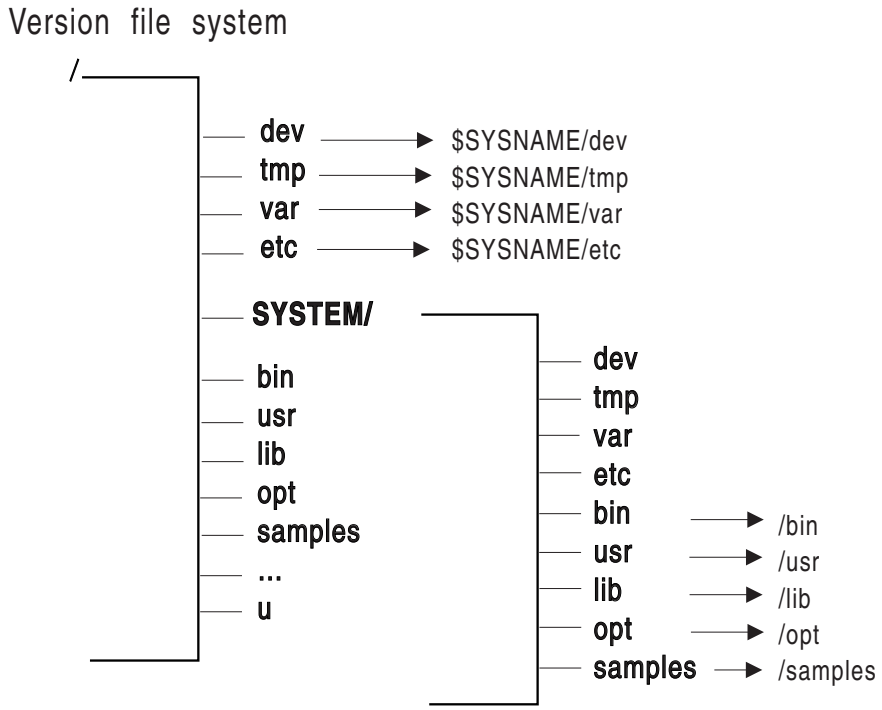

Figure 8. Version HFS

Recommendations:

1. IBM recommends that you mount the version HFS read-only in a sysplex environment, and that you designate it AUTOMOVE. The mount point for the version HFS is dynamically created if the VERSION statement is used in BPXPRMxx.
2. IBM does not recommend using &SYSNAME as one of the qualifiers for the version HFS data set name. In “Sysplex scenarios showing shared HFS capability” on page 18, REL9 and REL9A are used as qualifiers, which correspond to the system release levels. However, you do not necessarily have to use the same qualifiers. Other appropriate names are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer.

IBM supplies the version HFS in ServerPac. CBPDO users obtain the version HFS by following directions in the Program Directory. There is one version HFS for each set of systems participating in shared HFS and who are at the same release level (that is, using the same SYSRES volume). In other words, each version HFS denotes a different level of the system or a different service level. For example, if you have 20 systems participating in shared HFS and 10 of those systems are at OS/390 UNIX V2R9 and the other 10 are at z/OS UNIX V1R1, then you'll have one version HFS for the OS/390 V2R9 systems and one for the z/OS UNIX V1R1 systems. In essence, you will have as many version HFSes for the participating systems as you have different levels running.

Before you mount your version HFS read-only, you may have some element-specific actions. These are described in the table on post-installation actions for mounting the root file system in read-only mode in the chapter on managing the hierarchical file system.

Using the automove system list

When mounting file systems in the sysplex, you can specify a prioritized automove system list to indicate where the file system should or should not be moved to when ownership of a file system changes due to any of the following:

- A soft shutdown request is issued.
- Dead system takeover occurs (when a system leaves the sysplex without a prior soft shutdown).
- A PFS terminates on the owning system.
- A request to move ownership of the file system is issued.

There are different ways to specify the automove system list.

- On the MOUNT statement in BPXPRMxx, specify the AUTOMOVE keyword, including the indicator and system list.
- For the TSO MOUNT command, specify the AUTOMOVE keyword, including the indicator and system list.
- Use the MOUNT shell command.
- Use the ISHELL MOUNT interface.
- Specify the MNTE_SYSLIST variable for REXX.
- Specify the indicator and system list for the automove option in the **chmount** shell command.
- Specify the indicator and system list for the automove option in the SETOMVS operator command.

Steps in creating an OMVS couple data set (CDS)

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the **primary** and the other is a backup that is referred to as the **alternate**. In BPXISCDS, you also specify the number of mount records that are supported by the CDS.

Use of the CDS functions in the following manner:

1. The first system that enters the sysplex with SYSPLEX(YES) initializes an OMVS CDS. The CDS controls shared HFS mounts and will eventually contain information about all systems participating in shared HFS.

   This system processes its BPXPRMxx parmlib member, including all its ROOT and MOUNT statement information. It is also the designated owner of the byte range lock manager for the participating group. The MOUNT and ROOT information are logged in the CDS so that other systems that eventually join the participating group can read data about systems that are already using shared HFS.
2. Subsequent systems joining the participating group will read what is already logged in the CDS and will perform all mounts. Any new BPXPRMxx mounts are processed and logged into the CDS. Systems already in the participating group will then process the new mounts added to the CDS.

Following is the sample JCL with comments. The statements in bold contain the values that you specify based on your environment.

```
//*
//STEP10    EXEC PGM=IXCL1DSU
//STEPLIB   DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
```

```
/* Begin definition for OMVS couple data set(1)           */
 DEFINEDS SYSPLEX(PLEX1)
  /* Name of the sysplex in which the OMVS couple data set is to be used.*/
  DSN(SYS1.OMVS.CDS01) VOLSER(3390x1)
   /* The name and volume for the OMVS couple data set.
   The utility will allocate a new data set by the name specified on the
   volume specified.*/
   MAXSYSTEMS(8)
   /* Specifies the number of systems to be supported by the OMVS CDS.
       Default =     8          */
       NOCATALOG
   /* Default is not to CATALOG */
   DATA TYPE(BPXMCDS)
   /* The type of data in the data set being created for OMVS.
   BPXMCDS is the TYPE for OMVS. */
ITEM NAME(MOUNTS) NUMBER(500)
 /* Specifies the number of MOUNTS that can be supported by OMVS.*/
    Default =   100
    Suggested minimum =    10
    Suggested maximum = 35000  */
 ITEM NAME(AMTRULES) NUMBER(50)
  /* Specifies the number of automount rules that can be supported by OMVS.*/
     Default =    50
     Minimum =    50
     Maximum =  1000           */

  ITEM NAME(DISTBRLM) NUMBER(1)
  /*Enable conversion to a distributed BRLM.
    1, distributed BRLM enabled,
    0, distributed BRLM is not enabled during next sysplex IPL
    Default = 0  */
   /* Begin definition for OMVS couple data set(2)              */
  DEFINEDS SYSPLEX(PLEX1)
  /* Name of the sysplex in which the OMVS couple data set is to be used.      */
  DSN(SYS1.OMVS.CDS02) VOLSER(3390x2)
  /* The name and volume for the OMVS couple data set.  The utility will
     allocate a new data set by the namespecified on the volume specified.     */
   MAXSYSTEMS(8)
   /* Specifies the number of systems to be supported by the OMVS CDS.
      Default =     8           */
      NOCATALOG
   /* Default is not to CATALOG */
      DATA TYPE(BPXMCDS)
   /* The type of data in the data set being created is for OMVS.  BPXMCDS is the
      TYPE for OMVS.            */
    ITEM NAME(MOUNTS) NUMBER(500)
   /* Specifies the number of MOUNTS that can be supported by OMVS.
      Default =   100
      Suggested minimum =    10
      Suggested maximum = 35000 */
    ITEM NAME(AMTRULES) NUMBER(50)
   /* Specifies the number of automount rules that can be supported by OMVS.
      Default =     50
      Minimum =     50
      Maximum =   1000     */
    ITEM NAME(DISTBRLM) NUMBER(1)
    /*Enables conversion to a distributed BRLM.
      1, distributed BRLM enabled,
      0, distributed BRLM is not enabled during next sysplex IPL
      Default = 0  */
```

Rule: Automount mounts must be included in the MOUNTS value. The number of automount mounts is the expected number of concurrently mounted file systems using the automount facility. For example, you may have specified 1000 file systems to be automounted, but if you expect only 50 to be used concurrently, you should factor these 50 into your MOUNTS value.

For more information about setting up a sysplex on MVS™ and descriptions of XCF and CDS, see *z/OS MVS Setting Up a Sysplex*.

The NUMBER(*nnnn*) specified for mounts and automount rules (a generic or specific entry in an automount map file) is directly linked to function performance and the size of the CDS. If maximum values are specified, the size of the CDS will increase accordingly and the performance level for reading and updating it will decline.

Conversely, if the NUMBER values are too small, the function (for example, the number of mounts supported) would fail after the limit is reached. However, a new CDS can be formatted and switched in with larger values specified in NUMBER. To make the switch, issue the SETXCF COUPLE,PSWITCH command. For more information on this command, see the section on couple data set considerations in *z/OS MVS Setting Up a Sysplex*. The number of file systems required (factoring in an additional number to account for extra mounts), determines your minimum and maximum NUMBER value.

After the CDS is created, it must be identified to XCF for use by z/OS UNIX.

Steps in updating COUPLExx to define the OMVS CDS to XCF: Update the active COUPLExx parmlib member to define a primary and alternate OMVS CDS to XCF. The primary and alternate CDSs should be placed on separate volumes. (The sample JCL in “Steps in creating an OMVS couple data set (CDS)” on page 11 shows the primary CDS on volume 3390x1 and the secondary CDS on 3390x2.)

Figure 9 shows the COUPLExx parmlib member; statements that define the CDS are in bold.

```
/* For all systems in any combination, up to an eightway  */
COUPLE INTERVAL(60)                    /* 1 minute */
   OPNOTIFY(60)                        /* 1 minute */
   SYSPLEX(PLEX1)                      /* SYSPLEX NAME*/
   PCOUPLE(SYS1.PCOUPLE,CPLPKP)        /* COUPLE DS */
   ACOUPLE(SYS1.ACOUPLE,CPLPKA)        /* ALTERNATE DS*/
   MAXMSG(750)
   RETRY(10)
DATA TYPE(CFRM)
   PCOUPLE(SYS1.PFUNCT.CTTEST,FDSPKP)
   ACOUPLE(SYS1.AFUNCT.CTTEST,FDSPKA)
DATA TYPE(BPXMCDS)
   PCOUPLE(SYS1.OMVS.CDS01,3390x1)
   ACOUPLE(SYS1.OMVS.CDS02,3390x2)
/* CTC DEFINITIONS: ALL SYSTEMS  */
PATHOUT DEVICE(8E0)
PATHIN DEVICE(CEF)
```

Figure 9. COUPLExx parmlib member

The MVS operator commands (DISPLAY XCF, SETXCF, DUMP, CONFIG, and VARY) enable the operator to manage the z/OS UNIX CDS. For a complete description of these commands, see *z/OS MVS System Commands*.

Customizing BPXPRMxx for shared HFS

HFS sharing enables you to use one BPXPRMxx member to define all the file systems in the sysplex. This means that each participating system has its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of

system symbolics. Figure 12 on page 21 shows an example of this unified member. You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex. An example of this is Figure 13 on page 22.

The following parameters set up HFS sharing in a sysplex:

- SYSPLEX(YES) sets up HFS sharing for those who are in the SYSBPX XCF group, the group that is participating in HFS data sharing. To participate in HFS data sharing, the systems must be at the OS/390 V2R9 level or later. Those system that specify SYSPLEX(YES) make up the participating group for the sysplex.

  If SYSPLEX(YES) is specified in the BPXPRMxx member, but the system is initialized in XCF-local mode, either by specifying COUPLE SYSPLEX(LOCAL) in the COUPLExx member or by specifying PLEXCFG=XCFLOCAL in the IEASYSxx member, then the kernel will ignore the SYSPLEX(YES) value and initialize with SYSPLEX(NO). This system will not participate in shared HFS support after the initialization completes.
- VERSION('nnnn') allows multiple releases and service levels of the binaries to coexist and participate in HFS sharing. *nnnn* is a qualifier to represent a level of the version HFS. The most appropriate values for *nnnn* are the name of the target zone, &SYSR1, or another qualifier that is meaningful to the system programmer. A directory with the value *nnnn* specified on VERSION will be dynamically created at system initialization under the sysplex root and will be used as a mount point for the version HFS.

  There is one version HFS for every instance of the VERSION parameter. More information about version HFS appears in “Steps in mounting the version HFS” on page 9.
- The SYSNAME(sysname) parameter on ROOT and MOUNT statements specifies the particular system on which a mount should be performed. *sysname* is a 1–8 alphanumeric name of the system. This system will then become the owner of the file system mounted. The owning system must be IPLed with SYSPLEX(YES).

  Recommendation: Specify SYSNAME(&SYSNAME.) or omit the SYSNAME parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

  The SYSNAME parameter is also used with SETOMVS when moving file systems, as demonstrated in “Moving file systems in a sysplex” on page 27.

The AUTOMOVE|NOAUTOMOVE|UNMOUNT parameters on the ROOT and MOUNT statements indicate what happens to the ownership of a file system in the following situations:

- A shutdown process is issued (soft shutdown)
- Dead system takeover occurs (a system leaves the sysplex without a prior soft shutdown)
- A PFS terminates on the owning system
- A request to move ownership of a file system is issued

The action taken is determined by the AUTOMOVE value and the sysplex-awareness capability of the file system. AUTOMOVE is the default, specifying that ownership of the file system is automatically moved to another system.

The owner of a file system is the first system that processes the mount. This system always accesses the file system locally; that is, the system does not access the file system through a remote system. Other non-owning systems in the sysplex

access the file system either locally or through the remote owning system, depending on the PFS and the mount mode. If a PFS allows a file system to be locally accessed on all systems in a sysplex for a particular mode, then the PFS is *sysplex-aware* for that mode. If a PFS requires that a file system be accessed through the remote-owning system from all other systems in a sysplex for a particular mode, then the PFS is *sysplex-unaware* for that mode.

Even if a PFS is sysplex-aware for a particular mode, if a non-owning system does not have DASD connectivity, the file system is accessed remotely through the owning system. For example, HFS is sysplex-unaware for read-write mode, because all non-owning systems must access read-write file systems through the remote owning system. The non-owning systems are said to be *sysplex clients*.

However, HFS is sysplex-aware for read-only mode, which means that each system can access read-only file systems locally, and does not need to contact the owning system. AUTOMOVE is intended for sysplex-unaware file systems, where non-owning systems access the file systems remotely, to allow you to specify what will happen to the ownership of file systems when shutdown, PFS termination, dead system takeover, or move file system occur.

TFS file systems do not participate in move operations regardless of the AUTOMOVE setting. Automount-managed file systems are handled as AUTOMOVE if the file system is used locally.

Restriction: An AUTOMOVE file system cannot be moved to a system where OMVS has been shut down or where `F BPXOINIT,SHTUDOWN=FILEOWNER` has been used.

Table 2 shows what happens during soft shutdown for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems. Soft shutdown is done by issuing one of the following MODIFY commands:

```
F BPXOINIT,SHUTDOWN=FILESYS
F BPXOINIT,SHTUDOWN=FILEOWNER
F OMVS,SHUTDOWN
```

A *leaf file system* refers to a file system that does not contain any file systems that are mounted on a directory within that file system. A *subtree* is the file system and all file systems that are mounted beneath that file system.

Table 2. Soft shutdown actions for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Unmounts the file system. The unmount will fail if the file system being unmounted is not a leaf file system.	Unmounts the file system. The unmount will fail if the file system being unmounted is not a leaf file system.
AUTOMOVE (no system list)	Move is attempted to any system. If the move fails, the unmount is not attempted and ownership does not change.	Move is attempted to any system.. If the move fails, the unmount is not attempted and ownership does not change.
AUTOMOVE with a system list	The move uses the system list. If the file system cannot be moved, then the unmount is not attempted and ownership does not change.	The move uses the system list. If the file system cannot be moved, the unmount is not attempted and ownership does not change.

Note: Automount-managed file systems are unmounted by a soft shutdown operation if the file system is not referenced by any other system in the

sysplex. If it is referenced by another system or systems, ownership of the file system is moved. If the move fails, an unmount is not attempted and ownership does not change.

Table 3 shows what happens during dead system takeover for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems. *Dead system takeover* is the action taken by systems in a sysplex when they attempt to take over ownership of file systems that were previously owned by a system that has just left the sysplex.

Table 3. Dead system takeover for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE	Takeover is attempted by all systems. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is not attempted. The file system becomes unowned.
UNMOUNT	Takeover is attempted by all systems. The subtree is unmounted if it cannot be taken over by a new owning system.	Takeover is not attempted. The subtree is unmounted.
AUTOMOVE (no system list)	Takeover is attempted. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is attempted. The file system becomes unowned if it cannot be taken over by a new owning system.
AUTOMOVE with a system list	Takeover is attempted and the INCLUDE or EXCLUDE system list is honored. If the takeover does not happen, the subtree is unmounted.	Takeover is attempted and the INCLUDE or EXCLUDE system list is honored. If the takeover does not happen, the subtree is unmounted.

Note: There is no attempt to take over automount-managed file systems if the file system is not referenced by any system that is eligible to attempt takeover. Automount-managed, unowned file systems will be unmounted.

Table 4 shows what happens during PFS termination for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems.

Table 4. PFS termination for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	The subtree is unmounted.	The subtree is unmounted.
AUTOMOVE (no system list)	Moves to any system. If the move fails, the subtree is unmounted.	Moves to any system. If the move fails, the subtree is unmounted.
AUTOMOVE with a system list	The move uses the system list. If the move fails, the subtree is unmounted.	The move uses the system list. If the move fails, the subtree is unmounted.

Table 5 on page 17 shows what happens when a move file system is requested to move a specific file system to any target system (wildcard is used). A move file system request can be issued with a SETOMVS operator command or a **chmount** shell command.

Table 5. Move a specific file system to any system for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT or AUTOMOVE (no system list)	Move is attempted to all systems.	Move is attempted to all systems.
AUTOMOVE with a system list	Move is attempted using the system list.	Move is attempted using the system list.

Table 6 shows what happens when a move file system is requested to do a multi-file system move, moving all file systems from a system to a specific target system. A move file system request can be issued with a SETOMVS operator command or a **chmount** shell command.

Table 6. Move all file systems from a system to a specific target system for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Move is not attempted.	Move is not attempted.
AUTOMOVE (no system list)	Move is attempted to the target system.	Move is attempted to the target system.
AUTOMOVE with a system list	Move is attempted to the target system, ignoring the system list.	Move is attempted to the target system, ignoring the system list.

Rules:

- Define your version and sysplex root file systems as AUTOMOVE.
- Define your system-specific file systems as UNMOUNT.
- Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system underneath it as AUTOMOVE. If you do, the file system defined as AUTOMOVE will not be recovered after a system failure until that failing system has been restarted.

Guidelines:

1. To ensure that the root is always available, use the default, which is AUTOMOVE.
2. For sysplex-unaware file systems that are mostly exported by the DFS™ or SMB server to their remote clients, consider specifying NOAUTOMOVE on the MOUNT statement. Then the file systems will not change ownership if the system is suddenly recycled, and they will be available for automatic re-export by DFS or SMB.

   Specifying NOAUTOMOVE is suggested because a file system can only be exported by the DFS or SMB server at the system that owns the file system. A file system can only be exported by the DFS or SMB server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported by DFS. The same holds true of file systems exported by SMB. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS or SMB clients. When an owning system recycles and a file system exported by DFS or SMB has been taken over by one of the other systems, DFS or SMB cannot automatically re-export that file system. When an owning system is recycled and an exported

file system has been taken over by one of the other systems, that file system will not be automatically reexported. The file system will have to be moved from its current owner back to the original system, the one that has just been recycled, and then exported again.

For more information about VERSION, SYSPLEX, SYSNAME and AUTOMOVE|NOAUTOMVE|UNMOUNT, see *z/OS MVS Initialization and Tuning Reference*.

Sysplex scenarios showing shared HFS capability

Scenario 1: First system in the sysplex

Figure 10 and Figure 11 on page 20 shows a z/OS UNIX file system configuration for shared HFS. Here, SYSPLEX(YES) and a value on VERSION are specified, and a directory is dynamically created on which the version HFS data set is mounted. This type of configuration requires a sysplex root and system-specific HFS.

```
BPXPRMxx for (SY1)

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')          1
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')   2
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')               3
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')          4
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')          5
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 10. BPXPRMxx parmlib setup — HFS sharing

1 This is the sysplex root HFS data set, and was created by running the BPXISYSR job. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

2 This is the system-specific HFS data set, and was created by running the BPXISYSS job. It must be mounted read-write. NOAUTOMOVE is specified because this file system is system-specific and ownership of the file system should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME.** will resolve to **/SY1** during parmlib processing. This mount point is created dynamically at system initialization.

3 This is the old root HFS (version HFS).

Recommendation: It should be mounted read-only. Its mount point is created dynamically and the name of the HFS is the value specified on the VERSION statement in the BPXPRMxx member. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

4 This HFS contains the system-specific **/dev** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./dev** will resolve to **/SY1/dev** during parmlib processing.

5 This HFS contains system-specific **/tmp** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./tmp** will resolve to **/SY1/tmp** during parmlib processing.

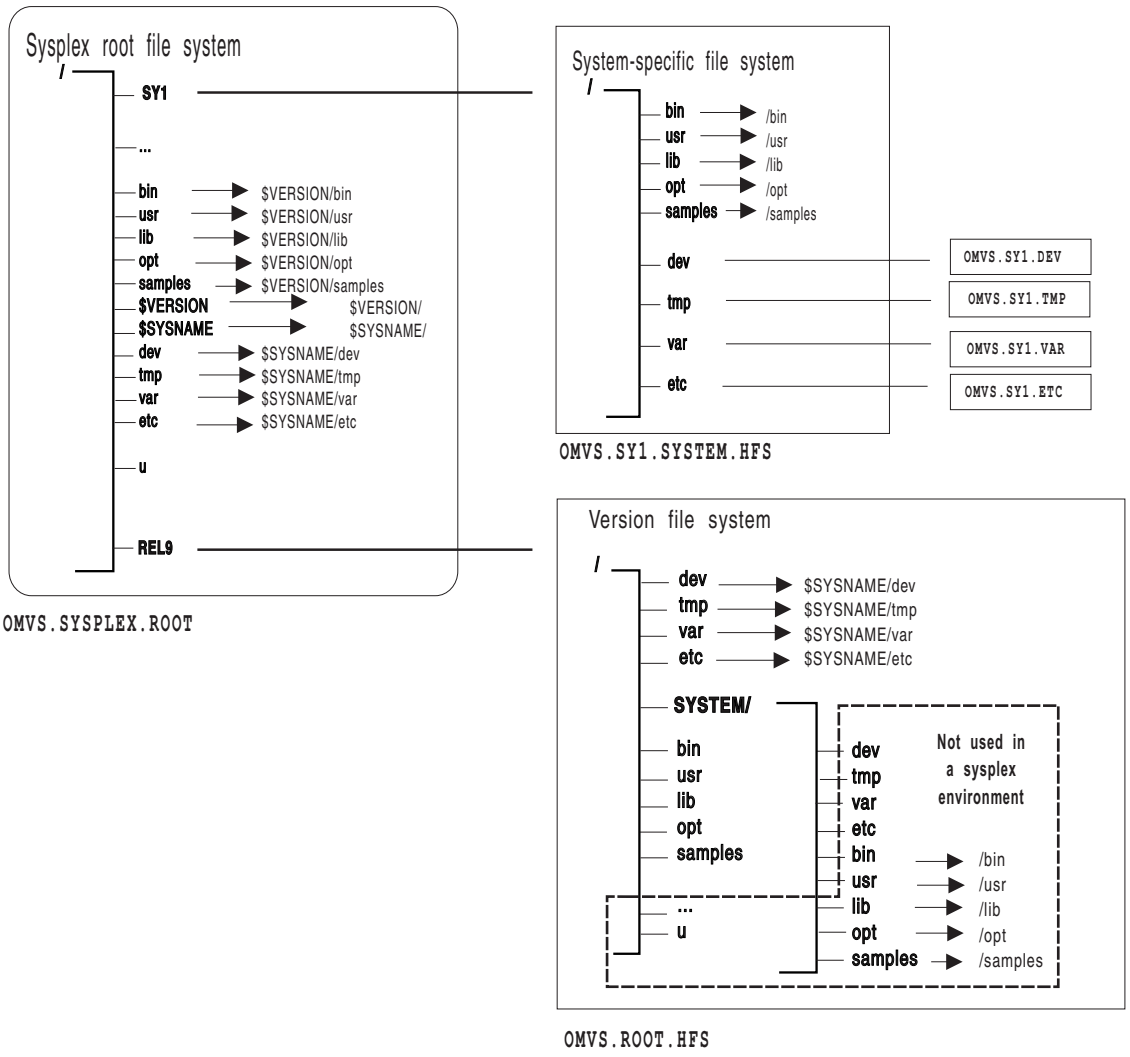

Figure 11. HFS sharing in a sysplex

If the content of the symbolic link begins with $VERSION or $SYSNAME, the symbolic link will resolve in the following manner:

- If you have specified SYSPLEX(YES) and the symbolic link for **/dev** has the contents **$SYSNAME/dev**, the symbolic link resolves to **/SY1/dev** on system SY1 and **/SY2/dev** on system SY2.
- If you have specified SYSPLEX(YES) and the content of the symbolic link begins with $VERSION, $VERSION resolves to the value *nnnn* specified on the VERSION parameter. Thus, if VERSION in parmlib is set to REL9, then $VERSION resolves to /REL9. For example, a symbolic link for **/bin**, which has the contents **$VERSION/bin**, resolves to **/REL9/bin** on a system whose $VERSION value is set to REL9.

In the above scenario, if **ls –l /bin/** is issued, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the pathname **/REL9/bin**. The contents of **/REL9/bin** will now be displayed.

Scenario 2: Multiple systems in the sysplex – using the same release level

Figure 14 on page 23 shows another SYSPLEX(YES) configuration. In this configuration, however, two or more systems are sharing the same version HFS (the same release level of code). Figure 12 shows a sample BPXPRMxx for the entire sysplex (what IBM recommends) using &SYSNAME. as a symbolic name, and Figure 13 on page 22 shows a configuration where each system in the sysplex has its own BPXPRMxx. For our example, SY1 has its own BPXPRMxx and SY2 has its own BPXPRMxx.

```
One BPXPRMxx Member to Define File Systems for the Entire Sysplex

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 12. Sharing HFS data sets: one version HFS and one BPXPRMxx for the entire sysplex

```
BPXPRMS1 (for SY1)                      BPXPRMS2 (for SY2)

FILESYSTYPE                             FILESYSTYPE
TYPE(HFS)                               TYPE(HFS)
ENTRYPOINT(GFUAINIT)                    ENTRYPOINT(GFUAINIT)
PARM(' ')                               PARM(' ')

VERSION('REL9')                         VERSION('REL9')
SYSPLEX(YES)                            SYSPLEX(YES)

ROOT                                    ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')         FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                    TYPE(HFS) MODE(RDWR)

MOUNT                                   MOUNT
FILESYSTEM('OMVS.SY1.SYSTEM.HFS')       FILESYSTEM('OMVS.SY2.SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1')                      MOUNTPOINT('/SY2')

MOUNT FILESYSTEM('OMVS.ROOT.HFS')       MOUNT FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)                    TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                 MOUNTPOINT('/$VERSION')

MOUNT FILESYSTEM('OMVS.SY1.DEV')        MOUNT FILESYSTEM('OMVS.SY2.DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/dev')                  MOUNTPOINT('/SY2/dev')

MOUNT FILESYSTEM('OMVS.SY1.TMP')        MOUNT FILESYSTEM('OMVS.SY2.TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/tmp')                  MOUNTPOINT('/SY2/tmp')
.
.
.
```

Figure 13. Sharing HFS data sets: one version HFS and separate BPXPRMxx members for each system in the sysplex

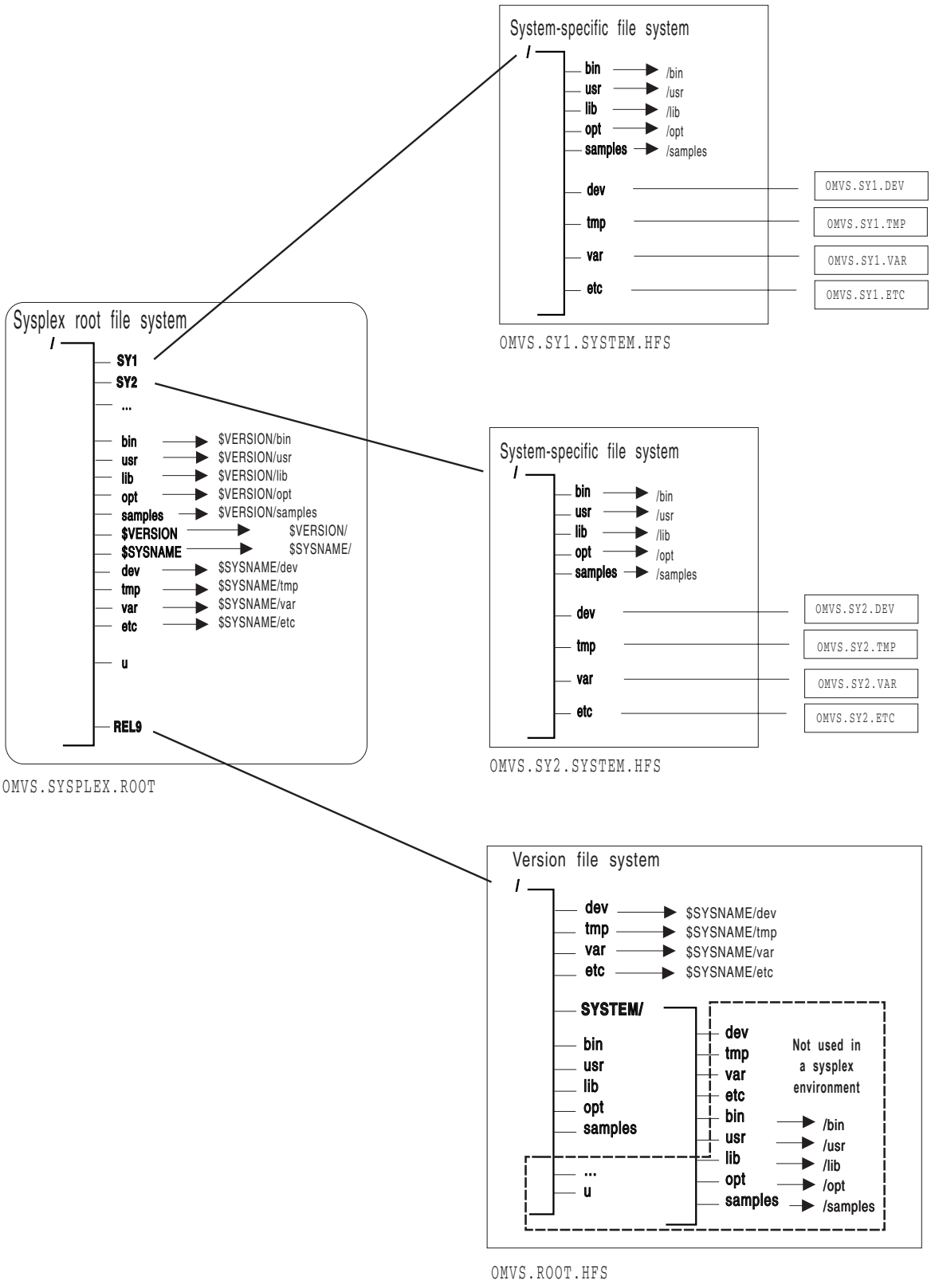

Figure 14. Sharing HFS data sets in a sysplex for Release 9: multiple systems in a sysplex using the same release level

In this scenario, where multiple systems in the sysplex are using the same version HFS, if **ls –l /bin/** is issued from either system, the user expects to see the contents

of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the pathname **/REL9/bin**. The contents of this directory will be displayed.

Scenario 3: Multiple systems in a sysplex using different release levels

If your participating group is in a sysplex that runs multiple levels of z/OS and/or OS/390, your configuration might look like the one in Figure 16 on page 25. In that configuration, each system is running a different level of z/OS and, therefore, has different version HFS data sets; SY1 has the version HFS named OMVS.SYSR9A.ROOT.HFS and SY2 has the version HFS named OMVS.SYSR9.ROOT.HFS. Figure 15 shows two BPXPRMxx parmlib members that define the file systems in this configuration. Figure 17 on page 26 shows a single BPXPRMxx parmlib member that can be used to define this same configuration; it uses &SYSR1. as the symbolic name for the two version HFS data sets.

```
BPXPRMxx (for SY1)                          BPXPRMxx (for SY2)

FILESYSTYPE                                 FILESYSTYPE
TYPE(HFS)                                   TYPE(HFS)
ENTRYPOINT(GFUAINIT)                        ENTRYPOINT(GFUAINIT)
PARM(' ')                                   PARM(' ')

VERSION('REL9A')                            VERSION('REL9')
SYSPLEX(YES)                                SYSPLEX(YES)

ROOT                                        ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')             FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                        TYPE(HFS) MODE(RDWR)

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')     FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')                    MOUNTPOINT('/&SYSNAME.')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.SYSR9A.ROOT.HFS')          FILESYSTEM('OMVS.SYSR9.ROOT.HFS')
TYPE(HFS) MODE(READ)                        TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                     MOUNTPOINT('/$VERSION')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')            FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')                MOUNTPOINT('/&SYSNAME./dev')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')            FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')                MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 15. BPXPRMxx parmlib setup for multiple systems sharing HFS data sets and using different release levels

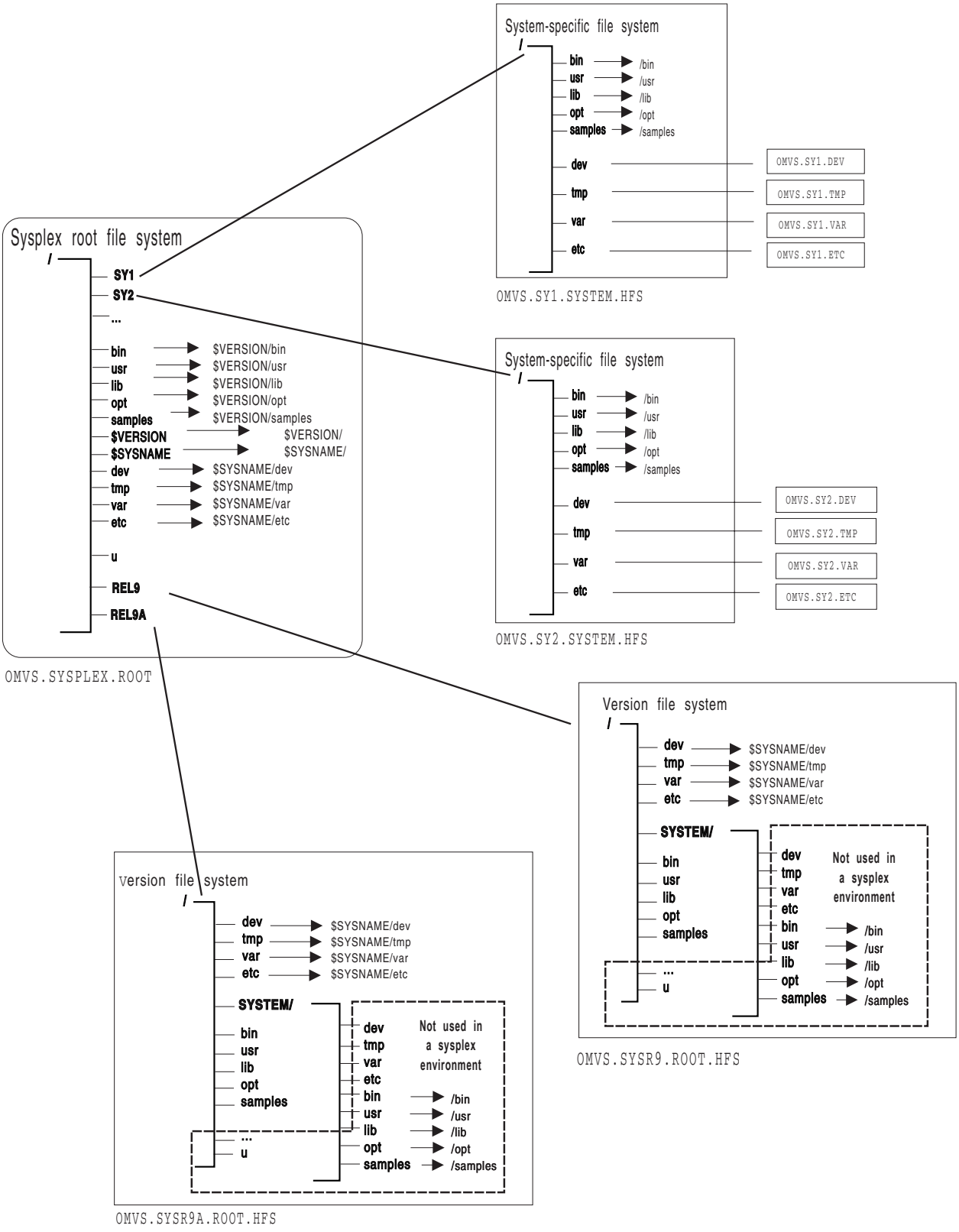

Figure 16. Sharing HFS data sets between multiple systems using different release levels

In this scenario, for example, if **ls –l /bin/** is issued on SY1, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to

/SYSR9A on SY1, which makes the pathname **/SYSR9A/bin**. The contents of this directory will now display. If **ls –l /bin/** is issued on SY2, the contents of **/SYSR9/bin** will display.

From SY2 you can display information on SY1 by fully qualifying the directory. For example, to view SY1's **/bin** directory, you issue **ls –l /SY1/bin/**.

```
One BPXPRMxx Member to define file systems for the entire sysplex
Using different releases

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('&SYSR1.')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.&SYSR1..ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 17. One BPXPRMxx parmlib member for multiple systems sharing HFS data sets and using different release levels

In order to use one BPXPRMxx parmlib file system member, we have used another system symbolic like &SYSR1. This system symbolic is used in the VERSION parameter and also as a qualifier in the version HFS data set name.

Keeping automount policies consistent on all systems in the sysplex

Rule: You must keep the automount policies consistent across all the participating systems in the sysplex. The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

Steps in keeping your automount policy consistent on all systems

Before OS/390 UNIX V2R9, your automount policy most likely resided in the **/etc/auto.master** and **/etc/u.map files**. For those using shared HFS, each participating system has a separate **/etc** file system. In order for automount policy to be consistent across participating systems, the same copy of the automount policy must exist in every system's **/etc/auto.master** and **/etc/u.map** files.

For example both SY1 and SY2 have the following files:

- **/etc/auto.master**

```
/u     /etc/u.map
```

- **/etc/u.map**

```
name       *
type       HFS
filesystem OMVS.<uc_name>.HFS
mode       rdwr
duration   60
delay      60
```

When the automount daemon initializes on SY1, it will read its local **/etc/auto.master** file to identify what directories to manage; in this case, it is **/u**. Next, the automount daemon will use the policy specified in the local **/etc/u.map** file to mount file systems with the specified naming convention under **/u**. The automount daemon on SY2 will perform similar actions. Because all mounted file systems are available to all participating systems in the sysplex, your automount policy must be consistent. This is true for the file system name specified in **/etc/u.map** and the values for other parameters in **/etc/u.map** and **/etc/auto.master**.

Moving file systems in a sysplex

You may need to change ownership of the file system for recovery or re-IPLing.

Tips:

- To check for file systems that have already been mounted, use the **df** command from the shell.
- The SETOMVS command used with the FILESYS, FILESYSTEM, mount point and SYSNAME parameters can be used to move a file system in a sysplex, or you can use the **chmount** command from the shell. However, do not move two types of file systems:
  - System-specific file systems
  - File systems that are being exported by DFS. You have to unexport them from DFS first and then move them

Examples:

1. To move ownership of the file system that contains **/u/wjs** to SY1:

```
chmount -d SY1 /u/wjs
```

2. To move ownership of the payroll file system from the current owner to SY2 using SETOMVS, issue:

```
SETOMVS FILESYS,FILESYSTEM='POSIX.PAYROLL.HFS',SYSNAME=SY2
```

   or (assuming the mount point is over directory **/PAYROLL**)

```
SETOMVS FILESYS,mountpoint='/PAYROLL',SYSNAME=SY2
```

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at **/dev** for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Also, opened FIFO files are automatically closed before the file system containing the FIFO is moved. They are closed because the in-storage FIFO data on the old system is not moved and is no longer accessible on new owning system.

Shared HFS implications during system failures and recovery

File system recovery in a shared HFS environment takes into consideration file system specifications such as AUTOMOVE, NOAUTOMOVE, UNMOUNT, and whether or not the file system is mounted read-only or read-write.

Generally, when an owning system fails, ownership over its automove-mounted file system is moved to another system and the file is usable. However, if a file system is mounted read-write and the owning system fails, then all file system operations for files in that file system will fail. This happens because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered. (The BPX1CLO and BPX1OPN callable services are discussed in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.)

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be started again.

In some situations, even though a file system is mounted AUTOMOVE, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes unowned; if this happens, you will see message BPXF213E. This is true if the file system is mounted either read-write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that are owned by another system are still usable. However, all file operations for the unowned file system will fail until a new owner is established. The shared HFS support will continue to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for shared HFS. Should a subsequent recovery attempt succeed, the file system transitions from the unowned to the active state.

Applications using files in unowned file systems will need to close (BPX1CLO) those files and reopen (BPX1OPN) them after the file system is recovered.

File systems that are mounted NOAUTOMOVE or UNMOUNT will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use.

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

Locking files in the sysplex

You can lock all or part of a file that you are accessing for read-write purposes by using the byte range lock manager (BRLM). As default, the lock manager is initialized on only one system in the sysplex. The first system that enters the sysplex initializes the BRLM and becomes the system that owns the manager. For example, if SY1 is the first system in the sysplex, then SY1 owns the BRLM; all lock requests are routed to SY1.

When a system failure occurs on the system owning the BRLM, all history of byte range locks is lost. A new BRLM is established by one of the surviving systems in the sysplex, and locking can begin once that recovery has completed. However, to maintain locking integrity for files open on surviving systems after the system that owns the BRLM goes down, z/OS UNIX prevents further locking or I/O on opened files that were locked. In addition, the applications are signalled, just in case they never issue locking requests or I/O. Running applications that did not issue locking requests and did not have files open are not affected. After a failure where byte range locks are lost, z/OS UNIX provides the following information to processes that have used byte range locking:

- Access to open files for which byte range locks are held by any process will result in an I/O error. The file must be closed and reopened before use can continue.
- A signal is issued to any process which has made use of byte range locking. By default, a SIGTERM signal is issued against every such process, and an EC6 abend with reason code 0D258038 will terminate the process. If you do not want the process to be terminated, the process can use BPX1PCT (the physical file system control callable service) to specify a different signal for z/OS UNIX to use for notifying the process that the BRLM has failed. Any signal can be used for this purpose, thus allowing the user or application the ability to catch or ignore the signal and react accordingly.

  The system completion code EC6 and its associated reason codes are described in *z/OS MVS System Codes*. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about BPX1PCT.

Using distributed BRLM

Centralized BRLM is set up as the default. You can choose to have distributed BRLM initialized on every system in the sysplex. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system. If you want distributed BRLM, you need to follow the steps in “Steps for setting up distributed BRLM on every system in the sysplex” on page 30.

Recommendation: Use distributed BRLM if you have applications that lock files that are mounted and owned locally. Examples are the **inetd** and **cron** daemons. Distributed BRLM eliminates a single point of failure by having applications like **inetd**, **cron**, and Domino® server send their lock requests to the local distributed BRLM server instead of the remote central BRLM server.

Restrictions: After you set up distributed BRLM, the following restrictions apply:

- You cannot go back to centralized BRLM unless you restore the sysplex to the state it was in before distributed BRLM was set up. This will require resetting the couple data set, followed by a sysplex-wide IPL
- If any open file in the file system has been locked by BRLM, you cannot move file systems by an external command such as `SETOMVS FILESYS, FILESYSTEM=,SYSNAME=`. Before the move can succeed, the file must be closed.

- With distributed BRLM, certain cross-system deadlock scenarios may not be detected. Locking applications must ensure that they do not cause deadlocks.

Steps for setting up distributed BRLM on every system in the sysplex:
Before you begin: You must bring down any locking applications if you are activating distributed BRLM in a running sysplex. If you do not, they will be exposed to any locking and I/O errors for any files that they have open.

Perform the following steps to set up BRLM on every system in the sysplex.

1. Ensure that all systems in the sysplex support BRLM.

   ```
   F BPXOINIT,FILESYS=DISPLAY,GLOBAL
   ```

   In the resulting display, you can check the supported levels. You should see v.p.m, with either m>5 or p>2, or v>1. If a system is not at these levels, you will have to apply APAR OW52293, with these corresponding PTFs: R609 UW85157, R703 UW85155, and R705 UW85156. (R706 automatically includes support equivalent to those PTFs.)

   ---

2. Update the BPXMCDS couple data set using the IXCL1DSU utility.

   Tip: See “Steps in creating an OMVS couple data set (CDS)” on page 11 for an example of the COUPLExx parmlib member. *z/OS MVS Setting Up a Sysplex* discusses the BPXMCDS couple data set and the IXCL1DSU utility.

   ---

3. Switch the CDS into a running sysplex.

   Example:

   ```
   SETXCF COUPLE,PSWITCH,TYPE=BPXMCDS
   ```

   ---

4. Remove the central BRLM server from the sysplex. Bringing it down eliminates all file locking history in the sysplex and allows the new distributed BRLM servers to start with a clean locking history.

When you are done, you have set up distributed BRLM. The system owning the file system will be the system that receives all locking requests for files in that file system.

Mounting file systems using NFS client mounts

With the z/OS NFS server, the client has remote access to z/OS UNIX files from a client workstation. Using the Network File System, the client can mount all or part of the file system and make it appear as part of its local file system. From the workstation, the client user can create, delete, read, write, and treat the host-located files as part of the workstation’s own file system.

In a similar way, the z/OS NFS client gives users remote access to files on an NFS server. Using NFS, the user can mount all or part of the remote file system and make it appear as part of the local z/OS file hierarchy. From there, the user can create, delete, read, write, and treat the remotely located files as part of the own file system.

In a sysplex, the NFS Client-NFS Server relationship is as follows: the data that becomes accessible is accessible from any place in the sysplex as long as at least one of the systems has connectivity to the NFS server.

Rule: Entries in the NFS Server Export Data Set can control which HFS directories can be mounted by client users. When specifying path names in this data set, you must specify fully qualified path names. That is, the use of symbolic links in this data set are not supported.

Preparing file systems for shutdown

File systems on the system where the shutdown was issued are immediately unmounted; data is synched to disk as a result.

For shared HFS, one of the following actions is done on the file systems that are owned by the system where the command was issued.

- Unmount if automounted or if a file system was mounted on an automounted file system.
- Move to another system if an AUTOMOVE(YES) was specified.
- Unmount for all other file systems.

File systems that are not owned by the system on which the shutdown was issued are not affected.

The shutdown should be done prior to an IPL. It replaces BPXSTOP.

On the system that you are preparing to shut down, issue the following command:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

File system availability

In the Shared HFS environment, file system availability and accessibility depends on a number of important factors. These factors can vary depending on how a file system is mounted and the capability of the file system to manage itself in a sysplex environment. After you set up the Shared HFS environment for cross-system communication (“Procedures for establishing shared HFS in a sysplex” on page 7), it will be helpful to understand how file systems availability is provided to your systems, and what kinds of actions can cause interruptions to that availability.

Minimum setup required for file system availability

Rules:

- For DASD file systems, at least one system in the Shared HFS group needs to have a physical I/O path to the volume where the file system resides and the volume varied online. Without connectivity from at least one system, the file system will not be available to any of the systems in the Shared HFS group. Connectivity from one system can provide Shared HFS accessibility to the file system for all other systems in the Shared HFS group.
- All systems need to have the physical file system (PFS) started. Accomplish this by placing the appropriate FILESYSTYPE statement in the BPXPRMxx parmlib member that is used in the configuration. Additionally, any necessary subsystems required by the PFS must be started and configured, especially if this system is to function as the file system owner. For example, the NFS Client PFS requires that the TCP/IP subsystem be started and a network connection configured.

Read-write connections for non-sysplex aware file systems: Most physical file systems (PFSes) allow only one connection for update at a time. Such file systems are called *non-sysplex aware for update*. This is directly related to the mount mode of the file system. With HFS for example, only one system can actually connect to the file system with a mode of RDWR. That system is called the *file system owner*.

The other systems that want to participate in Shared HFS sharing for the HFS file systems will also request a RDWR mount, but their access will be provided via cross-system messaging with the file system owner which has already established the read-write connection. These systems are called *file system clients*. When the file system owner becomes unavailable (for example, through system shutdown), it will be important for another system (one of the file system clients) to have the file system volume varied online so that a new owner can be established. This helps ensure maximum file system availability in the Shared HFS group.

Read-write connections for sysplex-aware file systems: Some PFSes can handle multiple concurrent connections for update. They are capable of managing the serialization of such requests. Such file systems are called *sysplex aware for update*. Most network file systems have this capability. NFS Client is one such file system type.

For a read-write mount to NFS Client, each system in the Shared HFS group will make a direct connection to NFS. The first system to make such a connection is still called the file system owner. All subsequent systems to make a direct connection are considered non-owners, rather than clients. This type of multiple direct connection for read-write access allows for maximum I/O performance by eliminating the need to send requests to the file system owner.

However, sometimes a non-owning system cannot make a direct connection to the PFS even after meeting the minimum requirements (for example, sometimes requests to NFS Client time out before they are satisfied). That system might be given a cross-system messaging connection, making it a client to the file system. While this is not the optimal mount mode for this type of file system, it does allow access to the file system.

Read-only connections for non-sysplex aware file systems: There may be some PFSes that do not support multiple concurrent connections for read-only access. These are called *non-sysplex aware for readonly*, and are handled the same as the read-write connections for non-sysplex aware file systems.

Read-only connections for sysplex-aware file systems: PFSes that support multiple concurrent connections for read-only access are called *sysplex aware for readonly*. The HFS PFS falls into this category. Such file systems are handled the same as the read-write connections for sysplex aware file systems. The read-only connections are attempted locally for each system in the Shared HFS group, but if the file system volume is not online to a system, then the system becomes a client to the file system via cross-system messaging with the owner.

Situations that can interrupt availability

Some situations may cause interruptions to file system availability on one or more systems. Following is a list of some of the most common causes. It is not meant to be an exhaustive list.

- **Loss of the file system owner.** If the file system owner leaves the Shared HFS group (through system failure, soft shutdown, VARY, XCF, OFFLINE, or some other means), an attempt may be made to establish another file system owner if requested by the AUTOMOVE specification of the mount. If a new file system owner cannot be established, the file system will become unowned. It will be unavailable until the original owner can reclaim it, or until another owner is established through subsequent automated recovery actions performed by Shared HFS.
- **PFS termination.** If a PFS terminates on one system, it can affect file system availability on other systems.

  - Prior to V1R2, if a PFS terminates on one system, all file systems of that type are unmounted across the sysplex.
  - In V1R2 and later, if a PFS terminates on one system, any file systems of that type that are owned by other systems are not affected. File systems of that type are moved to new owners whenever possible if they are owned by the system where the PFS is terminating and are automovable. These file systems remain accessible to other systems. If they cannot be moved to new owners, they are unmounted across the sysplex. It may not be possible to move a file system due to a lack of connectivity from other systems, or if the file system containing the mount point for the file system needed to be moved but could not be.
- **VARY volume,OFFLINE**. When the volume for a file system is varied offline, it will make that file system inaccessible to that system. However, if the volume is online to other systems, it may still be accessible to those systems and to other systems via cross-system messaging. This would be the case for sysplex-aware file systems for read-write or read-only access. Unlike loss of the file system owner, varying a file system volume offline will not result in any attempt by the system to try to restore accessibility to systems on which it is lost.

Tuning z/OS UNIX performance in a sysplex

The intersystem communication required to provide the additional availability and recoverability associated with z/OS UNIX shared HFS support, affects response time and throughput on R/W file systems being shared in a sysplex.

For example, assume that a user on SY1 requests a read on a file system mounted R/W and owned by SY2. Using shared HFS support, SY1 sends a message requesting this read to SY2 via an XCF messaging function:

```
SY1 ===> (XCF messaging function) ===> SY2
```

After SY2 gets this message, it issues the read on behalf of SY1, and gathers the data from the file. It then returns the data via the same route the request message took:

```
SY2 ===> (XCF messaging function) ===> SY1
```

Thus, adding z/OS UNIX to a sysplex increases XCF message traffic. To control this traffic, closely monitor the number and size of message buffers and the number of message paths within the sysplex. It is likely that you will need to define additional XCF paths and increase the number of XCF message buffers above the minimum default. For more information on signaling services in a sysplex environment, see *z/OS MVS Setting Up a Sysplex*.

You should also be aware that because of I/O operations to the CDS, every mount request requires additional system overhead. Mount time increases as a function of the number of mounts, the number of members in a sysplex, and the size of the CDS. You will need to consider the effect on your recovery time if a large number of mounts are required on any system participating in shared HFS.

DFS considerations

A file system can only be exported by the DFS server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported by DFS.

To recover from system outages, you need to weigh sysplex availability against availability to the DFS and Server Message Block (SMB) clients. When an owning

system recycles and a DFS-exported file system has been taken over by one of the other systems, DFS will not be able to automatically reexport that file system. The file system will have to be moved from its current owner back to the original DFS system, the one that has just been recycled, and then reexported.

Recommendation: For file systems that are mostly for use by DFS clients, you should consider specifying NOAUTOMOVE on the MOUNT statement. If you specify NOAUTOMOVE, the file systems will not be taken over if the system is recycled, and they will be available for automatic reexport by DFS.

Chapter 2. Changes for z/OS UNIX System Services Planning (Version 1 Release 6)

Notice for APAR OA12251

Throughout the document, all descriptions of the AUTOMOVE function as it relates to the behavior that occurs when an owning system becomes unavailable (for instance, by shutting down, crashing, or leaving the sysplex) should instead refer to the description below in "Customizing BPXPRMxx for shared HFS" on page 47.

Shared HFS in a sysplex

Overview of using shared HFS in a sysplex

This chapter describes shared HFS capability available as of OS/390 UNIX V2R9 for those who participate in a multi-system sysplex. It assumes that you already have a sysplex up. It defines what shared HFS is, introduces you to HFS data sets that exist in a sysplex, and helps you establish that environment. The topics in this chapter reflect the tasks you must do.

Although it is suggested that you exploit shared HFS support, you are not required to. If you choose not to, you will continue to share HFS data sets as you have before OS/390 UNIX V2R9. To see how your file system structure differs in OS/390 UNIX V2R9 from V2R8, see "Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or later environments" on page 3.

z/OS Parallel Sysplex Test Report describes how IBM's integration test team implemented shared HFS.

If you require a high level of security in your z/OS system and do not want superusers to have access to z/OS resources such as SYS1.PROCLIB, read the following sections:

- Comparing UNIX security and z/OS UNIX security in the chapter about setting up for daemons.
- Establishing the correct level of security for daemons in the chapter about setting up for daemons.

What does shared HFS mean?

Sysplex users can access data throughout the file hierarchy.

The best way to describe the benefit of this function is by comparing what was the file system sharing capability prior to OS/390 UNIX V2R9 with the capability that exists now. Consider a sysplex that consists of two systems, SY1 and SY2:

- Users logged onto SY1 can write to the directories on SY1. For users on SY1 to make a change to file systems mounted on SY2's **/u** directory, they would have to log onto SY2.
- The system programmer who makes configuration changes for the sysplex needs to change the entries in the **/etc** file systems for SY1 and SY2. To make the changes for both systems, the system programmer must log onto each system.

With shared HFS, all file systems that are mounted by a system participating in shared HFS are available to all participating systems. In other words, once a file system is mounted by a participating system, that file system is accessible by any

other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. Consider a OS/390 UNIX V2R9 sysplex that consists of two systems, SY1 and SY2:

- A user logged onto any system can make changes to file systems mounted on **/u**, and those changes are visible to all systems.
- The system programmer who manages maintenance for the sysplex can change entries in both **/etc** file systems from either system.

In this chapter, the term *participating group* is used to identify those systems that belong to the same SYSBPX XCF sysplex group and have followed the required installation and migration activities to participate in shared HFS. To be in the participating group, the system level must be at OS/390 UNIX V2R9 or later. Systems earlier than OS/390 UNIX V2R9 can coexist in the sysplex with systems using shared HFS support, but the earlier systems will only be able to share file systems mounted on other systems in read-only mode, and not in read/write mode.

With shared HFS, there is greater availability of data in case of system outage. There is also greater flexibility for data placement and the ability for a single BPXPRMxx member to define all the file systems in the sysplex.

To see an animated example of what shared HFS is, navigate to this URL:

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

How the end user views the HFS

This chapter describes the kinds of file systems and data sets that support the shared HFS capability in the sysplex. Figure 18 shows that, to the end users, the logical view of the HFS does not change for OS/390 UNIX V2R9. From their point of view, accessing files and directories in the system is just the same. That is true for all end users, whether they are in a sysplex or not.

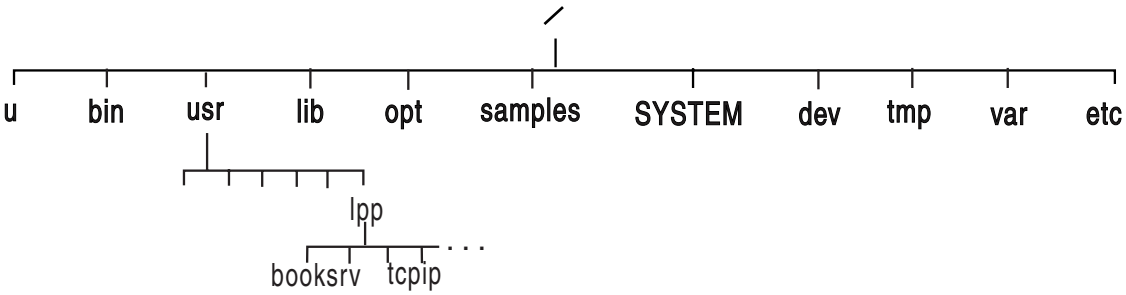

Figure 18. Logical view of shared HFS for the end user

This logical view applies to the end user only. However, system programmers need to know that the illustration of directories found in Figure 18 does not reflect the physical view of file systems. Starting in OS/390 UNIX V2R9, some of the directories are actually symbolic links, as is described in the following information.

Summary of new HFS data sets

This chapter introduces HFS data sets and terms needed to use shared HFS. Table 7 on page 37 summarizes the HFS data sets that are needed in a sysplex that has shared HFS. As you study the illustrations of file system configurations in this chapter, you can refer back to this table.

Table 7. HFS data sets that exist in a sysplex

Name	Characteristics	Purpose	How Created
Sysplex root	It contains directories and symbolic links that allow redirection of directories. Only one sysplex root HFS is allowed for all systems participating in shared HFS.	The sysplex root is used by the system to redirect addressing to other directories. It is very small and is mounted read-write. See "Procedures for establishing shared HFS in a sysplex" on page 7for a more complete description of the sysplex root HFS.	The user runs the BPXISYSR job.
System specific	It contains data specific to each system, including the **/dev**, **/tmp**, **/var**, and **/etc** directories for one system. There is one system-specific HFS data set for each system participating in the shared HFS capability.	The system-specific HFS data set is used by the system to mount system-specific data. It contains the necessary mount points for system-specific data and the symbolic links to access sysplex-wide data, and should be mounted read-write. See "Steps in creating the system-specific HFS data sets" on page 8 for a complete description of the system-specific HFS.	The user runs the BPXISYSS job on each participating system.
Version In a sysplex, *version HFS* is the new name for the root HFS.	It contains system code and binaries, including the **/bin**, **/usr**, **/lib**, **/opt**, and **/samples** directories. IBM delivers only one version root; you might define more as you add new system levels and new maintenance levels.	The version HFS has the same purpose as the root HFS in the non-sysplex world. It should be mounted read-only. See "Steps in mounting the version HFS" on page 9 for a complete description of the version HFS.	IBM supplies this HFS in the ServerPac. CBPDO users create the HFS by following steps defined in the Program Directory.

Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or later environments

The illustrations in this section show you how the file system structures that existed before OS/390 UNIX V2R9 compare with the structures in OS/390 UNIX V2R9 and later. IBM's suggestions for several releases prior to OS/390 UNIX V2R9 has been that you separate the system setup parameters from the file system parameters so that each system in the sysplex has two BPXPRMxx members: a system limits member and a file system member. In the shared HFS environment, that separation of system limit parameters from file system parameters is even more important. In the shared HFS environment, each system will continue to have a system limits BPXPMRxx member. As you will see in sections that follow, with shared HFS, you might have a file system BPXPRMxx member for each participating system or you

might replace those individual file system BPXPRMxx members with a single file system BPXPRMxx member for all participating systems.

File systems in single system pre-OS/390 UNIX V2R9 environments

The following example shows what BPXPRMxx file system parameters would look like in a single system environment (before OS/390 UNIX V2R9) with no regard to sysplex.

```
BPXPRMxx

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.ETC.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/etc')
.
.
.
```

Figure 19. BPXPRMxx for a single system before OS/390 UNIX V2R9 or later environments

The root can be mounted either read-only or read-write.

Figure 20 shows the suggested setup of the root HFS in a single system environment.

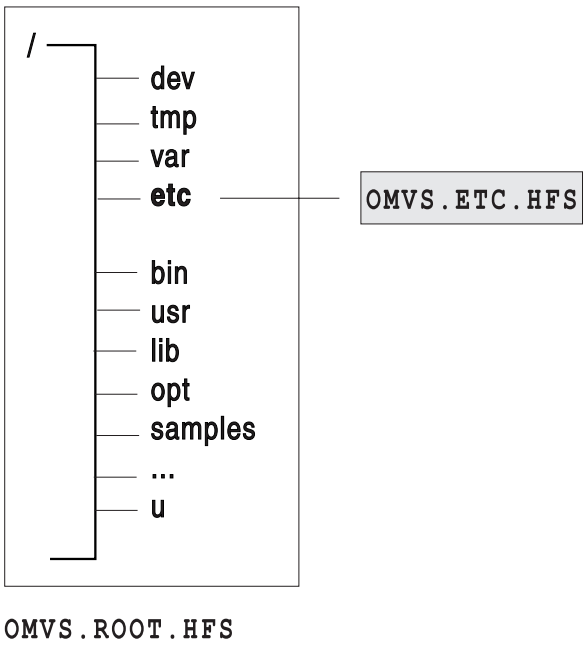

Figure 20. Single system before OS/390 UNIX V2R9

The directories in the root HFS represent “first-level” directories created by IBM. The **/etc**, **/dev**, **/var**, **/tmp**, and **/u** directories are used as mount points for other HFS data sets.

File systems in single system OS/390 UNIX V2R9 or later environments

Figure 21 shows what BPXPRMxx file system parameters would look like in an OS/390 UNIX V2R9 (or later) single system environment, and Figure 22 on page 40 shows the corresponding single system image. SYSPLEX(NO) is specified (or the default taken), and the mount mode is read-write.

The root can be mounted either read-only or read-write.

BPXPRMxx

```
FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

SYSPLEX(NO)

ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.DEV.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/dev')

MOUNT
FILESYSTEM('OMVS.TMP.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/tmp')

MOUNT
FILESYSTEM('OMVS.VAR.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/var')

MOUNT
FILESYSTEM('OMVS.ETC.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/etc')
```

Figure 21. BPXPRMxx parmlib member for single system: OS/390 UNIX V2R9

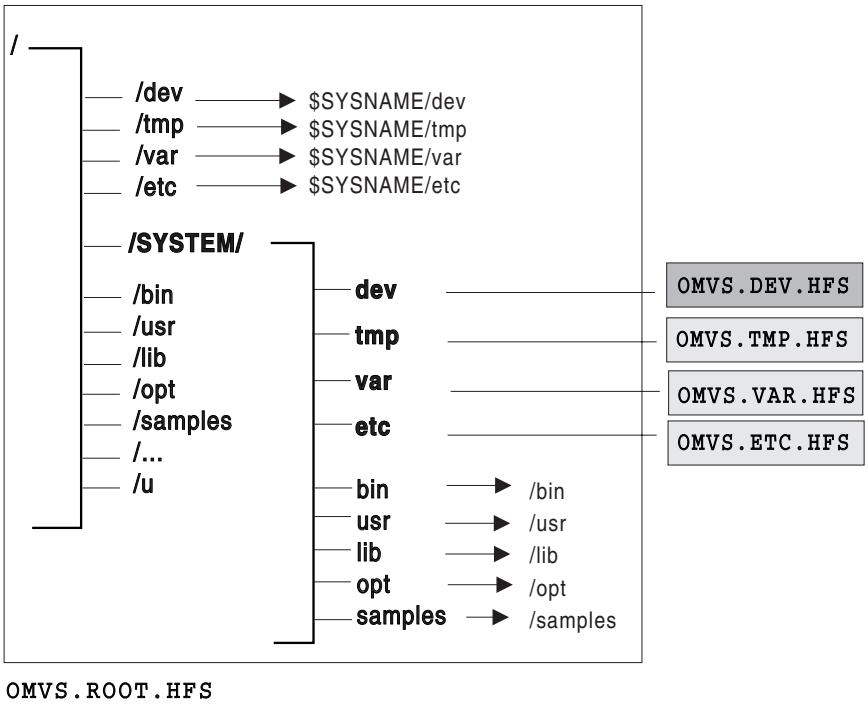

Figure 22. Single system: OS/390 UNIX V2R9

The presence of symbolic links is transparent to the user. In the illustrations used throughout this chapter, symbolic links are indicated with an arrow.

In Figure 22, the root file system contains an additional directory, **/SYSTEM**; existing directories, **/etc, /dev, /tmp** and **/var** are converted into symbolic links. These changes, however, are transparent to the user who brings up a single system environment.

If the content of the symbolic link begins with $SYSNAME and SYSPLEX is specified NO, then $SYSNAME is replaced with /SYSTEM when the symbolic link is resolved.

File systems in OS/390 UNIX V2R9 or later sysplex environments

This section describes file systems in sysplex environments (OS/390 UNIX V2R9 or later) and what you need to do to take advantage of shared HFS support, such as creating specific HFS data sets (also see Table 1 on page 3) and the OMVS couple data set, updating COUPLExx, and customizing BPXPRMxx.

Do not assume that with shared HFS, two systems can share a common HFS data set for **/etc**, **/tmp**, **/var**, and **/dev**. This is not the case. Even with shared HFS, each system must have specific HFS data sets for each of these file systems. The file systems are then mounted under the system-specific HFS (see Figure 14 on page 23). With shared HFS support, one system can access system-specific file systems on another system. (The existing security model remains the same.) For example, while logged onto SY2, you can gain read-write access to SY1's **/tmp** by specifying **/SY1/tmp/**.

You should also be aware that when SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating group must be defined for all systems participating in shared HFS. The easiest way to accomplish this is to create a single

BPXPRMxx member that contains file system information for each system participating in shared HFS. If you decide to define a BPXPRMxx member for each system, the FILESYSTYPE statements must be identical on each system. To see the differences between having one BPXPRMxx member for all participating systems and having one member for each participating system, see the two examples in “Scenario 2: Multiple systems in the sysplex – using the same release level” on page 21.

In addition, facilities required for a particular file system must be initiated on all systems in the participating group. For example, NFS requires TCP/IP; if you specify a filesystype of NFS, you must also initialize TCP/IP when you initialize NFS, even if there is no network connection.

Procedures for establishing shared HFS in a sysplex

For an animated overview of establishing shared HFS in a sysplex, navigate to this URL:

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

Creating the sysplex root HFS data set

The sysplex root is an HFS data set that is used as the sysplex-wide root. This HFS data set must be mounted read-write and designated AUTOMOVE. (See “Customizing BPXPRMxx for shared HFS” on page 13 for a description of the AUTOMOVE parameter in BPXPRMxx.). Only one sysplex root is allowed for all systems participating in shared HFS. The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB. After the job runs, the sysplex root file system structure would look like Figure 23:

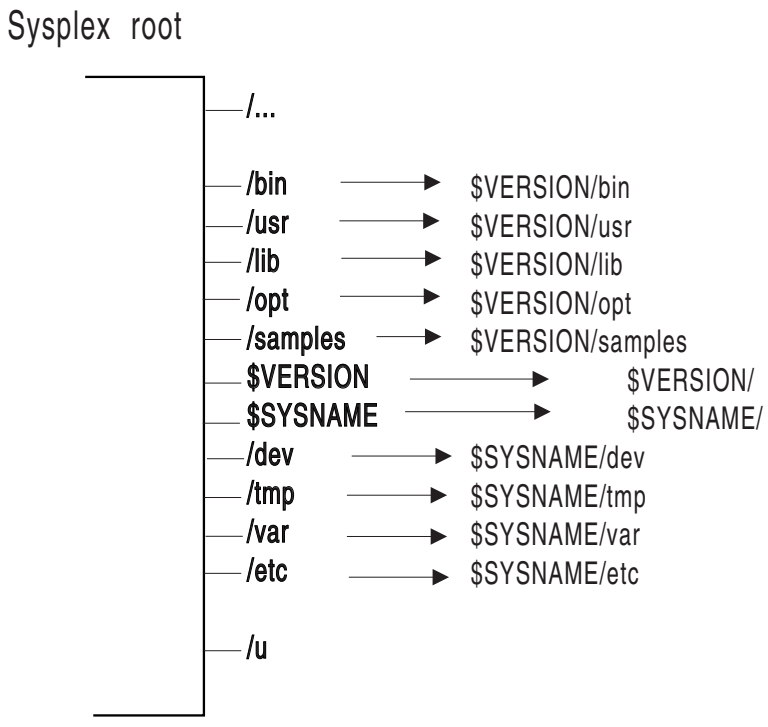

Figure 23. Sysplex root

To see an animated example of the procedures for establishing shared HFS in a sysplex, navigate to this URL:

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

No files or code reside in the sysplex root data set. It consists of directories and symbolic links only, and it is a small data set.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequent as possible.

Creating the system-specific HFS data sets

Directories in the system-specific HFS data set are used as mount points, specifically for **/etc**, **/var**, **/tmp**, and **/dev**. To create the system-specific HFS, run the BPXISYSS sample job in SYS1.SAMPLIB on each participating system (in other words, you must run the sample job separately for each system that will participate in shared HFS). After you invoke the job, the system-specific file system structure would look like Figure 24:

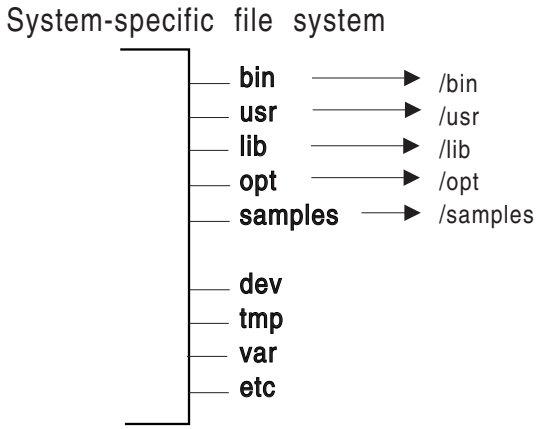

Figure 24. System HFS

The system-specific HFS data set should be mounted read-write, and should be designated AUTOMOVE(UNMOUNT). Also, **/etc**, **/var**, **/tmp**, and **/dev** should be mounted AUTOMOVE(UNMOUNT).

Guideline: The name of the system-specific data set should contain the system name as one of the qualifiers. This enables you to use the &SYSNAME symbolic (defined in IEASYMxx) in BPXPRMxx.

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at **/dev** for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Mounting the version HFS

The version HFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root HFS data set, “root HFS” has been renamed to “version HFS”.

Figure 25 on page 43 shows a version HFS.

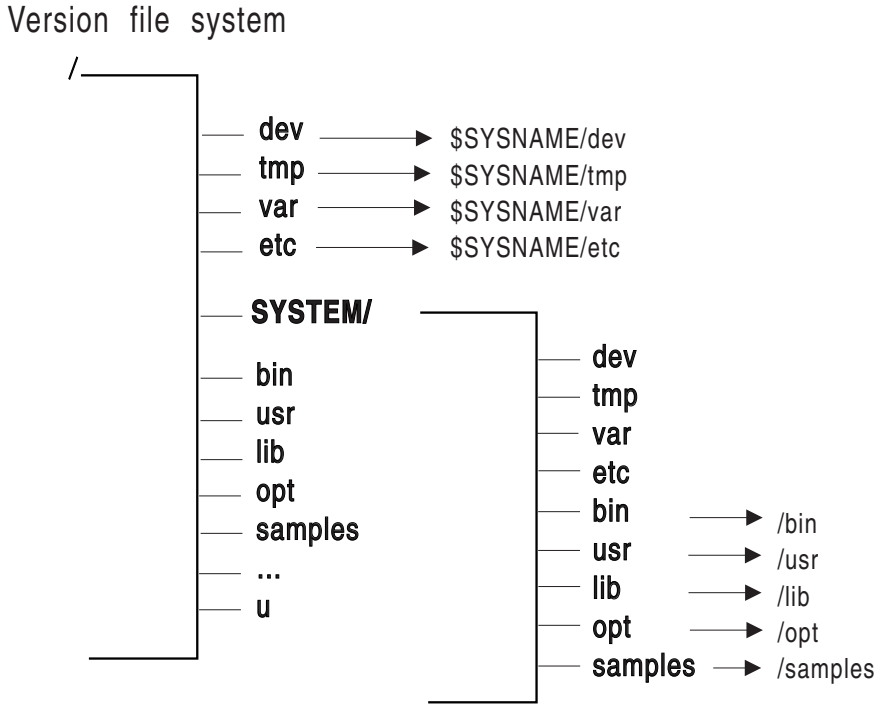

Figure 25. Version HFS

Guidelines:

1. Mount the version HFS read-only in a sysplex environment, and designate it AUTOMOVE. The mount point for the version HFS is dynamically created if the VERSION statement is used in BPXPRMxx.
2. Do not use &SYSNAME as one of the qualifiers for the version HFS data set name. In “Sysplex scenarios showing shared HFS capability” on page 18, REL9 and REL9A are used as qualifiers, which correspond to the system release levels. However, you do not necessarily have to use the same qualifiers. Other appropriate names are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer.

IBM supplies the version HFS in ServerPac. CBPDO users obtain the version HFS by following directions in the Program Directory. There is one version HFS for each set of systems participating in shared HFS and who are at the same release level (that is, using the same SYSRES volume). In other words, each version HFS denotes a different level of the system or a different service level. For example, if you have 20 systems participating in shared HFS and 10 of those systems are at OS/390 UNIX V2R9 and the other 10 are at z/OS UNIX V1R1, then you'll have one version HFS for the OS/390 V2R9 systems and one for the z/OS UNIX V1R1 systems. In essence, you will have as many version HFSes for the participating systems as you have different levels running.

Before you mount your version HFS read-only, you may have some element-specific actions. These are described in the section on post-installation actions for mounting the root file system in the chapter on managing the hierarchical file system.

Using the automove system list

When mounting file systems in the sysplex, you can specify a prioritized automove system list to indicate where the file system should or should not to moved to when ownership of a file system changes due to any of the following:

- A soft shutdown request is issued.
- Dead system takeover occurs (when a system leaves the sysplex without a prior soft shutdown).
- A PFS terminates on the owning system.
- A request to move ownership of the file system is issued.

There are different ways to specify the automove system list.

- On the MOUNT statement in BPXPRMxx, specify the AUTOMOVE keyword, including the indicator and system list.
- For the TSO MOUNT command, specify the AUTOMOVE keyword, including the indicator and system list.
- Use the **mount** shell command.
- Use the ISHELL MOUNT interface.
- Specify the MNTE_SYSLIST variable for REXX.
- Specify the indicator and system list for the automove option in the **chmount** shell command.
- Specify the indicator and system list for the automove option in the SETOMVS operator command.

Using wildcards: When you specify the automove system list, you can use wildcards in certain situations.

Example: If you have a large number of systems in your sysplex, you can specify only the systems that should have priority and use a wildcard to indicate the rest of the systems.

```
AUTOMOVE INCLUDE(s1,S2,...*)
```

At first glance, AUTOMOVE INCLUDE (*) appears to work the same way as AUTOMOVE(YES) because all of the systems will try to take over the file system. However with AUTOM0VE INCLUDE (*), if none of the systems can take over the file system, it will be unmounted. If AUTOMOVE(YES) is used, the file system will become unowned.

Restrictions:

1. All systems must be at the V1R6 level or later. Otherwise, results will be unpredictable.
2. You can use the wildcard support on all methods of mounts, including the MOUNT statement in BPXPRMxx, the TSO MOUNT command, the **mount** shell command, the ISHELL MOUNT interface, the MNTE_SYSLIST variable for REXX, C program, and assembler program.
3. The wildcard is only allowed in an INCLUDE list. It is not allowed in an EXCLUDE list.
4. The wildcard must be the last item (or the only item) in the system list.

Specifying the automount delay time: **Rule:** In a shared HFS environment, do not use the default automount delay time of 0. Instead, specify a delay time of at least 10.

Creating a couple data set (CDS)

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the primary and the other is a backup that is referred to as the alternate. In BPXISCDS, you also specify the number of mount records that are supported by the CDS.

Use of the CDS functions in the following manner:

1. The first system that enters the sysplex with SYSPLEX(YES) initializes an OMVS CDS. The CDS controls shared HFS mounts and will eventually contain information about all systems participating in shared HFS.

   This system processes its BPXPRMxx parmlib member, including all its ROOT and MOUNT statement information. It is also the designated owner of the byte range lock manager for the participating group. The MOUNT and ROOT information are logged in the CDS so that other systems that eventually join the participating group can read data about systems that are already using shared HFS.
2. Subsequent systems joining the participating group will read what is already logged in the CDS and will perform all mounts. Any new BPXPRMxx mounts are processed and logged into the CDS. Systems already in the participating group will then process the new mounts added to the CDS.

Following is the sample JCL with comments. The statements in bold contain the values that you specify based on your environment.

```
//*
//STEP10    EXEC PGM=IXCL1DSU
//STEPLIB   DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
 /* Begin definition for OMVS couple data set(1)            */
  DEFINEDS SYSPLEX(PLEX1)
   /* Name of the sysplex in which the OMVS couple data set is to be used.*/
   DSN(SYS1.OMVS.CDS01) VOLSER(3390x1)
    /* The name and volume for the OMVS couple data set.
    The utility will allocate a new data set by the name specified on the
    volume specified.*/
    MAXSYSTEMS(8)
    /* Specifies the number of systems to be supported by the OMVS CDS.
        Default =     8          */
        NOCATALOG
    /* Default is not to CATALOG */
    DATA TYPE(BPXMCDS)
    /* The type of data in the data set being created for OMVS.
    BPXMCDS is the TYPE for OMVS. */
 ITEM NAME(MOUNTS) NUMBER(500)
  /* Specifies the number of MOUNTS that can be supported by OMVS.*/
     Default =   100
     Suggested minimum =    10
     Suggested maximum = 35000  */
  ITEM NAME(AMTRULES) NUMBER(50)
   /* Specifies the number of automount rules that can be supported by OMVS.*/
      Default =    50
      Minimum =    50
      Maximum =  1000           */

   ITEM NAME(DISTBRLM) NUMBER(1)
   /*Enable conversion to a distributed BRLM. */
     1, distributed BRLM enabled,
     0, distributed BRLM is not enabled during next sysplex IPL
```

```
        not allowed for V1R6
  Default = 1  */
 /* Begin definition for OMVS couple data set(2)              */
DEFINEDS SYSPLEX(PLEX1)
/* Name of the sysplex in which the OMVS couple data set is to be used.      */
DSN(SYS1.OMVS.CDS02) VOLSER(3390x2)
/* The name and volume for the OMVS couple data set.  The utility will
   allocate a new data set by the namespecified on the volume specified.     */
 MAXSYSTEMS(8)
 /* Specifies the number of systems to be supported by the OMVS CDS.
    Default =     8           */
    NOCATALOG
 /* Default is not to CATALOG */
    DATA TYPE(BPXMCDS)
 /* The type of data in the data set being created is for OMVS.  BPXMCDS is the
    TYPE for OMVS.            */
  ITEM NAME(MOUNTS) NUMBER(500)
 /* Specifies the number of MOUNTS that can be supported by OMVS.
    Default =   100
    Suggested minimum =    10
    Suggested maximum = 35000 */
  ITEM NAME(AMTRULES) NUMBER(50)
 /* Specifies the number of automount rules that can be supported by OMVS.
    Default =     50
    Minimum =     50
    Maximum =   1000     */
  ITEM NAME(DISTBRLM) NUMBER(1)
  /*Enables conversion to a distributed BRLM.
    1, distributed BRLM enabled,
    0, distributed BRLM is not enabled; cannot be specified any longer
    Default = 1  */
```

Rule: Automount mounts must be included in the MOUNTS value. The number of automount mounts is the expected number of concurrently mounted file systems using the automount facility. For example, you may have specified 1000 file systems to be automounted, but if you expect only 50 to be used concurrently, you should factor these 50 into your MOUNTS value.

For more information about setting up a sysplex on MVS, see *z/OS MVS Setting Up a Sysplex*.

The NUMBER(*nnnn*) specified for mounts and automount rules (a generic or specific entry in an automount map file) is directly linked to function performance and the size of the CDS. If maximum values are specified, the size of the CDS will increase accordingly and the performance level for reading and updating it will decline.

Conversely, if the NUMBER values are too small, the function (for example, the number of mounts supported) would fail after the limit is reached. However, a new CDS can be formatted and switched in with larger values specified in NUMBER. To make the switch, issue the SETXCF COUPLE,PSWITCH command. The number of file systems required (factoring in an additional number to account for extra mounts), determines your minimum and maximum NUMBER value.

After the CDS is created, it must be identified to XCF for use by z/OS UNIX.

Updating COUPLExx to define the OMVS CDS to XCF: Update the active COUPLExx parmlib member to define a primary and alternate OMVS CDS to XCF. The primary and alternate CDSs should be placed on separate volumes. (The sample JCL in “Steps in creating an OMVS couple data set (CDS)” on page 11 shows the primary CDS on volume 3390x1 and the secondary CDS on 3390x2.)

Figure 26 shows the COUPLExx parmlib member; statements that define the CDS are in bold.

```
/* For all systems in any combination, up to an eightway  */
COUPLE INTERVAL(60)                    /* 1 minute */
   OPNOTIFY(60)                        /* 1 minute */
   SYSPLEX(PLEX1)                      /* SYSPLEX NAME*/
   PCOUPLE(SYS1.PCOUPLE,CPLPKP)        /* COUPLE DS */
   ACOUPLE(SYS1.ACOUPLE,CPLPKA)        /* ALTERNATE DS*/
   MAXMSG(750)
   RETRY(10)
DATA TYPE(CFRM)
   PCOUPLE(SYS1.PFUNCT.CTTEST,FDSPKP)
   ACOUPLE(SYS1.AFUNCT.CTTEST,FDSPKA)
DATA TYPE(BPXMCDS)
   PCOUPLE(SYS1.OMVS.CDS01,3390x1)
   ACOUPLE(SYS1.OMVS.CDS02,3390x2)
/* CTC DEFINITIONS: ALL SYSTEMS  */
PATHOUT DEVICE(8E0)
PATHIN DEVICE(CEF)
```

Figure 26. COUPLExx parmlib member

The MVS operator commands (DISPLAY XCF, SETXCF, DUMP, CONFIG, and VARY) enable the operator to manage the z/OS UNIX CDS.

Customizing BPXPRMxx for shared HFS

To see an animation that shows you the customization process, navigate to this URL:

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

HFS sharing enables you to use one BPXPRMxx member to define all the file systems in the sysplex. This means that each participating system has its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of system symbolics. Figure 12 on page 21 shows an example of this unified member. You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex. An example of this is Figure 13 on page 22.

The following parameters set up HFS sharing in a sysplex:

- SYSPLEX(YES) sets up HFS sharing for those who are in the SYSBPX XCF group, the group that is participating in HFS data sharing. To participate in HFS data sharing, the systems must be at the OS/390 V2R9 level or later. Those system that specify SYSPLEX(YES) make up the participating group for the sysplex.

  If SYSPLEX(YES) is specified in the BPXPRMxx member, but the system is initialized in XCF-local mode, either by specifying COUPLE SYSPLEX(LOCAL) in the COUPLExx member or by specifying PLEXCFG=XCFLOCAL in the IEASYSxx member, then the kernel will ignore the SYSPLEX(YES) value and initialize with SYSPLEX(NO). This system will not participate in shared HFS support after the initialization completes.
- VERSION('nnnn') allows multiple releases and service levels of the binaries to coexist and participate in HFS sharing. *nnnn* is a qualifier to represent a level of the version HFS. The most appropriate values for *nnnn* are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer.

A directory with the value *nnnn* specified on VERSION will be dynamically created at system initialization under the sysplex root and will be used as a mount point for the version HFS.

There is one version HFS for every instance of the VERSION parameter. More information about version HFS appears in “Steps in mounting the version HFS” on page 9.

- The SYSNAME(sysname) parameter on ROOT and MOUNT statements specifies the particular system on which a mount should be performed. *sysname* is a 1–8 alphanumeric name of the system. This system will then become the owner of the file system mounted. The owning system must be IPLed with SYSPLEX(YES).

  Tip: Specify SYSNAME(&SYSNAME.) or omit the SYSNAME parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

  The SYSNAME parameter is also used with SETOMVS when moving file systems, as demonstrated in “Moving file systems in a sysplex” on page 27.

The AUTOMOVE|NOAUTOMOVE|UNMOUNT parameters on the ROOT and MOUNT statements indicate what happens to the ownership of a file system in the following situations:

- A shutdown process is issued (soft shutdown)
- Dead system takeover occurs (a system leaves the sysplex without a prior soft shutdown)
- A PFS terminates on the owning system
- A request to move ownership of a file system is issued

AUTOMOVE is the default, specifying that ownership of the file system is automatically moved to another system.

The action taken is determined by the AUTOMOVE value and the sysplex-awareness capability of the file system.

The owner of a file system is the first system that processes the mount. This system always accesses the file system locally; that is, the system does not access the file system through a remote system. Other non-owning systems in the sysplex access the file system either locally or through the remote owning system, depending on the PFS and the mount mode. If a PFS allows a file system to be locally accessed on all systems in a sysplex for a particular mode, then the PFS is *sysplex-aware* for that mode. If a PFS requires that a file system be accessed through the remote owning system from all other systems in a sysplex for a particular mode, then the PFS is *sysplex-unaware* for that mode.

Even if a PFS is sysplex-aware for a particular mode, if a non-owning system does not have DASD connectivity, the file system is accessed remotely through the owning system. For example, HFS is sysplex-unaware for read-write mode, because all non-owning systems must access read-write file systems through the remote owning system. The non-owning systems are said to be *sysplex clients*. However, HFS is sysplex-aware for read-only mode, which means that each system can access read-only file systems locally, and do not need to contact the owning system. AUTOMOVE is intended for sysplex-unaware file systems, where non-owning systems access the file systems remotely, to allow you to specify what will happen to the ownership of file systems owned when shutdown, PFS termination, dead system takeover, or move file system occur. For sysplex-aware file systems, since they tend to be accessed locally on each system, independent of

other systems, ownership is simply moved, and file system access continues as it was, regardless of the value of AUTOMOVE.

TFS file systems do not participate in move operations, regardless of the AUTOMOVE setting. Automount-managed file systems are handled as AUTOMOVE if the file system is being used locally.

Restriction: An AUTOMOVE file system cannot be moved to a system where OMVS has been shut down or where `F BPXOINIT,SHUTDOWN=FILEOWNER` has been issued.

The following is the behavior for file systems that are mounted in a mode for which the PFS is sysplex-aware:

1. MOUNT allows AUTOMOVE(YES) or AUTOMOVE(UNMOUNT). If AUTOMOVE(NO) or a system list is specified, it is changed to AUTOMOVE(YES) and a message, BPXF234I, is put in the hardcopy log. After the MOUNT, an attempt to change AUTOMOVE to NO or specify a system list using **chmount** is rejected.
2. A move of a file system that is AUTOMOVE(NO) or has a system list to a new owner that is at V1R6 or later will change the file system to AUTOMOVE(YES) if all systems in the sysplex are at least at V1R6 or later. Again, a message will be put in the hardcopy log. This applies to any way that a file system can be moved to a new owner, including **chmount**, PFS termination, soft shutdown, and dead system recovery.
3. Remount does not change the AUTOMOVE setting, even if the new mount mode is now sysplex-aware.

Restriction: For z/OS V1R6, AUTOMOVE(NO) or system list will only be permitted for a sysplex-aware file system if the file system is mounted in a mixed-release sysplex where one or more systems are at a release below V1R6.

Table 8 shows what happens during soft shutdown for various AUTOMOVE settings for sysplex-aware and sysplex-unaware systems. A *leaf file system* refers to a file system that does not contain any file systems that are mounted on a directory within that file system. A *subtree* is the file system and all file systems that are mounted beneath that file system. Soft shutdown is done by issuing one of the following MODIFY commands:

```
F BPXOINIT,SHUTDOWN=FILESYS
F BPXOINIT,SHTUDOWN=FILEOWNER
F OMVS,SHUTDOWN
```

Table 8. Soft shutdown actions for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Unmounts the file system. The unmount will fail if it is not a leaf file system.	Unmounts the file system. The unmount will fail if it is not a leaf file system.
AUTOMOVE (no system list)	Moves the file system to any system. If the move fails, the unmount is not attempted and ownership does not change.	Moves the file system to any system. If the move fails, the unmount is not attempted and ownership does not change.
AUTOMOVE with a system list	Moves the file system to any system. The system list is ignored. If the file system cannot be moved, the unmount is not attempted and ownership does not change.	The move uses the system list. If the file system cannot be moved, the unmount is not attempted and ownership does not change.

Note: Automount-managed file systems are unmounted by a soft shutdown operation if the file system is not referenced by any other system in the sysplex. If it is referenced by another system or systems, ownership of the file system is moved. If the move fails, an unmount is not attempted and ownership does not change.

Table 9 shows what happens during dead system takeover for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems. *Dead system takeover* is the action taken by systems in a sysplex when they attempt to take over ownership of file systems that were previously owned by a system that has just left the sysplex.

Note: Takeover is always attempted for file systems that are sysplex aware. Most of these systems already have the file system locally mounted.

Table 9. Dead system takeover for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE	Takeover is attempted by all systems. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is not attempted. The file system becomes unowned.
UNMOUNT	Takeover is attempted by all systems. The subtree is unmounted if it cannot be taken over by a new owning system.	Takeover is not attempted. The subtree is unmounted.
AUTOMOVE (no system list)	Takeover is attempted. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is attempted. The file system becomes unowned if it cannot be taken over by a new owning system.
AUTOMOVE with a system list	Takeover is attempted by all systems in the sysplex. The system list is ignored. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is attempted and the INCLUDE or EXCLUDE system list is honored. The subtree is unmounted if the takeover does not happen.

Note: There is no attempt to take over automount-managed file systems if the file system is not referenced by any system that is eligible to attempt takeover. Automount-managed, unowned file systems will be unmounted.

Table 10 shows what happens during PFS termination for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems.

Table 10. PFS termination for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Moves to any system. If the move fails, the subtree is unmounted.	The subtree is unmounted.
AUTOMOVE (no system list)	Moves to any system. If the move fails, the subtree is unmounted.	Moves to any system. If the move fails, the subtree is unmounted.
AUTOMOVE with a system list	Moves to any system, ignoring the system list. If the move fails, the subtree is unmounted.	The move uses the system list. If the move fails, the subtree is unmounted.

Table 11 shows what happens when a move file system is requested to move a specific file system to any target system (wildcard is used). A move file system request can be issued with a SETOMVS operator command or a **chmount** shell command.

Table 11. Move a specific file system to any system to any system for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT or AUTOMOVE (no system list)	Move is attempted to all systems.	Move is attempted to all systems.
AUTOMOVE with a system list	Move is attempted to all systems, ignoring the system list.	Move is attempted using the system list.

Table 12 shows what happens when a move file system is requested to do a multi-file system move, moving all file systems from a system to a specific target system. A move file system request can be issued with a SETOMVS operator command or a **chmount** shell command.

Table 12. Move all file systems from a system to a specific target system to any system for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Move is attempted to the target system.	Move is not attempted.
AUTOMOVE (no system list)	Move is attempted to the target system.	Move is attempted to the target system.
AUTOMOVE with a system list	Move is attempted to the target system, ignoring the system list.	Move is attempted to the target system, ignoring the system list.

Rules:

- Define your version and sysplex root file systems as AUTOMOVE.
- Define your system-specific file systems as UNMOUNT.
- Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system underneath it as AUTOMOVE. If you do, the file system defined as AUTOMOVE will not be recovered after a system failure until that failing system has been restarted.

Guidelines:

1. To ensure that the root is always available, use the default, which is AUTOMOVE.
2. For sysplex-unaware file systems that are mostly exported by the DFS or SMB server to their remote clients, consider specifying NOAUTOMOVE on the MOUNT statement. Then the file systems will not change ownership if the system is suddenly recycled, and they will be available for automatic re-export by DFS or SMB.

   Specifying NOAUTOMOVE is suggested because a file system can only be exported by the DFS or SMB server at the system that owns the file system. A file system can only be exported by the DFS or SMB server at the system that owns the file system. Once a file system has been exported by DFS, it cannot

be moved until it has been unexported by DFS. The same holds true of file systems exported by SMB. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS or SMB clients. When an owning system recycles and a file system exported by DFS or SMB has been taken over by one of the other systems, DFS or SMB cannot automatically re-export that file system. When an owning system is recycled and an exported file system has been taken over by one of the other systems, that file system will not be automatically reexported. The file system will have to be moved from its current owner back to the original system, the one that has just been recycled, and then exported again.

For more information about VERSION, SYSPLEX, SYSNAME and AUTOMOVE|NOAUTOMVE|UNMOUNT, see *z/OS MVS Initialization and Tuning Reference*.

Sysplex scenarios showing shared HFS capability

Scenario 1: First system in the sysplex

Figure 27 on page 53 and Figure 28 on page 54 shows a z/OS UNIX file system configuration for shared HFS. Here, SYSPLEX(YES) and a value on VERSION are specified, and a directory is dynamically created on which the version HFS data set is mounted. This type of configuration requires a sysplex root and system-specific HFS.

```
BPXPRMxx for (SY1)

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')          1
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')   2
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')               3
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')          4
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')          5
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 27. BPXPRMxx parmlib setup — HFS sharing

1 This is the sysplex root HFS data set, and was created by running the BPXISYSR job. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

2 This is the system-specific HFS data set, and was created by running the BPXISYSS job. It must be mounted read-write. NOAUTOMOVE is specified because this file system is system-specific and ownership of the file system should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME.** will resolve to **/SY1** during parmlib processing. This mount point is created dynamically at system initialization.

3 This is the old root HFS (version HFS).

Guideline: It should be mounted read-only. Its mount point is created dynamically and the name of the HFS is the value specified on the VERSION statement in the BPXPRMxx member. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

4 This HFS contains the system-specific **/dev** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move

to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./dev** will resolve to **/SY1/dev** during parmlib processing.

5 This HFS contains system-specific **/tmp** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./tmp** will resolve to **/SY1/tmp** during parmlib processing.

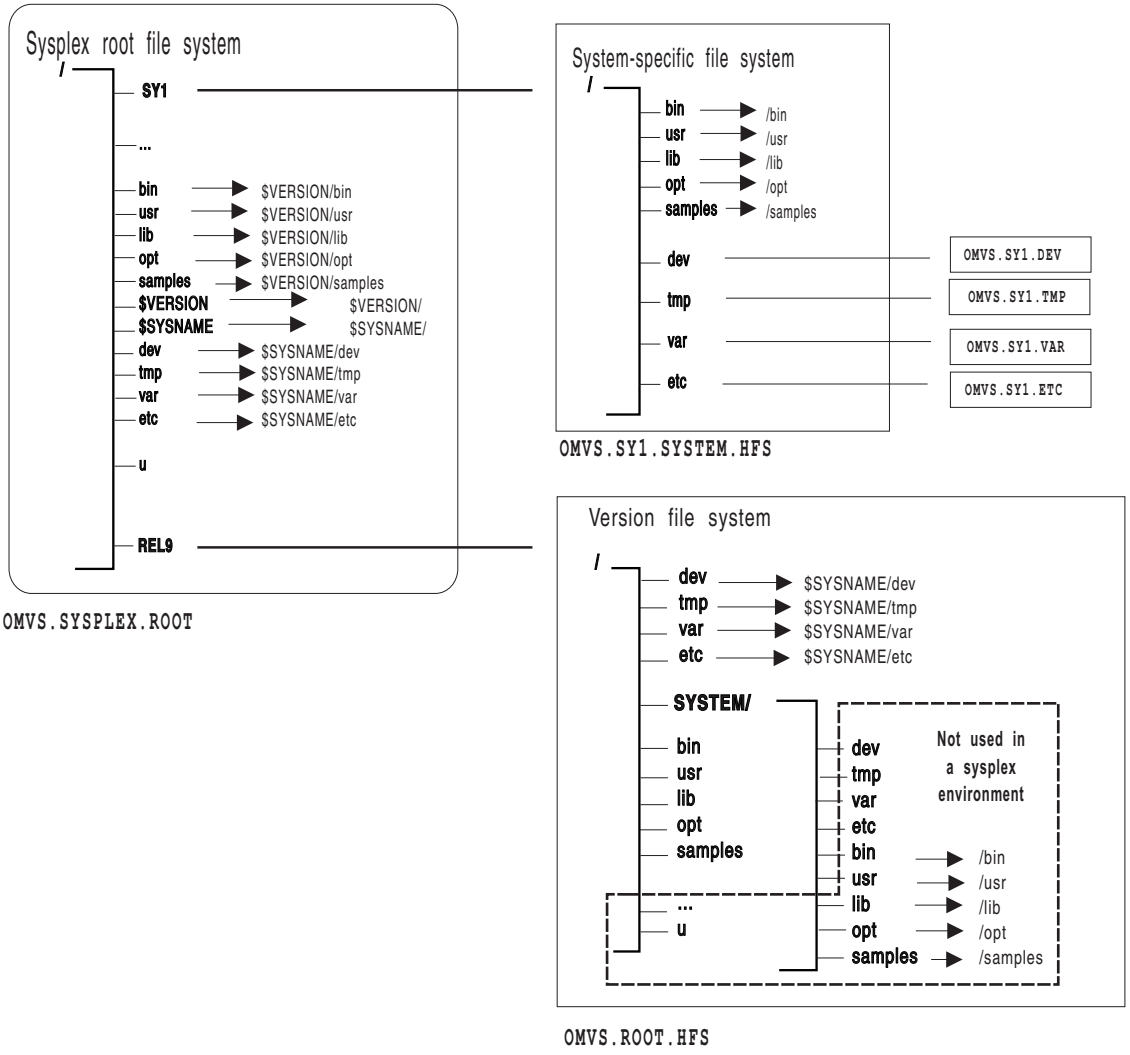

Figure 28. HFS sharing in a sysplex

If the content of the symbolic link begins with $VERSION or $SYSNAME, the symbolic link will resolve in the following manner:

- If you have specified SYSPLEX(YES) and the symbolic link for **/dev** has the contents **$SYSNAME/dev**, the symbolic link resolves to **/SY1/dev** on system SY1 and **/SY2/dev** on system SY2.
- If you have specified SYSPLEX(YES) and the content of the symbolic link begins with $VERSION, $VERSION resolves to the value *nnnn* specified on the VERSION parameter. Thus, if VERSION in parmlib is set to REL9, then $VERSION resolves to /REL9. For example, a symbolic link for **/bin**, which has the contents **$VERSION/bin**, resolves to **/REL9/bin** on a system whose $VERSION value is set to REL9.

In the above scenario, if **ls –l /bin/** is issued, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the path name **/REL9/bin**. The contents of **/REL9/bin** will now be displayed.

Scenario 2: Multiple systems in the sysplex using the same release level

Figure 31 on page 58 shows another SYSPLEX(YES) configuration. In this configuration, however, two or more systems are sharing the same version HFS (the same release level of code). Figure 29 on page 56 shows a sample BPXPRMxx for the entire sysplex (what IBM suggests) using &SYSNAME. as a symbolic name, and Figure 13 on page 22 shows a configuration where each system in the sysplex has its own BPXPRMxx. For our example, SY1 has its own BPXPRMxx and SY2 has its own BPXPRMxx.

```
One BPXPRMxx Member to Define File Systems for the Entire Sysplex

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT FILESYSTEM('OMVS.USER.HFS')
MOUNTPOINT('u') AUTOMOVE
TYPE(HFS) MODE(RDWR)


MOUNT FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 29. Sharing HFS data sets: one version HFS and one BPXPRMxx for the entire sysplex

```
BPXPRMS1 (for SY1)                      BPXPRMS2 (for SY2)

FILESYSTYPE                             FILESYSTYPE
TYPE(HFS)                               TYPE(HFS)
ENTRYPOINT(GFUAINIT)                    ENTRYPOINT(GFUAINIT)
PARM(' ')                               PARM(' ')

VERSION('REL9')                         VERSION('REL9')
SYSPLEX(YES)                            SYSPLEX(YES)

ROOT                                    ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')         FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                    TYPE(HFS) MODE(RDWR)

MOUNT                                   MOUNT
FILESYSTEM('OMVS.SY1.SYSTEM.HFS')       FILESYSTEM('OMVS.SY2.SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1')                      MOUNTPOINT('/SY2')

MOUNT FILESYSTEM('OMVS.ROOT.HFS')       MOUNT FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)                    TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                 MOUNTPOINT('/$VERSION')

MOUNT FILESYSTEM('OMVS.SY1.DEV')        MOUNT FILESYSTEM('OMVS.SY2.DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/dev')                  MOUNTPOINT('/SY2/dev')

MOUNT FILESYSTEM('OMVS.SY1.TMP')        MOUNT FILESYSTEM('OMVS.SY2.TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/tmp')                  MOUNTPOINT('/SY2/tmp')
.
.
.
```

Figure 30. Sharing HFS data sets: one version HFS and separate BPXPRMxx members for each system in the sysplex

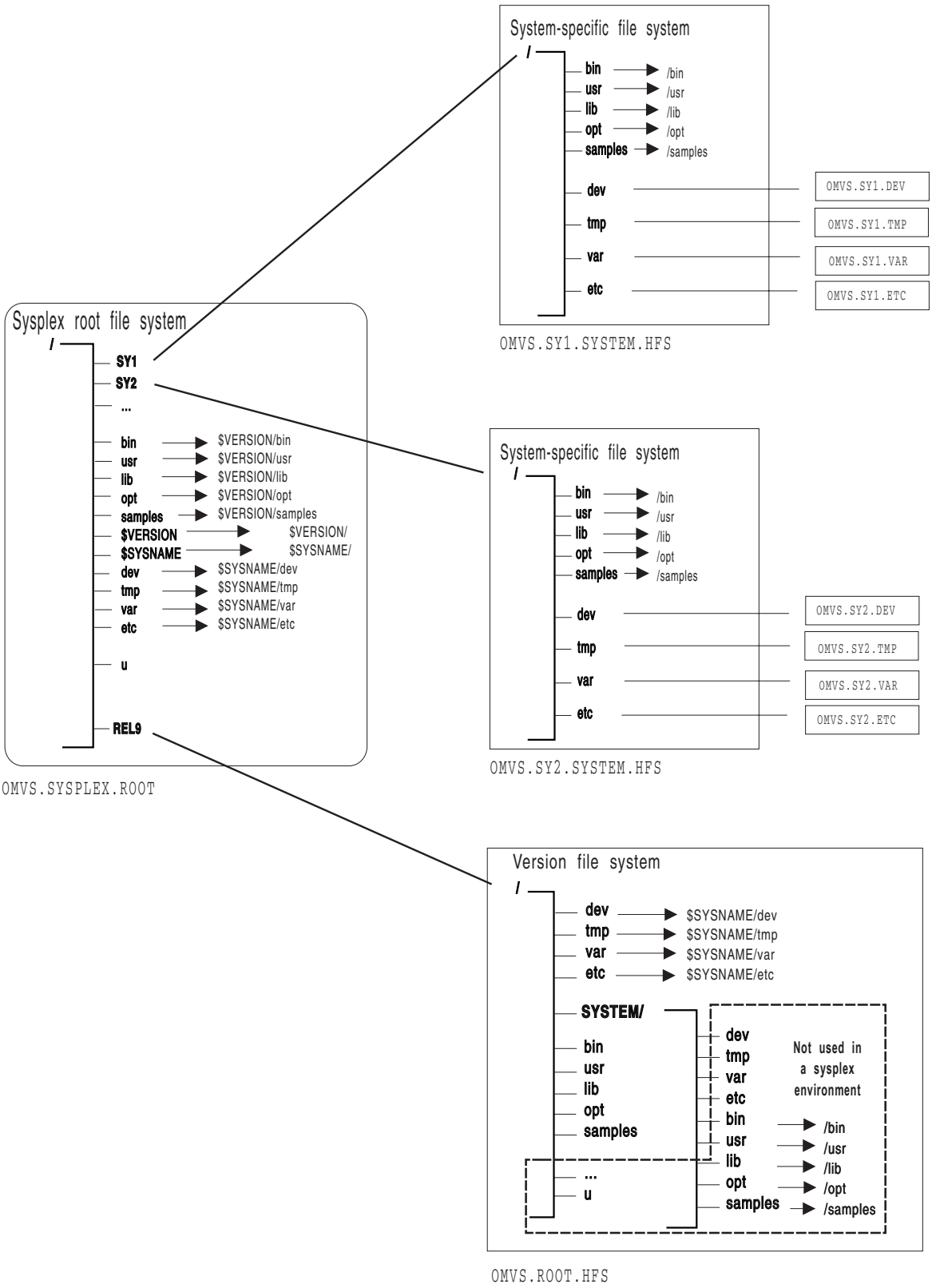

Figure 31. Sharing HFS data sets in a sysplex for OS/390 V2R9: multiple systems in a sysplex using the same release level

In this scenario, where multiple systems in the sysplex are using the same version HFS, if **ls –l /bin/** is issued from either system, the user expects to see the contents

of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the path name **/REL9/bin**. The contents of this directory will be displayed.

Scenario 3: Multiple systems in a sysplex using different release levels

If your participating group is in a sysplex that runs multiple levels of z/OS and/or OS/390, your configuration might look like the one in Figure 33 on page 60. In that configuration, each system is running a different level of z/OS and, therefore, has different version HFS data sets; SY1 has the version HFS named OMVS.SYSR9A.ROOT.HFS and SY2 has the version HFS named OMVS.SYSR9.ROOT.HFS. Figure 32 shows two BPXPRMxx parmlib members that define the file systems in this configuration. Figure 34 on page 61 shows a single BPXPRMxx parmlib member that can be used to define this same configuration; it uses &SYSR1. as the symbolic name for the two version HFS data sets.

```
BPXPRMxx (for SY1)                          BPXPRMxx (for SY2)

FILESYSTYPE                                 FILESYSTYPE
TYPE(HFS)                                   TYPE(HFS)
ENTRYPOINT(GFUAINIT)                        ENTRYPOINT(GFUAINIT)
PARM(' ')                                   PARM(' ')

VERSION('REL9A')                            VERSION('REL9')
SYSPLEX(YES)                                SYSPLEX(YES)

ROOT                                        ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')             FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                        TYPE(HFS) MODE(RDWR)


MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')     FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')                    MOUNTPOINT('/&SYSNAME.')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.SYSR9A.ROOT.HFS')          FILESYSTEM('OMVS.SYSR9.ROOT.HFS')
TYPE(HFS) MODE(READ)                        TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                     MOUNTPOINT('/$VERSION')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')            FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')                MOUNTPOINT('/&SYSNAME./dev')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')            FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')                MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 32. BPXPRMxx parmlib setup for multiple systems sharing HFS data sets and using different release levels

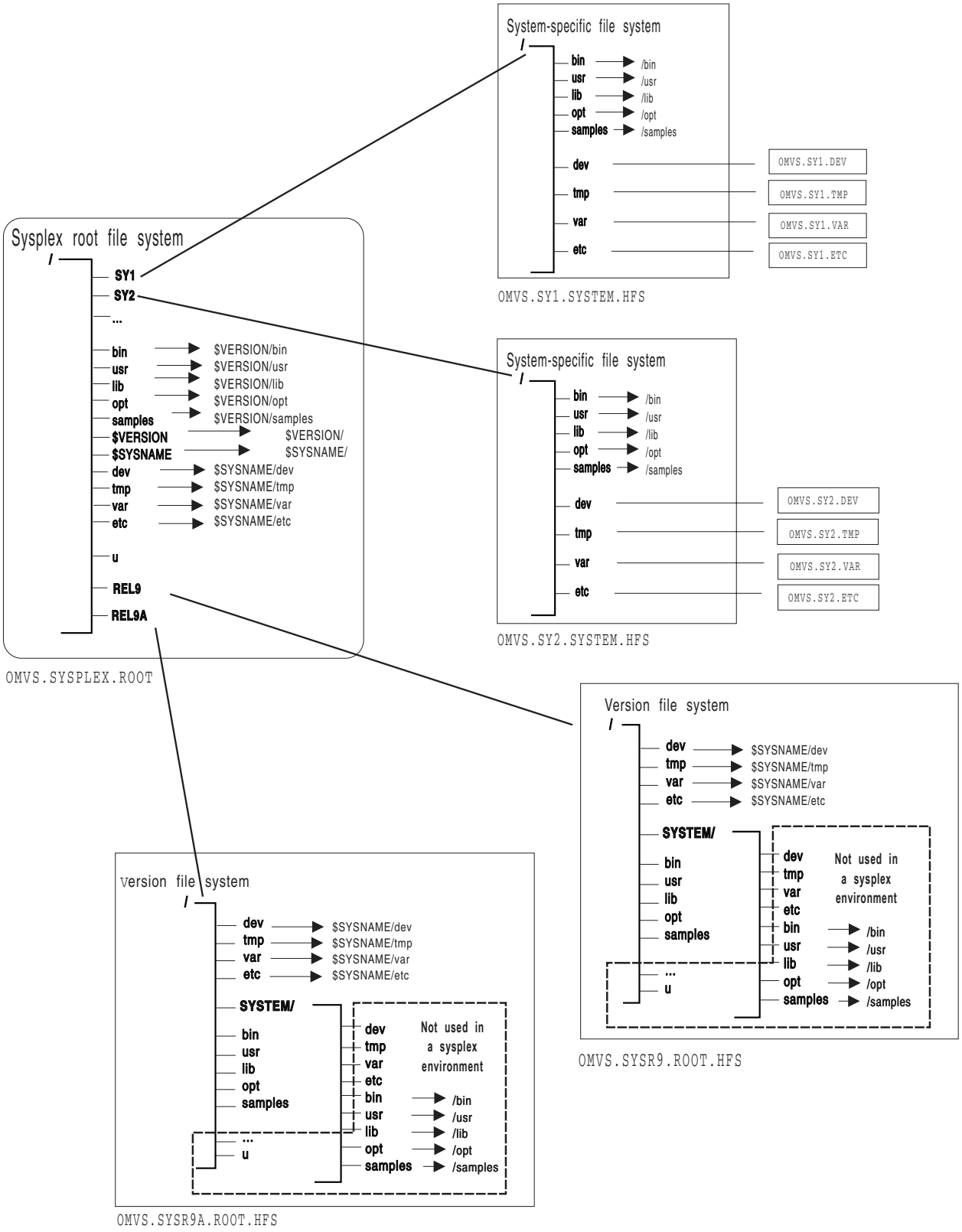

Figure 33. Sharing HFS data sets between multiple systems using different release levels

In this scenario, for example, if **ls –l /bin/** is issued on SY1, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to

/SYSR9A on SY1, which makes the path name **/SYSR9A/bin**. The contents of this directory will now be displayed. If **ls –l /bin/** is issued on SY2, the contents of **/SYSR9/bin** will display.

From SY2 you can display information on SY1 by fully qualifying the directory.

Example: To view SY1's **/bin** directory:

```
ls –l /SY1/bin/
```

```
One BPXPRMxx member to define file systems for the entire sysplex
Using different releases

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('&SYSR1.')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT FILESYSTEM('OMVS.USER.HFS')
MOUNTPOINT('u') AUTOMOVE
TYPE(HFS) MODE(RDWR)


MOUNTFILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.&SYSR1..ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 34. One BPXPRMxx parmlib member for multiple systems sharing HFS data sets and using different release levels

In order to use one BPXPRMxx parmlib file system member, we have used another system symbolic like &SYSR1. This system symbolic is used in the VERSION parameter and also as a qualifier in the version HFS data set name.

Automount policies

Rule: You must keep the automount policies consistent across all the participating systems in the sysplex. The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

Keeping your automount policy consistent on all systems

Before OS/390 UNIX V2R9, your automount policy most likely resided in the **/etc/auto.master** and **/etc/u.map files**. For those using shared HFS, each participating system has a separate **/etc** file system. In order for the automount policy to be consistent across participating systems, the same copy of the automount policy must exist in every system's **/etc/auto.master** and **/etc/u.map** files.

AUTOMOUNT is the preferred method of managing the **/u** directory. You do not need a mount statement for **/u** in the BPXPRMxx parmlib member.

For example, both SY1 and SY2 have the following files:

- **/etc/auto.master**

```
/u     /etc/u.map
```

- **/etc/u.map**

```
name       *
type       HFS
filesystem OMVS.<uc_name>.HFS
mode       rdwr
duration   60
delay      60
```

When the automount daemon initializes on SY1, it will read its local **/etc/auto.master** file to identify what directories to manage; in this case, it is **/u**. Next, the automount daemon will use the policy specified in the local **/etc/u.map** file to mount file systems with the specified naming convention under **/u**. The automount daemon on SY2 will perform similar actions. Because all mounted file systems are available to all participating systems in the sysplex, your automount policy must be consistent. This is true for the file system name specified in **/etc/u.map** and the values for other parameters in **/etc/u.map** and **/etc/auto.master**.

Moving file systems in a sysplex

You may need to change ownership of the file system for recovery or re-IPLing.

Tips:

- To check for file systems that have already been mounted, use the **df** command from the shell.
- The SETOMVS command used with the FILESYS, FILESYSTEM, mount point and SYSNAME parameters can be used to move a file system in a sysplex, or you can use the **chmount** command from the shell. However, do not move two types of file systems:
  - System-specific file systems
  - File systems that are being exported by DFS. You have to unexport them from DFS first and then move them

Examples:

1. To move ownership of the file system that contains **/u/wjs** to SY1:

```
chmount -d SY1 /u/wjs
```

2. To move ownership of the payroll file system from the current owner to SY2 using SETOMVS, issue:

   ```
   SETOMVS FILESYS,FILESYSTEM='POSIX.PAYROLL.HFS',SYSNAME=SY2
   ```

   or (assuming the mount point is over directory **/PAYROLL**)

   ```
   SETOMVS FILESYS,mountpoint='/PAYROLL',SYSNAME=SY2
   ```

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at **/dev** for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Also, opened FIFO files are automatically closed before the file system containing the FIFO is moved. They are closed because the in-storage FIFO data on the old system is not moved and is no longer accessible on new owning system.

Shared HFS implications during system failures and recovery

File system recovery in a shared HFS environment takes into consideration file system specifications such as the sysplex awareness capability, the AUTOMOVE value, and whether or not the file system is mounted read-only or read-write.

Takeover is always attempted for file systems that are sysplex-aware, regardless of the AUTOMOVE value. Most systems in the sysplex already have the file system locally mounted, so the ownership is simply moved and file system access continues as it was. Table 9 on page 50 describes the recovery actions that occur for each combination of settings.

Generally, when an owning system fails, ownership of a file system that is mounted AUTOMOVE is moved to another system and the file system remains usable. However, if a file system is mounted read-write and the owning system fails, then all file system operations for files in that file system will fail. This happens because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered. The BPX1CLO and BPX1OPN callable services are discussed in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be started again.

In some situations, even though a file system is mounted AUTOMOVE, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes unowned; if this happens, you will see message BPXF213E. This is true if the file system is mounted either read-write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that are owned by another system are still usable. However, all file operations for the unowned file system will fail until a new owner is established. The shared HFS support will continue to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for shared HFS. If a subsequent recovery attempt succeeds, the file system transitions from the unowned to the active state.

Applications using files in unowned file systems will need to close (BPX1CLO) those files and reopen (BPX1OPN) them after the file system is recovered.

Sysplex-unaware file systems that are mounted NOAUTOMOVE will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Note that since the file system still exists in the file system hierarchy, the file system mount point is still in use. File systems that are mounted below a NOAUTOMOVE file system will not be accessible via path name when the NOAUTOMOVE file system becomes available.

Do not mount AUTOMOVE file systems within NOAUTOMOVE file systems. When a NOAUTOMOVE file system becomes unowned and there are AUTOMOVE file systems mounted within it, those AUTOMOVE file systems will retain a level of availability, but only for files that are already open. When the NOAUTOMOVE file system becomes unowned, it will not be possible to perform path name lookup through it to the file systems mounted within it, which will make those file systems unavailable for new access. When ownership is restored to the unowned file system, access to the file systems mounted within it is also restored.

Managing the movement of data

File systems can be managed so as to maximize their availability when systems exit the participating group. You have more control over this when the outage is planned, but there are steps you can take to help manage the placement of data in the event of a system failure.

Recovery processing for the file systems that are owned by a failed system is managed internally by all the systems in the participating group. If you want special considerations for the placement of certain file systems, you can use the options provided by the various mount services to specify the original owner and subsequent owners for a particular file system.

"Customizing BPXPRMxx for shared HFS" on page 47 describes the behavior of the various AUTOMOVE options.

Table 13 shows the AUTOMOVE options that you can use with the MOUNT command to manage sysplex-unaware file systems. (Table 14 on page 65 covers the AUTOMOVE options for sysplex-aware file systems.)

Table 13. Automove options supported by the MOUNT command for sysplex-unaware file systems

UNMOUNT	Attempts will not be made to keep the file system active when the current owner fails. The file system will be unmounted when the owner is no longer active in the participating group, as well as all the file systems mounted within it. It is suggested for use on parmlib mounts for system-specific file systems, such as those that would be mounted at /**etc, /dev**, **/tmp** and **/var**.

Table 13. Automove options supported by the MOUNT command for sysplex-unaware file systems (continued)

NOAUTOMOVE	Attempts will not be made to keep the file system active when the current owner fails. The file system will remain in the hierarchy for possible recovery when the original owner reinitializes. Use this option on mounts for system-specific file systems if you want to have automatic recovery when the original owner rejoins the participating group. When the NOAUTOMOVE option is used, the file system becomes unowned when the owning system exits the participating group. The file system remains unowned until the last owning system restarts, or until the file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use. An unowned file system is a mounted file system that does not have an owner. Because it still exists in the file system hierarchy, it can be recovered or unmounted.
AUTOMOVE (no system list)	Recovery of the file system is to be performed when the current owner fails. This option is suggested for use on mounts of file systems that are critical to operation across all the systems in the participating group. AUTOMOVE is the default.
AUTOMOVE with a system list	AUTOMOVE(EXCLUDE|INCLUDE,*sysname1*,*sysname2*,...,*sysnameN*) specifies managed recovery of the file system if the current owner fails. • Use the EXCLUDE list to prevent recovery of a file system from transferring ownership to a particular system, or set of systems, in the participating group. When the current owner fails, recovery of the file system is performed to an owner outside the exclude list. • Use the INCLUDE list to ensure that recovery of a file system will transfer ownership only to a particular system or set of systems in the participating group. Recovery of the file system is performed in priority order only by the list of systems specified in the INCLUDE list. **Restriction:** Only use this option on mounts of file systems that are critical to operation across a subset of systems in the participating group, or when you do not want certain systems in the participating group to have ownership of the file system. If recovery processing fails to establish a new owner for the file system, the file system is unmounted, along with all the file systems mounted within it.

Table 14 shows the AUTOMOVE options that you can use with the MOUNT command to manage sysplex-aware file systems.

Table 14. Automove options supported by the MOUNT command for sysplex-aware file systems

UNMOUNT	Attempt will be made to keep the file system active when the current owner fails. The file system and all file systems that are mounted beneath it will be unmounted if the file system cannot be taken over by a new owning system.
AUTOMOVE	Recovery of the file system is to be performed when the current owner fails. This option is suggested for use on mounts of file systems that are critical to operation across all the systems in the participating group. AUTOMOVE is the default.

Note: AUTOMOVE or UNMOUNT are the only options you can use for file systems that are mounted in a mode for which they are capable of being directly mounted to the PFS on all systems (sysplex-aware). If you specify any other option on a MOUNT, it is ignored and you will see message BPXF234I.

Most of the z/OS UNIX interfaces that provide for mounting file systems (such as TSO, shell, ISHELL, and BPX2MNT) support some form of the options described in "Customizing BPXPRMxx for shared HFS" on page 47. See the associated documentation for the exact syntax.

Guideline: To ensure that the root is always available, use the default, which is AUTOMOVE.

For file systems that are mostly used by DFS or SMB clients, consider specifying NOAUTOMOVE on the MOUNT statement. Then the file systems will not change ownership if the system is suddenly recycled, and they will be available for automatic re-export by DFS or SMB. Specifying NOAUTOMOVE is suggested because a file system can only be exported by the DFS or SMB server at the system that owns the file system. Once a file system has been exported by DFS or SMB, it cannot be moved until it has been unexported from DFS or SMB. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS or SMB clients. When an owning system recycles and a DFS- or SMB-exported file system has been taken over by one of the other systems, DFS or SMB cannot automatically re-export that file system. The file system will have to be moved from its current owner back to the original DFS or SMB system, the one that has just been recycled, and then exported again.

Shared HFS implications during a planned shutdown of z/OS UNIX

These sections contain the procedures to use when shutting down z/OS UNIX.

- Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN in the Managing Operations chapter.
- Steps for shutting down z/OS UNIX usinG F BPXOINIT,SHUTDOWN=... in the Managing Operations chapter.

It is important that you understand the system actions that result when you use those procedures.

The current automove option dictates if and how the participating group recovers file system ownership from an exited system. It has no effect on the manual movement of the file system. However, when you are using the procedures for shutting down z/OS UNIX to prepare for a planned system outage, the automove option does apply. This can be explained with the following rationale:

- A system failure does not provide any means for manual intervention. The automove option provides a set of rules for automatic recovery.
- A request to move a file system manually is a deliberate action on behalf of an authorized user or administrator, and should override any rules for automatic recovery.
- Using tools to prepare for a system outage is also a deliberate action on behalf of an authorized user or administrator, but you are using these tools in an environment that can be customized to allow for additional manual intervention. You can synchronize data before the system outage, and then manage the planned outage in the same way as the unplanned outage, by making use of the automatic recovery rules that are supplied by the automove options. If you prefer some other action, you can perform manual intervention to move specific file system ownership before you use these methods for shutdown preparation.

Use F OMVS,SHUTDOWN to shut down file systems. If this is not appropriate for your installation, use the F BPXOINIT,SHUTDOWN=... procedure.

State of file systems after shutdown

File systems on the system where the shutdown was issued are immediately unmounted. As a result, data is synched to disk. For shared HFS, one of the following actions is done on the file systems that are owned by the system where the command was issued.

- Unmount if automounted or if a file system was mounted on an automounted file system.
- Move to another system if an AUTOMOVE(YES) was specified.
- Unmount for all other file systems.

File systems that are not owned by the system on which the shutdown was issued are not affected.

File system initialization

When you are preparing to bring a system back into the participating group after it has left, it is helpful to understand the coordination that occurs among the systems that are already participating in the group. You might see delays in the availability of the entering system because of activity occurring elsewhere in the sysplex. Although it is possible to bring up multiple systems simultaneously, when they reach the point of z/OS UNIX initialization, their processing is serialized so as to allow only one system at a time to initialize z/OS UNIX.

Other examples of activities occurring on other active systems that can cause the initializing system to experience delays are

- Unmounting a file system
- Changing ownership of a file system
- Recovering for systems that have left the participating group

Before it rejoins the participating group, a system processes all the file systems that are listed in the current hierarchy of the participating group. It also attempts to reclaim any unowned file systems that it previously owned when it was part of the participating group. It does not attempt to reclaim those file systems that were successfully moved or recovered to another system in the sysplex.

During initialization, any new MOUNT statements in the BPXPRMxx parmlib member are processed, which makes those file systems available for use within the participating group after they are successfully mounted.

While a system is initializing in a sysplex, critical file systems that are necessary for initialization to complete successfully might become unavailable due to a system outage. When a system is removed from the sysplex, there is a window of time during which any file systems it owned will become inaccessible to other systems. This window of time occurs while other systems are being notified of the system's exit from the sysplex and before they start the cleanup for that system.

Ideally, ownership of critical file systems will have been moved to other systems before the system exits. If that has not happened, there will be a window of time during which these critical file systems are unowned. If the initializing system requires access to these critical file systems during this window, there will likely be mount failures that prevent the initialization from completing successfully. To avoid this situation, you must make sure that any system that is being removed from the sysplex does not own any critical file systems.

Locking files in the sysplex

You can lock all or part of a file that you are accessing for read-write purposes by using the byte range lock manager (BRLM).

With V1R6, the lock manager is initialized on every system in the sysplex. This is known as distributed BRLM, and it is the only supported byte range locking method when all systems are at the V1R6 level. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system. Distributed BRLM was formerly an option on previous levels of z/OS, and central BRLM was formerly the default.

When a system failure occurs, all byte range locks are lost for files in file systems owned by that system. To maintain locking integrity for those locked files that are still open on surviving systems, z/OS UNIX prevents further locking or I/O on those files. In addition, the applications are signaled, in case they never issue locking requests or I/O. Running applications that did not issue locking requests and did not have files open are not affected.

After a failure where byte range locks are lost, z/OS UNIX provides the following information to processes that have used byte range locking:

- Access to open files for which byte range locks are held by any process will result in an I/O error. The file must be closed and reopened before use can continue.
- A signal is issued to any process which has made use of byte range locking. By default, a SIGTERM signal is issued against every such process, and an EC6 abend with reason code 0D258038 will terminate the process. If you do not want the process to be terminated, the process can use BPX1PCT (the physical file system control callable service) to specify a different signal for z/OS UNIX to use for notifying the process that the BRLM has failed. Any signal can be used for this purpose, thus allowing the user or application the ability to catch or ignore the signal and react accordingly.

  z/OS MVS System Codes describes the system completion code EC6 and its associated reason codes. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about BPX1PCT.

Using distributed BRLM

With V1R6, a file system can be moved while byte range locks are held for files in the file system. When a file system changes owners, the corresponding locking history changes BRLM servers at the same time. (This is not the case when a system failure occurs, as was discussed in “Locking files in the sysplex” on page 29.) For this reason, distributed BRLM is now the only supported method when all systems are at the V1R6 level.

If you are already running with a z/OS UNIX CDS indicating that distributed BRLM is enabled, there is no change required to activate distributed BRLM for V1R6. Likewise, if your sysplex only has systems at the V1R6 level, there is no change required, because distributed BRLM is the default. V1R6 systems ignore the z/OS UNIX CDS DISTBRLM setting.

However, if you migrate to V1R6 by running mixed levels in a sysplex, you should enable distributed BRLM before IPLing the V1R6 system because a V1R6 system may attempt to activate distributed BRLM when the central BRLM server leaves the sysplex, regardless of the z/OS UNIX CDS setting. The inconsistency between distributed BRLM being active and central BRLM being defined in the z/OS UNIX CDS can cause an EC6-BadOmvsCds abend on downlevel systems. It is a

notification-only abend indicating that the CDS should be updated. z/OS UNIX will still operate normally, and distributed BRLM will be active in the sysplex.

Before bringing the first V1R6 system into the sysplex, enable distributed BRLM by using the IXCL1DSU utility to update the BPXMCDS couple data set and then activate it. Message BPXF235I is issued when the switch from central BRLM to distributed BRLM occurs.

See “Creating a couple data set (CDS)” on page 45 for an example of the COUPLExx parmlib member. For more information about the BPXMCDS couple data set and the IXCL1DSU utility, see *z/OS MVS Setting Up a Sysplex*.With V1R6, if you run your IXCL1DSU job to create a z/OS UNIX couple data set, distributed BRLM is set up as a default.

Mounting file systems using symbolic links

You can mount different file systems at a logical mount point that resolves to a different path name on different systems.

While $VERSION/ can be used to differentiate a path based on the version level of a system and $SYSNAME/ can be used to differentiate on each system, you can use special identifiers to mount file systems using symbolic links. These are $SYSSYMR/template and $SYSSYMA/template.

Restrictions:

1. Like $VERSION/ and $SYSNAME/, the identifiers need to be at the beginning of the link name.
2. Only the first occurrence of $SYSSYMR/ or $SYSSYMA/ in the link name will be recognized as an identifier for which the remaining text requires substitutions. Any other identifiers after the first one will remain as is in the resolved linkname.
3. Text must follow a $SYSSYMR/ or $SYSSYMA/ in order for it to be recognized as a valid identifier with text containing symbols to be resolved.
4. Any system symbol in the symbolic link text that is recognized by the ASASYMBM service will be resolved. However, only static system symbols should be used in order to avoid unexpected results. These symbols are assigned a value at initialization. For information about system symbols, see *z/OS MVS Initialization and Tuning Reference*.

Tip: You can use D SYMBOLS to display the current settings of system symbols.

Examples

These examples assume that the standard MVS symbol &SYSR1. resolves to OSV315 on SY1 and resolves to OSV315B on SY2.

1. If the symbolic link is /x/y/sym1, and the symbolic link contains $SYSSYMR/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to OSV315/resdir. Because it is a relative path name (the identifier was $SYSSYMR/), the resulting path name will be /x/y/OSV315/resdir.

   Example: On a mount, passing /x/y/sym1 as the input mount point path name, the mount point would be: /x/y/OSV315/resdir on SY1.

   - If the symbol &SYSR1. resolves to OSV315B on SY2, a lookup of the same path name would result in a mount point of /x/y/OSV315B/resdir.
   - On a v_readlink syscall, passing the VnToken for the symbolic link, the output linkname would be OSV315/resdir on SY1 or OSV315B/resdir on SY2.

2. If the symbolic link is /x/y/sym1, and the symbolic link contains $SYSSYMA/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to /OSV315/resdir. Because it is an absolute path name (the identifier was $SYSSYMA/), the resulting path name will be /OSV315/resdir.

   Example: On a mount, passing /x/y/sym1 as the input mount point path name, the mount point would be /OSV315/resdir on SY1.

   - If the symbol &SYSR1. resolves to OSV315B on SY2, a lookup of the same path name from SY2 would result in a mount point of /OSV315B/resdir.
   - On a v_readlink syscall, passing the VnToken for the symbolic link, the output linkname would be /OSV315/resdir on SY1 and /OSV315B/resdir on SY2.

Mounting file systems using NFS client mounts

With the z/OS NFS server, the client has remote access to z/OS UNIX files from a client workstation. Using the Network File System, the client can mount all or part of the file system and make it appear as part of its local file system. From the workstation, the client user can create, delete, read, write, and treat the host-located files as part of the workstation's own file system.

In a similar way, the z/OS NFS client gives users remote access to files on an NFS server. Using NFS, the user can mount all or part of the remote file system and make it appear as part of the local z/OS file hierarchy. From there, the user can create, delete, read, write, and treat the remotely located files as part of their own file system.

In a sysplex, the NFS Client-NFS Server relationship is as follows: the data that becomes accessible is accessible from any place in the sysplex as long as at least one of the systems has connectivity to the NFS server.

Rule: Entries in the NFS Server Export Data Set can control which HFS directories can be mounted by client users. When specifying path names in this data set, you must specify fully qualified path names. That is, the use of symbolic links in this data set are not supported.

File system availability

In the shared HFS environment, file system availability and accessibility depend on a number of important factors. These factors can vary depending on how a file system is mounted and the capability of the file system to manage itself in a sysplex environment. After you set up the shared HFS environment for cross-system communication ("Procedures for establishing shared HFS in a sysplex" on page 41), it will be helpful to understand how file system availability is provided to your systems, and what kinds of actions can cause interruptions to that availability.

Minimum setup required for file system availability

Rules:

- For DASD file systems, at least one system in the shared HFS group needs to have a physical I/O path to the volume where the file system resides and the volume is varied online. Without connectivity from at least one system, the file system will not be available to any of the systems in the shared HFS group. Connectivity from one system can provide shared HFS accessibility to the file system for all other systems in the shared HFS group.
- All systems need to have the physical file system (PFS) started. Accomplish this by placing the appropriate FILESYSTYPE statement in the BPXPRMxx parmlib

member that is used in the configuration. Additionally, any necessary subsystems required by the PFS must be started and configured, especially if this system is to function as the file system owner. For example, the NFS Client PFS requires that the TCP/IP subsystem be started and a network connection configured.

Read-write connections for non-sysplex aware file systems: Most physical file systems (PFSes) allow only one connection for update at a time. Such file systems are called *non-sysplex aware for update*. This is directly related to the mount mode of the file system. With HFS, for example, only one system can actually connect to the file system with a mode of RDWR. That system is called the *file system owner*.

The other systems that want to participate in shared HFS sharing for the HFS file systems will also request a RDWR mount, but their access will be provided via cross-system messaging with the file system owner which has already established the read-write connection. These systems are called *file system clients*. When the file system owner becomes unavailable (for example, through system shutdown), it will be important for another system (one of the file system clients) to have the file system volume varied online so that a new owner can be established. This helps ensure maximum file system availability in the shared HFS group.

Read-write connections for sysplex-aware file systems: Some PFSes can handle multiple concurrent connections for update. They are capable of managing the serialization of such requests. Such file systems are called *sysplex aware for update*. Most network file systems have this capability. NFS Client is one such file system type.

For a read-write mount to NFS Client, each system in the shared HFS group will make a direct connection to NFS. The first system to make such a connection is still called the file system owner. All subsequent systems to make a direct connection are considered non-owners, rather than clients. This type of multiple direct connection for read-write access allows for maximum I/O performance by eliminating the need to send requests to the file system owner.

However, sometimes a non-owning system cannot make a direct connection to the PFS even after meeting the minimum requirements (for example, sometimes requests to NFS Client time out before they are satisfied). That system might be given a cross-system messaging connection, making it a client to the file system. While this is not the optimal mount mode for this type of file system, it does allow access to the file system.

Read-only connections for non-sysplex aware file systems: There may be some physical file systems that do not support multiple concurrent connections for read-only access. These are called *non-sysplex aware for readonly*, and are handled the same as the read-write connections for non-sysplex aware file systems.

Read-only connections for sysplex-aware file systems: Physical file systems that support multiple concurrent connections for read-only access are called *sysplex aware for readonly*. The HFS physical file system falls into this category. Such file systems are handled the same as the read-write connections for sysplex aware file systems. The read-only connections are attempted locally for each system in the shared HFS group, but if the file system volume is not online to a system, then the system becomes a client to the file system via cross-system messaging with the owner.

Situations that can interrupt availability

Some situations may cause interruptions to file system availability on one or more systems. Following is a list of some of the most common causes. It is not meant to be an exhaustive list.

- **Loss of the file system owner.** If the file system owner leaves the shared HFS group (through system failure, soft shutdown, VARY, XCF, OFFLINE, or some other means), an attempt may be made to establish another file system owner if requested by the AUTOMOVE specification of the mount. If a new file system owner cannot be established, the file system will become unowned. It will be unavailable until the original owner can reclaim it, or until another owner is established through subsequent automated recovery actions performed by shared HFS.
- **PFS termination.** If a PFS terminates on one system, it can affect file system availability on other systems.
  - Prior to V1R2, if a PFS terminates on one system, all file systems of that type are unmounted across the sysplex.
  - In V1R2 and later, if a PFS terminates on one system, any file systems of that type that are owned by other systems are not affected. File systems of that type are moved to new owners whenever possible if they are owned by the system where the PFS is terminating and are automovable. These file systems remain accessible to other systems. If they cannot be moved to new owners, they are unmounted across the sysplex. It may not be possible to move a file system due to a lack of connectivity from other systems, or if the file system containing the mount point for the file system needed to be moved but could not be.
- **VARY volume,OFFLINE**. When the volume for a file system is varied offline, it will make that file system inaccessible to that system. However, if the volume is online to other systems, it may still be accessible to those systems and to other systems via cross-system messaging. This would be the case for sysplex-aware file systems for read-write or read-only access. Unlike loss of the file system owner, varying a file system volume offline will not result in any attempt by the system to restore accessibility to systems on which it is lost.

Tuning z/OS UNIX performance in a sysplex

The intersystem communication required to provide the additional availability and recoverability associated with z/OS UNIX shared HFS support, affects response time and throughput on R/W file systems being shared in a sysplex.

For example, assume that a user on SY1 requests a read on a file system mounted R/W and owned by SY2. Using shared HFS support, SY1 sends a message requesting this read to SY2 via an XCF messaging function:

```
SY1 ===> (XCF messaging function) ===> SY2
```

After SY2 gets this message, it issues the read on behalf of SY1, and gathers the data from the file. It then returns the data via the same route the request message took:

```
SY2 ===> (XCF messaging function) ===> SY1
```

Thus, adding z/OS UNIX to a sysplex increases XCF message traffic. To control this traffic, closely monitor the number and size of message buffers and the number of message paths within the sysplex. It is likely that you will need to define additional XCF paths and increase the number of XCF message buffers above the minimum default. For more information about signaling services in a sysplex environment, see *z/OS MVS Setting Up a Sysplex*.

You should also be aware that because of I/O operations to the CDS, every mount request requires additional system overhead. Mount time increases as a function of the number of mounts, the number of members in a sysplex, and the size of the CDS. You will need to consider the effect on your recovery time if a large number of mounts are required on any system participating in shared HFS.

DFS considerations

A file system can only be exported by the DFS server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported by DFS.

To recover from system outages, you need to weigh sysplex availability against availability to the DFS and Server Message Block (SMB) clients. When an owning system recycles and a DFS-exported file system has been taken over by one of the other systems, DFS will not be able to automatically reexport that file system. The file system will have to be moved from its current owner back to the original DFS system, the one that has just been recycled, and then reexported.

Tip: For file systems that are mostly for use by DFS clients, you should consider specifying NOAUTOMOVE on the MOUNT statement. If you specify NOAUTOMOVE, the file systems will not be taken over if the system is recycled, and they will be available for automatic reexport by DFS.

Chapter 3. Changes for z/OS UNIX System Services Planning (Version 1 Release 7)

Notice for APAR OA12251

Throughout the document, all descriptions of the AUTOMOVE function as it relates to the behavior that occurs when an owning system becomes unavailable (for instance, by shutting down, crashing, or leaving the sysplex) should instead refer to the description below in "Customizing BPXPRMxx for a shared file system" on page 87.

Sharing file systems in a sysplex

Overview of sharing file systems in a sysplex

Terminology changes

Starting in V1R7, the following terminology changes have been made:

- The term "shared HFS" has been renamed to "shared file system".
- The term "root HFS" has been changed to "root file system".
- The term "version HFS" has been changed to "version file system".

This chapter describes the shared file system capability in a multisystem sysplex environment. It assumes that you already have completed the other setup activities for a sysplex environment. This chapter defines the shared file system concept, introduces the different file systems that exist in a sysplex, and helps you establish that environment. The topics in this chapter reflect the tasks you must do.

Although it is suggested that you exploit shared file system support when running in a sysplex environment, you are not required to do so. If you choose not to, you will continue to share file systems as you have before. To see how the file system structure has changed to support the shared file system environment, even when running on a single system, see "Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or later environments" on page 3.

z/OS Parallel Sysplex Test Report describes how IBM's integration test team implemented a shared file system.

Using IBM Health Checker for z/OS

You can install IBM Health Checker for z/OS to check the file system configuration.

What does shared file system mean?

By establishing the shared file system environment, sysplex users can access data throughout the file hierarchy from any system in the sysplex.

The best way to describe the benefit of this function is by comparing what was the file system sharing capability prior to the introduction of shared file system support with the capability that exists now. Consider a sysplex that consists of two systems, SY1 and SY2:

- Users logged onto SY1 can write to the directories on SY1. For users on SY1 to make a change to file systems mounted on SY2's **/u** directory, they would have to log onto SY2.
- The system programmer who makes configuration changes for the sysplex needs to change the entries in the **/etc** file systems for SY1 and SY2. To make the changes for both systems, the system programmer must log onto each system.

With shared file system support, all file systems that are mounted by a system participating in a shared file system are available to all participating systems. In other words, once a file system is mounted by a participating system, that file system is accessible by any other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. Consider a sysplex that consists of two systems, SY1 and SY2:

- A user logged onto any system can make changes to file systems mounted on **/u**, and those changes are visible to all systems.
- The system programmer who manages maintenance for the sysplex can change entries in both **/etc** file systems from either system.

In this chapter, the term *participating group* is used to identify those systems that belong to the same SYSBPX XCF sysplex group and have followed the required installation and migration activities to participate in a shared file system.

There is also greater availability of data in case of system outage, and a greater flexibility for data placement and the ability for a single BPXPRMxx member to define all the file systems in the sysplex.

To see an animated example of what a shared file system is, navigate to this URL:
`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/animations/ussanims.html`

How the end user views the shared file system

This chapter describes the kinds of file systems and data sets that support the shared file system capability in the sysplex. Figure 35 shows that, to the end users, the logical view of the hierarchical file system does not change. From their point of view, accessing files and directories in the system is just the same. That is true for all end users, whether they are in a sysplex or not.

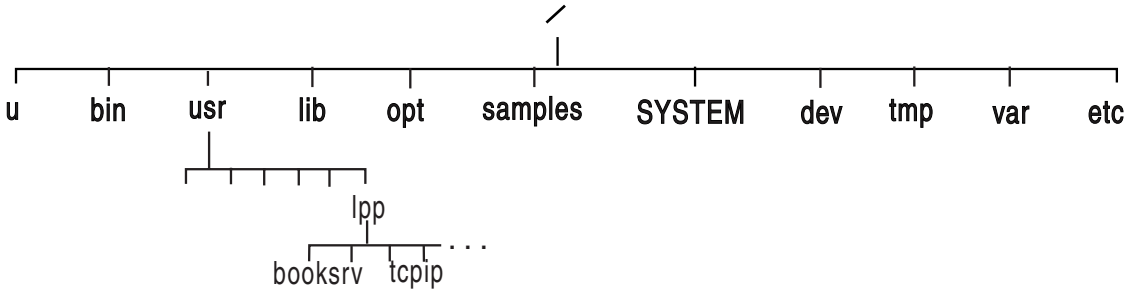

Figure 35. Logical view of a shared file system for the end user

This logical view applies to the end user only. However, system programmers need to know that the illustration of directories found in Figure 1 on page 2 does not reflect the physical view of file systems. For example, some of the directories are actually symbolic links, as is described in the following information.

Summary of various file systems in a shared environment

This chapter introduces the various file systems and terms needed to use the shared file system support. Table 15 summarizes the file systems that are needed in a sysplex environment. As you study the illustrations of file system configurations in this chapter, you can refer back to this table.

Table 15. Various file systems that exist in a sysplex

Name	Characteristics	Purpose	How created
Sysplex root	It contains directories and symbolic links that allow redirection of directories. Only one sysplex root file system is allowed for all systems participating in a shared file system.	The sysplex root is used by the system to redirect addressing to other directories. It is very small and is mounted read-write. See "Procedures for establishing shared HFS in a sysplex" on page 7 for a more complete description of the sysplex root.	For the zFS file system, the user runs the BPXISYZR job. For the HFS file system, the user runs the BPXISYSR job.
System specific	It contains data specific to each system, including the **/dev**, **/tmp**, **/var**, and **/etc** directories for one system. There is one system-specific file system for each system participating in a shared file system.	The system-specific file system is used by the system to mount system-specific data. It contains the necessary mount points for system-specific data and the symbolic links to access sysplex-wide data, and should be mounted read-write. See "Steps in creating the system-specific HFS data sets" on page 8 for a complete description of the system-specific file system.	For the zFS file system, the user runs the BPXISYSS job on each participating system. For the HFS file system, the user runs the BPXISYZS job on each participating system.
Version In a sysplex, the *version file system* is the new name for the root file system.	It contains system code and binaries, including the **/bin**, **/usr**, **/lib**, **/opt**, and **/samples** directories. IBM delivers only one version root; you might define more as you add new system levels and new maintenance levels.	The version file system has the same purpose as the root file system in the non-sysplex world. It should be mounted read-only. See "Steps in mounting the version HFS" on page 9 for a complete description of the version file system.	IBM supplies this file system in the ServerPac. CBPDO users create the file system by following steps defined in the Program Directory.

Illustrating file systems in single system and sysplex environments

The illustrations in this section show you how the file system structures have changed since the introduction of the shared file system support. These illustrations build upon IBM's long-standing suggestions that you separate the system setup parameters from the file system parameters so that each system in the sysplex has two BPXPRMxx members: a system limits member and a file system member.

In the shared file system environment, that separation of system limit parameters from file system parameters is even more important. Each system will continue to have a system limits BPXPRxx member. As you will see in sections that follow, with shared file system support, you can have a file system BPXPRMxx member for each participating system or you can replace those individual file system BPXPRMxx members with a single file system BPXPRMxx member for all participating systems.

File systems in single system environments

Figure 36 shows what BPXPRMxx file system parameters would look like in a single system environment, and Figure 37 on page 79 shows the corresponding single system image. SYSPLEX(NO) is specified (or the default taken), and the mount mode is read-write.

The root can be mounted either read-only or read-write.

BPXPRMxx

```
FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

SYSPLEX(NO)

ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.DEV.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/dev')

MOUNT
FILESYSTEM('OMVS.TMP.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/tmp')

MOUNT
FILESYSTEM('OMVS.VAR.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/var')

MOUNT
FILESYSTEM('OMVS.ETC.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/etc')
```

Figure 36. BPXPRMxx parmlib member for single system

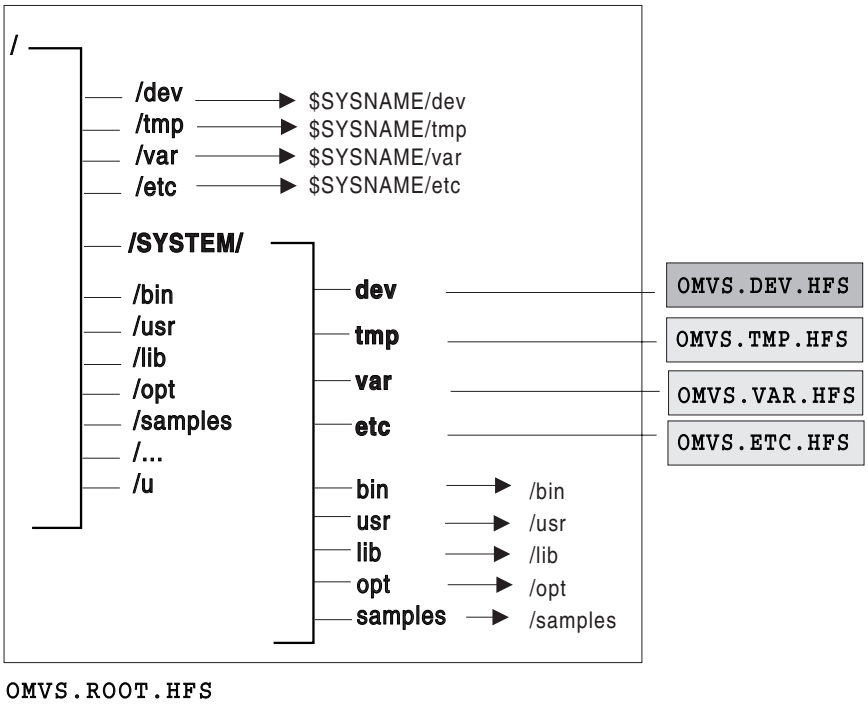

Figure 37. Single system illustration

The presence of symbolic links is transparent to the user. In the illustrations used throughout this chapter, symbolic links are indicated with an arrow.

In Figure 5 on page 6, the root file system contains an additional directory, **/SYSTEM**; existing directories, **/etc, /dev, /tmp** and **/var** are converted into symbolic links. These changes, however, are transparent to the user who brings up a single system environment.

If the content of the symbolic link begins with $SYSNAME and SYSPLEX is specified NO, then $SYSNAME is replaced with /SYSTEM when the symbolic link is resolved.

File systems in sysplex environments

This section describes file systems in sysplex environments and what you need to do to take advantage of shared file system support, such as creating specific file systems (also see Table 1 on page 3) and the OMVS couple data set, updating COUPLExx, and customizing BPXPRMxx.

Do not assume that with shared file systems, two systems can share a common HFS data set for **/etc**, **/tmp**, **/var**, and **/dev**. This is not the case. Even with shared file systems, each system must have specific file systems for each of these mount points. The file systems are then mounted under the system-specific file system (see Figure 14 on page 23). With shared file system support, one system can access system-specific file systems on another system. (The existing security model remains the same.) For example, while logged onto SY2, you can gain read-write access to SY1's **/tmp** by specifying **/SY1/tmp/**.

You should also be aware that when SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating group must be defined for all systems participating in a shared file system. The easiest way to accomplish this is to create

a single BPXPRMxx member that contains file system information for each system participating in a shared file system. If you decide to define a BPXPRMxx member for each system, the FILESYSTYPE statements must be identical on each system. To see the differences between having one BPXPRMxx member for all participating systems and having one member for each participating system, see the two examples in “Scenario 2: Multiple systems in the sysplex – using the same release level” on page 21.

In addition, facilities required for a particular file system must be initiated on all systems in the participating group. For example, NFS requires TCP/IP; if you specify a FILESYSTYPE of NFS, you must also initialize TCP/IP when you initialize NFS, even if there is no network connection.

Procedures for establishing a shared file system in a sysplex

For an animated overview of establishing a shared file system in a sysplex, navigate to this URL:

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

Creating the sysplex root file system

The sysplex root is a file system that is used as the sysplex-wide root. This file system must be mounted read-write and designated AUTOMOVE. (See “Customizing BPXPRMxx for shared HFS” on page 13 for a description of the AUTOMOVE parameter in BPXPRMxx.). Only one sysplex root is allowed for all systems participating in a shared file system.

There are two ways the sysplex root is created.

- For the zFS file system, it is created by invoking the BPXISYZR job in SYS1.SAMPLIB.
- For the HFS file system, it is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB.

After the job runs, the sysplex root file system structure would look like Figure 38 on page 81:

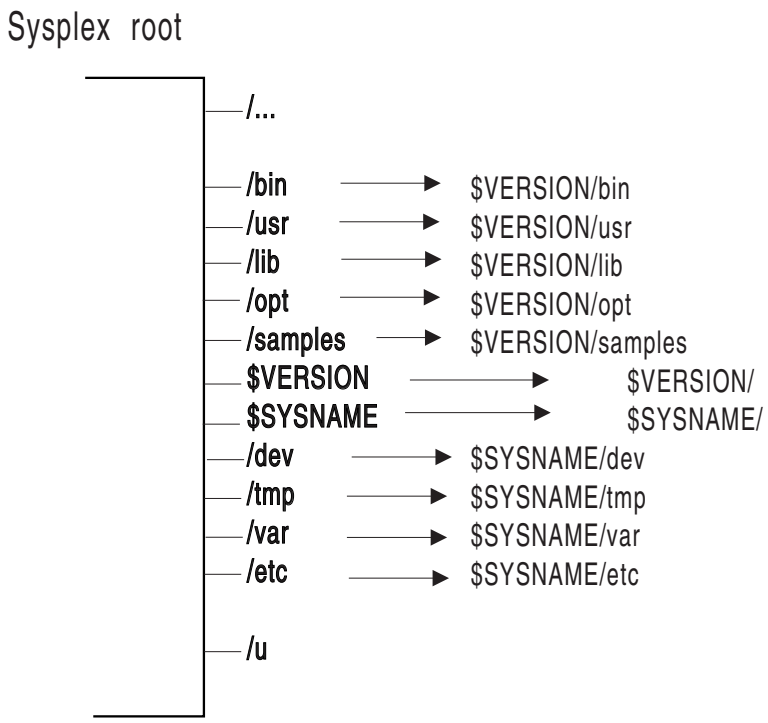

Figure 38. Sysplex root

To see an animated example of the procedures for establishing a shared file system in a sysplex, navigate to this URL:

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

No files or code reside in the sysplex root file system. It consists of directories and symbolic links only, and hence the size of the data set representing the sysplex root is very small.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories.

Creating the system-specific file system

Directories in the system-specific file system are used as mount points, specifically for **/etc**, **/var**, **/tmp**, and **/dev**.

Rule: To create the system-specific file system, you need to run the appropriate sample job in SYS1.SAMPLIB on each participating system. In other words, you must run the sample job separately for each system that will participate in a file system.

- For the zFS file system, run the BPXISYZS sample job in SYS1.SAMPLIB.
- For the HFS file system, run the BPXISYSS sample job in SYS1.SAMPLIB.

After you invoke the job, the system-specific file system structure would look like Figure 39 on page 82:

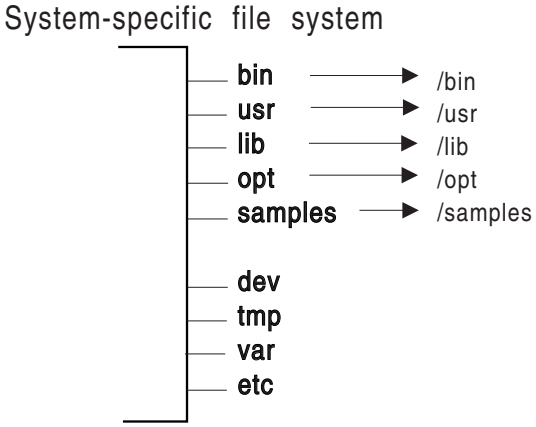

Figure 39. System-specific file system

The system-specific file system should be mounted read-write, and should be designated AUTOMOVE(UNMOUNT). Also, **/etc**, **/var**, **/tmp**, and **/dev** should be mounted AUTOMOVE(UNMOUNT).

Guideline: The name of the system-specific data set should contain the system name as one of the qualifiers. This enables you to use the &SYSNAME symbolic (defined in IEASYMxx) in BPXPRMxx.

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at **/dev** for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Mounting the version file system

The version file system is the IBM-supplied root file system. To avoid confusion with the sysplex root file system, “root file system” has been renamed to “version file system”.

Figure 40 on page 83 shows a version file system.

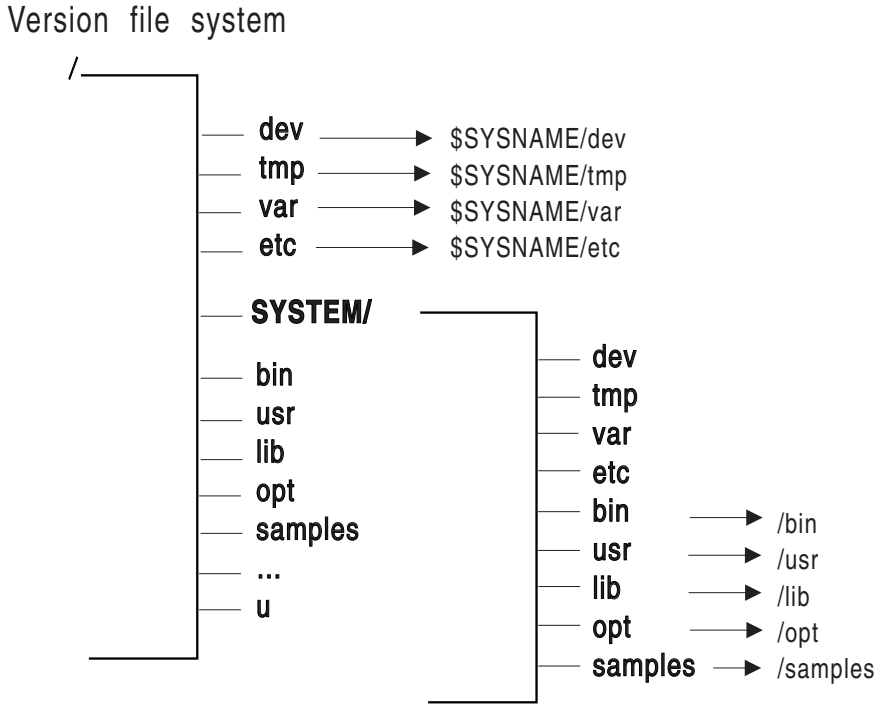

Figure 40. Version file system

Guidelines:

1. Mount the version file system read-only in a sysplex environment, and designate it AUTOMOVE. The mount point for the version file system is dynamically created if the VERSION statement is used in BPXPRMxx.
2. Do not use &SYSNAME as one of the qualifiers for the version file system data set name. In “Sysplex scenarios showing shared HFS capability” on page 18, REL9 and REL9A are used as qualifiers, which correspond to the system release levels. However, you do not necessarily have to use the same qualifiers. Other appropriate names are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer.

IBM supplies the version file system in ServerPac. CBPDO users obtain the version file system by following directions in the Program Directory. There is one version file system for each set of systems participating in a shared file system and who are at the same release level (that is, using the same SYSRES volume). In other words, each version file system denotes a different level of the system or a different service level. For example, if you have 20 systems participating in a shared file system, and 10 of those systems are at Release 9 and the other 10 are at Release 9A, then you'll have one version file system for the Release 9 systems and one for the Release 9A systems. In essence, you will have as many version file systems for the participating systems as you have different levels running.

Before you mount your version HFS read-only, you may have some element-specific actions. These are described in the section on post-installation actions for mounting the root file system in the chapter on managing the hierarchical file system.

Using the automove system list

When mounting file systems in the sysplex, you can specify a prioritized automove system list to indicate where the file system should or should not to moved to when ownership of a file system changes due to any of the following:

- A soft shutdown request is issued.
- Dead system takeover occurs (when a system leaves the sysplex without a prior soft shutdown).
- A PFS terminates on the owning system.
- A request to move ownership of the file system is issued.

There are different ways to specify the automove system list.

- On the MOUNT statement in BPXPRMxx, specify the AUTOMOVE keyword, including the indicator and system list.
- For the TSO MOUNT command, specify the AUTOMOVE keyword, including the indicator and system list.
- Use the **mount** shell command.
- Use the ISHELL MOUNT interface.
- Specify the MNTE_SYSLIST variable for REXX.
- Specify the indicator and system list for the automove option in the **chmount** shell command.
- Specify the indicator and system list for the automove option in the SETOMVS operator command.

Using wildcards: When you specify the automove system list, you can use wildcards in certain situations.

Example: If you have a large number of systems in your sysplex, you can specify only the systems that should have priority and use a wildcard to indicate the rest of the systems.

```
AUTOMOVE INCLUDE(s1,S2,...*)
```

At first glance, AUTOMOVE INCLUDE (*) appears to work the same way as AUTOMOVE(YES) because all of the systems will try to take over the file system. However with AUTOM0VE INCLUDE (*), if none of the systems can take over the file system, it will be unmounted. If AUTOMOVE(YES) is used, the file system will become unowned.

Restrictions:

1. You can use the wildcard support on all methods of mounts, including the MOUNT statement in BPXPRMxx, the TSO MOUNT command, the **mount** shell command, the ISHELL MOUNT interface, the MNTE_SYSLIST variable for REXX, C program, and assembler program.
2. The wildcard is only allowed in an INCLUDE list. It is not allowed in an EXCLUDE list.
3. The wildcard must be the last item (or the only item) in the system list.

Specifying the automount delay time: **Rule:** In a shared file system, do not use the default automount delay time of 0. Instead, specify a delay time of at least 10.

Creating a couple data set (CDS)

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To

allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the primary and the other is a backup that is referred to as the alternate. In BPXISCDS, you also specify the number of mount records that are supported by the CDS.

Use of the CDS functions in the following manner:

1. The first system that enters the sysplex with SYSPLEX(YES) initializes an OMVS CDS. The CDS controls shared file system mounts and will eventually contain information about all systems participating in shared file system.

   This system processes its BPXPRMxx parmlib member, including all its ROOT and MOUNT statement information. It is also the designated owner of the byte range lock manager for the participating group. The MOUNT and ROOT information are logged in the CDS so that other systems that eventually join the participating group can read data about systems that are already using shared file system.
2. Subsequent systems joining the participating group will read what is already logged in the CDS and will perform all mounts. Any new BPXPRMxx mounts are processed and logged into the CDS. Systems already in the participating group will then process the new mounts added to the CDS.

Following is the sample JCL with comments. The statements in bold contain the values that you specify based on your environment.

```
//*
//STEP10    EXEC PGM=IXCL1DSU
//STEPLIB   DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
 /* Begin definition for OMVS couple data set(1)            */
  DEFINEDS SYSPLEX(PLEX1)
   /* Name of the sysplex in which the OMVS couple data set is to be used.*/
   DSN(SYS1.OMVS.CDS01) VOLSER(3390x1)
    /* The name and volume for the OMVS couple data set.
    The utility will allocate a new data set by the name specified on the
    volume specified.*/
    MAXSYSTEMS(8)
    /* Specifies the number of systems to be supported by the OMVS CDS.
        Default =     8          */
        NOCATALOG
    /* Default is not to CATALOG */
    DATA TYPE(BPXMCDS)
    /* The type of data in the data set being created for OMVS.
    BPXMCDS is the TYPE for OMVS. */
 ITEM NAME(MOUNTS) NUMBER(500)
  /* Specifies the number of MOUNTS that can be supported by OMVS.*/
     Default =   100
     Suggested minimum =    10
     Suggested maximum = 35000  */
  ITEM NAME(AMTRULES) NUMBER(50)
   /* Specifies the number of automount rules that can be supported by OMVS.*/
      Default =    50
      Minimum =    50
      Maximum =  1000           */

   ITEM NAME(DISTBRLM) NUMBER(1)
   /*Enable conversion to a distributed BRLM. */
     1, distributed BRLM enabled,
     0, distributed BRLM is not enabled during next sysplex IPL
            not allowed for z/OS V1R6 or higher
     Default = 1  */
    /* Begin definition for OMVS couple data set(2)              */
   DEFINEDS SYSPLEX(PLEX1)
```

```
/* Name of the sysplex in which the OMVS couple data set is to be used.      */
DSN(SYS1.OMVS.CDS02) VOLSER(3390x2)
/* The name and volume for the OMVS couple data set.  The utility will
   allocate a new data set by the namespecified on the volume specified.     */
 MAXSYSTEMS(8)
 /* Specifies the number of systems to be supported by the OMVS CDS.
    Default =     8           */
    NOCATALOG
 /* Default is not to CATALOG */
    DATA TYPE(BPXMCDS)
 /* The type of data in the data set being created is for OMVS.  BPXMCDS is the
    TYPE for OMVS.            */
  ITEM NAME(MOUNTS) NUMBER(500)
 /* Specifies the number of MOUNTS that can be supported by OMVS.
    Default =   100
    Suggested minimum =    10
    Suggested maximum = 35000 */
  ITEM NAME(AMTRULES) NUMBER(50)
 /* Specifies the number of automount rules that can be supported by OMVS.
    Default =     50
    Minimum =     50
    Maximum =   1000     */
  ITEM NAME(DISTBRLM) NUMBER(1)
  /*Enables conversion to a distributed BRLM.
    1, distributed BRLM enabled,
    0, distributed BRLM is not enabled; cannot be specified any longer
    Default = 1  */
```

Rule: Automount mounts must be included in the MOUNTS value. The number of automount mounts is the expected number of concurrently mounted file systems using the automount facility. For example, you may have specified 1000 file systems to be automounted, but if you expect only 50 to be used concurrently, you should factor these 50 into your MOUNTS value.

For more information about setting up a sysplex on MVS, see *z/OS MVS Setting Up a Sysplex*.

The NUMBER(*nnnn*) specified for mounts and automount rules (a generic or specific entry in an automount map file) is directly linked to function performance and the size of the CDS. If maximum values are specified, the size of the CDS will increase accordingly and the performance level for reading and updating it will decline.

Conversely, if the NUMBER values are too small, the function (for example, the number of mounts supported) would fail after the limit is reached. However, a new CDS can be formatted and switched in with larger values specified in NUMBER. To make the switch, issue the SETXCF COUPLE,PSWITCH command. The number of file systems required (factoring in an additional number to account for extra mounts), determines your minimum and maximum NUMBER value.

After the CDS is created, it must be identified to XCF for use by z/OS UNIX.

Updating COUPLExx to define the OMVS CDS to XCF: Update the active COUPLExx parmlib member to define a primary and alternate OMVS CDS to XCF. The primary and alternate CDSs should be placed on separate volumes. (The sample JCL in “Steps in creating an OMVS couple data set (CDS)” on page 11 shows the primary CDS on volume 3390x1 and the secondary CDS on 3390x2.)

Figure 41 on page 87 shows the COUPLExx parmlib member; statements that define the CDS are in bold.

```
/* For all systems in any combination, up to an eightway  */
COUPLE INTERVAL(60)                    /* 1 minute */
   OPNOTIFY(60)                        /* 1 minute */
   SYSPLEX(PLEX1)                      /* SYSPLEX NAME*/
   PCOUPLE(SYS1.PCOUPLE,CPLPKP)        /* COUPLE DS */
   ACOUPLE(SYS1.ACOUPLE,CPLPKA)        /* ALTERNATE DS*/
   MAXMSG(750)
   RETRY(10)
DATA TYPE(CFRM)
   PCOUPLE(SYS1.PFUNCT.CTTEST,FDSPKP)
   ACOUPLE(SYS1.AFUNCT.CTTEST,FDSPKA)
DATA TYPE(BPXMCDS)
   PCOUPLE(SYS1.OMVS.CDS01,3390x1)
   ACOUPLE(SYS1.OMVS.CDS02,3390x2)
/* CTC DEFINITIONS: ALL SYSTEMS  */
PATHOUT DEVICE(8E0)
PATHIN DEVICE(CEF)
```

Figure 41. COUPLExx parmlib member

The MVS operator commands (DISPLAY XCF, SETXCF, DUMP, CONFIG, and VARY) enable the operator to manage the z/OS UNIX CDS.

Customizing BPXPRMxx for a shared file system

To see an animation that shows you the customization process, navigate to this URL:

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

You can use one BPXPRMxx member to define all the file systems in the sysplex. Each participating system has its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of system symbolics. Figure 12 on page 21 shows an example of this unified member. You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex. An example of this is Figure 13 on page 22.

To set up a shared file system in a sysplex, use the following parameters:

- SYSPLEX(YES) sets up a shared file system for those who are in the SYSBPX XCF group, the group that is participating in a shared file system. To participate, the systems must be at the OS/390 V2R9 level or later. Those system that specify SYSPLEX(YES) make up the participating group for the sysplex.

  If SYSPLEX(YES) is specified in the BPXPRMxx member, but the system is initialized in XCF-local mode, either by specifying COUPLE SYSPLEX(LOCAL) in the COUPLExx member or by specifying PLEXCFG=XCFLOCAL in the IEASYSxx member, then the kernel will ignore the SYSPLEX(YES) value and initialize with SYSPLEX(NO). This system will not participate in a shared file system support after the initialization completes.
- VERSION('*nnnn*') allows multiple releases and service levels of the binaries to coexist and participate in a shared file system. *nnnn* is a qualifier to represent a level of the version file system. The most appropriate values for *nnnn* are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer. A directory with the value *nnnn* specified on VERSION will be dynamically created at system initialization under the sysplex root and will be used as a mount point for the version file system.

  There is one version file system for every instance of the VERSION parameter. More information about version file system appears in “Steps in mounting the version HFS” on page 9.

- The SYSNAME(*sysname*) parameter on ROOT and MOUNT statements specifies the particular system on which a mount should be performed. *sysname* is a 1-to-8 alphanumeric name of the system. This system will then become the owner of the file system mounted. The owning system must be IPLed with SYSPLEX(YES).

  Tip: Specify SYSNAME(&SYSNAME.) or omit the SYSNAME parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

  The SYSNAME parameter is also used with SETOMVS when moving file systems, as demonstrated in "Moving file systems in a sysplex" on page 27.

The AUTOMOVE|NOAUTOMOVE|UNMOUNT parameters on the ROOT and MOUNT statements indicate what happens to the ownership of a file system in the following situations:

- A shutdown process is issued (soft shutdown)
- Dead system takeover occurs (a system leaves the sysplex without a prior soft shutdown)
- A PFS terminates on the owning system
- A request to move ownership of a file system is issued

The action taken is determined by the AUTOMOVE value and the sysplex-awareness capability of the file system. AUTOMOVE is the default, specifying that ownership of the file system is automatically moved to another system.

The owner of a file system is the first system that processes the mount. This system always accesses the file system locally; that is, the system does not access the file system through a remote system. Other non-owning systems in the sysplex access the file system either locally or through the remote owning system, depending on the PFS and the mount mode. If a PFS allows a file system to be locally accessed on all systems in a sysplex for a particular mode, then the PFS is *sysplex-aware* for that mode. If a PFS requires that a file system be accessed through the remote owning system from all other systems in a sysplex for a particular mode, then the PFS is *sysplex-unaware* for that mode.

Even if a PFS is sysplex-aware for a particular mode, if a non-owning system does not have DASD connectivity, the file system is accessed remotely through the owning system. For example, HFS is sysplex-unaware for read-write mode, because all non-owning systems must access read-write file systems through the remote owning system. The non-owning systems are said to be *sysplex clients*. However, HFS is sysplex-aware for read-only mode, which means that each system can access read-only file systems locally, and do not need to contact the owning system. AUTOMOVE is intended for sysplex-unaware file systems, where non-owning systems access the file systems remotely, to allow specification as to what will happen to the ownership of file systems owned by a system that is leaving the sysplex. For sysplex-aware file systems, since they tend to be accessed locally on each system, independent of other systems, ownership is simply moved and file system access continues as it was, regardless of the value of AUTOMOVE.

TFS file systems do not participate in move operations regardless of the AUTOMOVE setting. Automount-managed file systems are handled as AUTOMOVE if the file system is used locally.

Restriction: An AUTOMOVE file system cannot be moved to a system where OMVS has been shut down or where `F BPXOINIT,SHUTDOWN=FILEOWNER` has been issued.

Following is the behavior for file systems that are mounted in a mode for which the PFS is sysplex-aware:

1. MOUNT allows AUTOMOVE(YES) or AUTOMOVE(UNMOUNT). If AUTOMOVE(NO) or a system list is specified, it will be changed to AUTOMOVE(YES) and a message, BPXF234I, put in the hardcopy log. After the MOUNT, an attempt to change AUTOMOVE to NO or specify a system list using **chmount** is rejected.
2. A move of a file system that is AUTOMOVE(NO) or has a system list to a new owner that is at V1R6 or later will change the file system to AUTOMOVE(YES) if all systems in the sysplex are at least at V1R6 or later. Again, a message will be put in the hardcopy log. This applies to any way that a file system can be moved to a new owner, including chmount, PFS termination, soft shutdown, and dead system recovery.
3. Remount does not change the AUTOMOVE setting, even if the new mount mode is now sysplex-aware.

Note: For z/OS R7, AUTOMOVE(NO) or a system list is permitted for a sysplex-aware file system only if the file system is mounted in a mixed-release sysplex where one or more systems are at a release below V1R6.

Table 16 shows what happens during soft shutdown for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems. Soft shutdown is done by issuing one of the following MODIFY commands:

```
F BPXOINIT,SHUTDOWN=FILESYS
F BPXOINIT,SHTUDOWN=FILEOWNER
F OMVS,SHUTDOWN
```

A *leaf file system* refers to a file system that does not contain any file systems that are mounted on a directory within that file system. A *subtree* is the file system and all file systems that are mounted beneath that file system.

Table 16. Soft shutdown actions for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Unmounts the file system. The unmount will fail if it is not a leaf file system.	Unmounts the file system. The unmount will fail if it is not a leaf file system.
AUTOMOVE (no system list)	Moves the file system to any system. If the move fails, the unmount is not attempted and ownership does not change.	Moves the file system to any system. If the move fails, the unmount is not attempted and ownership does not change.
AUTOMOVE with a system list	Moves the file system to any system. If the file system cannot be moved, the unmount is not attempted and ownership does not change.	The move uses the system list. If the file system cannot be moved, the unmount is not attempted and ownership does not change.

Note: Automount-managed file systems are unmounted by a soft shutdown operation if the file system is not referenced by any other system in the

sysplex. If it is referenced by another system or systems, ownership of the file system is moved. If the move fails, an unmount is not attempted and ownership does not change.

Table 17 shows what happens during dead system takeover for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems. *Dead system takeover* is the action taken by systems in a sysplex when they attempt to take over ownership of file systems that were previously owned by a system that has just left the sysplex.

Note: Takeover is always attempted for file systems that are sysplex aware. Most of these systems already have the file system locally mounted.

Table 17. Dead system takeover for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE	Takeover is attempted by all systems. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is not attempted. The file system becomes unowned.
UNMOUNT	Takeover is attempted by all systems. The subtree is unmounted if it cannot be taken over by a new owning system.	Takeover is not attempted. The subtree is unmounted.
AUTOMOVE (no system list)	Takeover is attempted. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is attempted. The file system becomes unowned if it cannot be taken over by a new owning system.
AUTOMOVE with a system list	Takeover is attempted by all systems in the sysplex. The system list is ignored. The file system becomes unowned if it cannot be taken over by a new owning system.	Takeover is attempted and the INCLUDE or EXCLUDE system list is honored. If the takeover does not happen, the file system mounted under this file are unmounted.

Note: There is no attempt to take over automount-managed file systems if the file system is not referenced by any system that is eligible to attempt takeover. Automount-managed, unowned file systems will be unmounted.

Table 18 shows what happens during PFS termination for various AUTOMOVE settings for sysplex-aware and sysplex-unaware file systems.

Table 18. PFS termination for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Moves to any system. If the move fails, the subtree is unmounted.	The subtree is unmounted.
AUTOMOVE (no system list)	Moves to any system. If the move fails, the subtree is unmounted.	Moves to any system. If the move fails, the subtree is unmounted.
AUTOMOVE with a system list	Moves to any system, ignoring the system list. If the move fails, the subtree is unmounted.	The move uses the system list. If the move fails, the subtree is unmounted.

Table 19 on page 91 shows what happens when a move file system is requested to move a specific file system to any target system (wildcard is used). A move file system request can be issued with a SETOMVS operator command or a **chmount** shell command.

Table 19. Move all file systems from a system to any system for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT or AUTOMOVE (no system list)	Move is attempted to all systems.	Move is attempted to all systems.
AUTOMOVE with a system list	Move is attempted to all systems, ignoring the system list.	Move is attempted using the system list.

Table 20 shows what happens when a move file system is requested to do a multi-file system move, moving all file systems from a system to a specific target system. A move file system request can be issued with a SETOMVS operator command or a **chmount** shell command.

Table 20. Move all file systems from a system to a specific target system to any system for various AUTOMOVE settings

What happens if . . .	For sysplex-aware file systems	For sysplex-unaware file systems
NOAUTOMOVE or UNMOUNT	Move is attempted to the target system.	Move is not attempted.
AUTOMOVE (no system list)	Move is attempted to the target system.	Move is attempted to the target system.
AUTOMOVE with a system list	Move is attempted to the target system, ignoring the system list.	Move is attempted to the target system, ignoring the system list.

Rules:

- Define your version and sysplex root file systems as AUTOMOVE.
- Define your system-specific file systems as UNMOUNT.
- Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system under it as AUTOMOVE; otherwise, the file system defined as AUTOMOVE will not be recovered after a system failure until that failing system has been restarted.

Guidelines:

1. To ensure that the root is always available, use the default, which is AUTOMOVE.
2. For sysplex-unaware file systems that are mostly exported by the DFS or SMB server to their remote clients, consider specifying NOAUTOMOVE on the MOUNT statement. Then the file systems will not change ownership if the system is suddenly recycled, and they will be available for automatic re-export by DFS or SMB.

   Tip: Consider specifying NOAUTOMOVE because a file system can only be exported by the DFS or SMB server at the system that owns the file system. A file system can only be exported by the DFS or SMB server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported by DFS. The same holds true of file systems exported by SMB. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS or SMB clients. When an owning system recycles and a file system exported by DFS or SMB has been taken over by one of the other systems, DFS or SMB cannot automatically

re-export that file system. When an owning system is recycled and an exported file system has been taken over by one of the other systems, that file system will not be automatically reexported. The file system will have to be moved from its current owner back to the original system, the one that has just been recycled, and then exported again.

For more information about VERSION, SYSPLEX, SYSNAME and AUTOMOVE|NOAUTOMVE|UNMOUNT, see *z/OS MVS Initialization and Tuning Reference*.

Sysplex scenarios showing shared file system capability

Scenario 1: First system in the sysplex

Figure 42 and Figure 43 on page 94 show a z/OS UNIX file system configuration for a shared file system. Here, SYSPLEX(YES) and a value on VERSION are specified, and a directory is dynamically created on which the version file system data set is mounted. This type of configuration requires a sysplex root and system-specific file system.

```
BPXPRMxx for (SY1)

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')          1
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')   2
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')               3
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')          4
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')          5
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 42. BPXPRMxx setup — sharing file systems

1 This is the sysplex root file system, and was created by running the BPXISYSR job. To create a sysplex root file system that is a zFS, run the sample job BPXISYZR. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

2 This is the system-specific file system, and was created by running the BPXISYSS job. To create a system-specific file system that is a zFS, run the sample job BPXISYZS. It must be mounted read-write. NOAUTOMOVE is specified because this file system is system-specific and ownership of the file system should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME.** will resolve to **/SY1** during parmlib processing. This mount point is created dynamically at system initialization.

3 This is the old root file system (version file system).

Guideline: It should be mounted read-only. Its mount point is created dynamically and the name of the file system is the value specified on the VERSION statement in the BPXPRMxx member. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

4 This file system contains the system-specific **/dev** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./dev** will resolve to **/SY1/dev** during parmlib processing.

5 This file system contains system-specific **/tmp** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./tmp** will resolve to **/SY1/tmp** during parmlib processing.

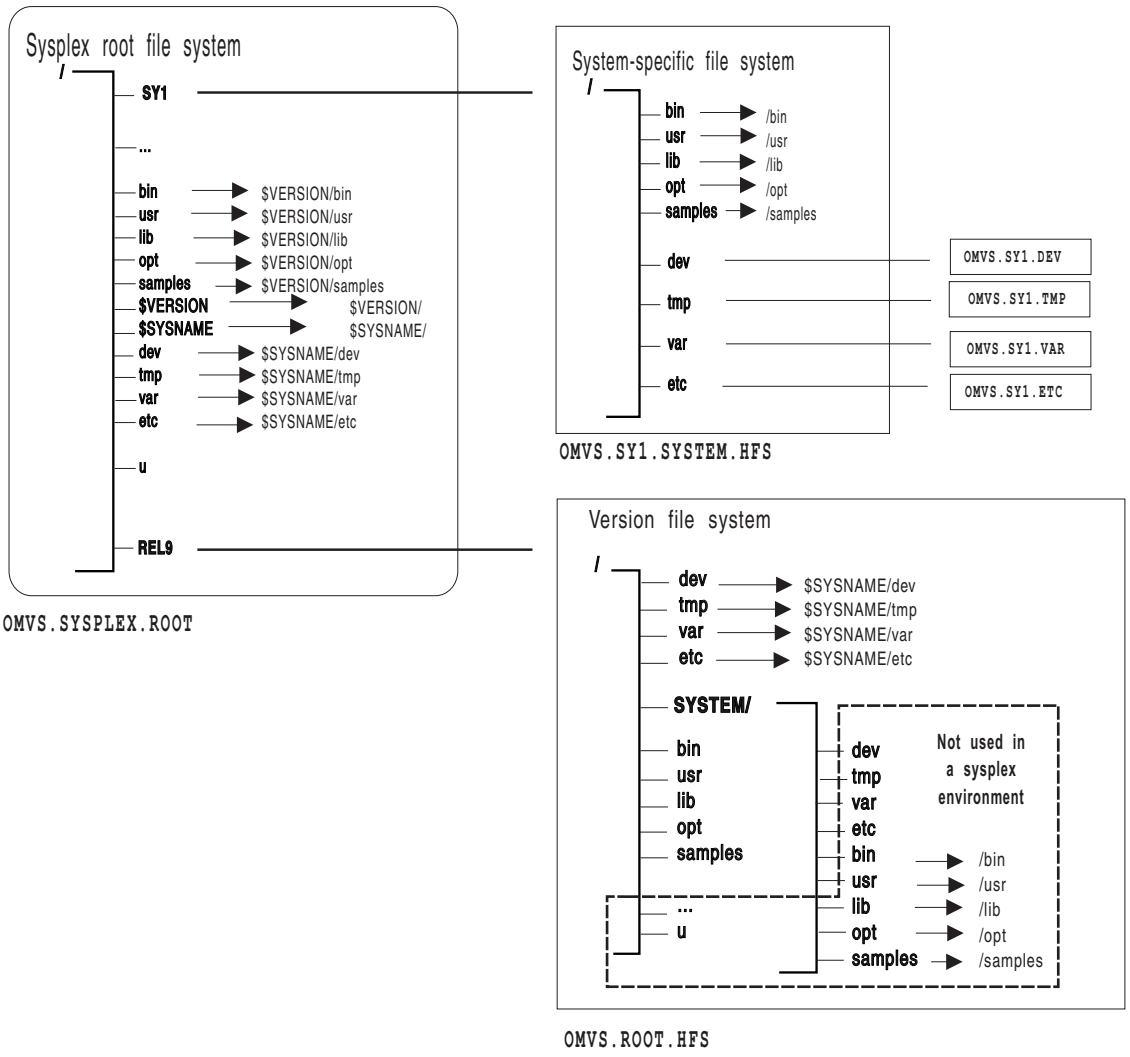

Figure 43. Shared file systems in a sysplex

If the content of the symbolic link begins with $VERSION or $SYSNAME, the symbolic link will resolve in the following manner:

- If you have specified SYSPLEX(YES) and the symbolic link for **/dev** has the contents **$SYSNAME/dev**, the symbolic link resolves to **/SY1/dev** on system SY1 and **/SY2/dev** on system SY2.
- If you have specified SYSPLEX(YES) and the content of the symbolic link begins with $VERSION, $VERSION resolves to the value *nnnn* specified on the VERSION parameter. Thus, if VERSION in parmlib is set to REL9, then $VERSION resolves to /REL9. For example, a symbolic link for **/bin**, which has the contents **$VERSION/bin**, resolves to **/REL9/bin** on a system whose $VERSION value is set to REL9.

In the above scenario, if **ls –l /bin/** is issued, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the path name **/REL9/bin**. The contents of **/REL9/bin** will now be displayed.

Scenario 2: Multiple systems in the sysplex using the same release level

Figure 46 on page 97 shows another SYSPLEX(YES) configuration. In this configuration, however, two or more systems are sharing the same version file system (the same release level of code). Figure 44 shows a sample BPXPRMxx for the entire sysplex (what IBM suggests) using &SYSNAME. as a symbolic name, and Figure 45 on page 96 shows a configuration where each system in the sysplex has its own BPXPRMxx. For our example, SY1 has its own BPXPRMxx and SY2 has its own BPXPRMxx.

```
One BPXPRMxx member to define file systems for the entire sysplex

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT FILESYSTEM('OMVS.USER.HFS')
MOUNTPOINT('u') AUTOMOVE
TYPE(HFS) MODE(RDWR)


MOUNT FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 44. Sharing file systems: one version file system and one BPXPRMxx for the entire sysplex

```
BPXPRMS1 (for SY1)                      BPXPRMS2 (for SY2)

FILESYSTYPE                             FILESYSTYPE
TYPE(HFS)                               TYPE(HFS)
ENTRYPOINT(GFUAINIT)                    ENTRYPOINT(GFUAINIT)
PARM(' ')                               PARM(' ')

VERSION('REL9')                         VERSION('REL9')
SYSPLEX(YES)                            SYSPLEX(YES)

ROOT                                    ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')         FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                    TYPE(HFS) MODE(RDWR)

MOUNT                                   MOUNT
FILESYSTEM('OMVS.SY1.SYSTEM.HFS')       FILESYSTEM('OMVS.SY2.SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1')                      MOUNTPOINT('/SY2')

MOUNT FILESYSTEM('OMVS.ROOT.HFS')       MOUNT FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)                    TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                 MOUNTPOINT('/$VERSION')

MOUNT FILESYSTEM('OMVS.SY1.DEV')        MOUNT FILESYSTEM('OMVS.SY2.DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/dev')                  MOUNTPOINT('/SY2/dev')

MOUNT FILESYSTEM('OMVS.SY1.TMP')        MOUNT FILESYSTEM('OMVS.SY2.TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/tmp')                  MOUNTPOINT('/SY2/tmp')
.
.
.
```

Figure 45. Sharing file systems: one version file system and separate BPXPRMxx members for each system in the sysplex

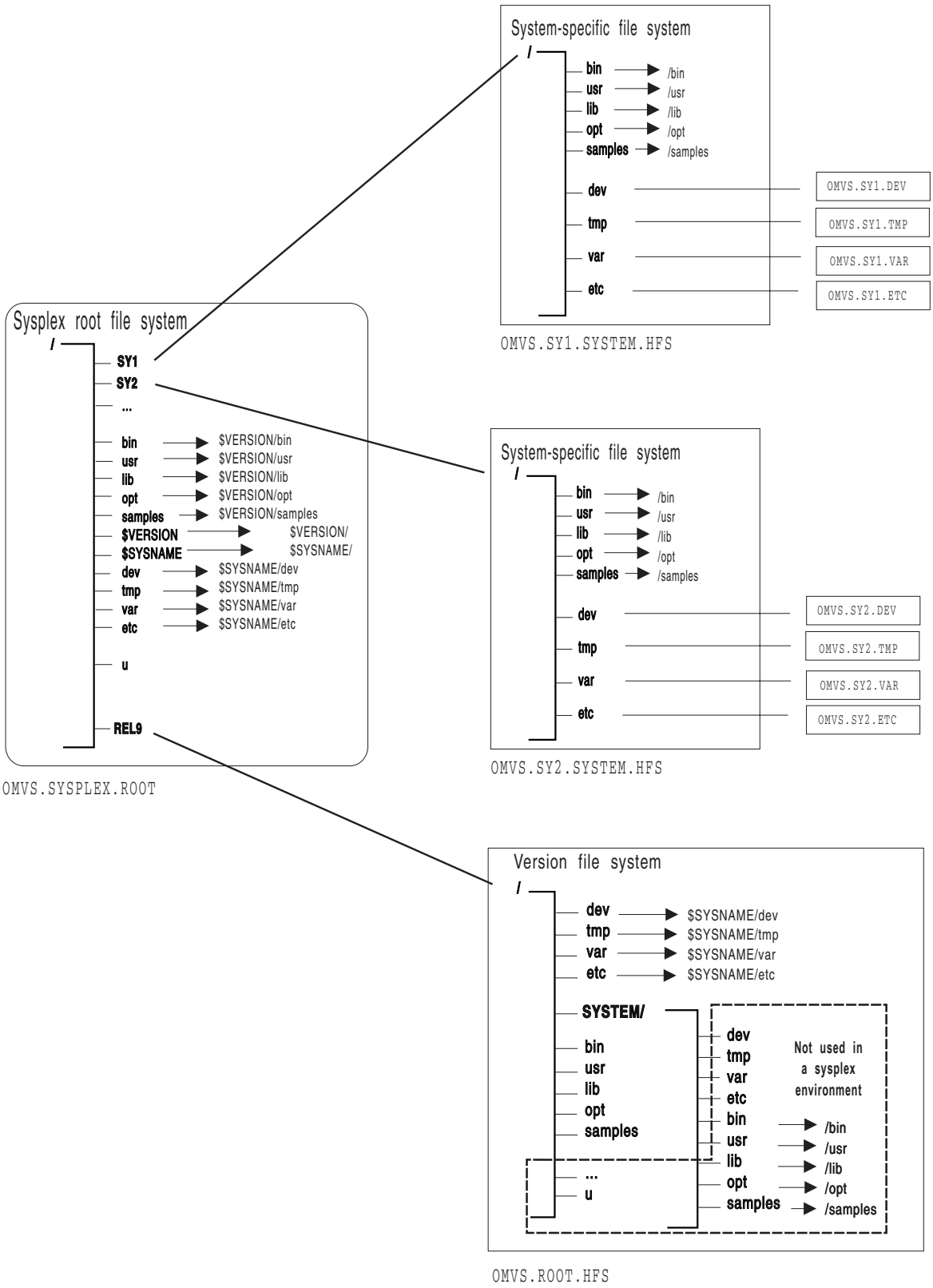

Figure 46. Sharing file systems in a sysplex: multiple systems in a sysplex using the same release level

In this scenario, where multiple systems in the sysplex are using the same version file system, if **ls –l /bin/** is issued from either system, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to

$VERSION/bin, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the path name **/REL9/bin**. The contents of this directory will be displayed.

Scenario 3: Multiple systems in a sysplex using different release levels

If your participating group is in a sysplex that runs multiple levels of z/OS, your configuration might look like the one in Figure 48 on page 99. In that configuration, each system is running a different level of z/OS and, therefore, has different version file system data sets; SY1 has the version file system named OMVS.SYSR9A.ROOT.HFS and SY2 has the version file system named OMVS.SYSR9.ROOT.HFS. Figure 47 shows two BPXPRMxx parmlib members that define the file systems in this configuration. Figure 49 on page 100 shows a single BPXPRMxx parmlib member that can be used to define this same configuration; it uses &SYSR1. as the symbolic name for the two version file system data sets.

```
BPXPRMxx (for SY1)                          BPXPRMxx (for SY2)

FILESYSTYPE                                 FILESYSTYPE
TYPE(HFS)                                   TYPE(HFS)
ENTRYPOINT(GFUAINIT)                        ENTRYPOINT(GFUAINIT)
PARM(' ')                                   PARM(' ')

VERSION('REL9A')                            VERSION('REL9')
SYSPLEX(YES)                                SYSPLEX(YES)

ROOT                                        ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')             FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                        TYPE(HFS) MODE(RDWR)


MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')     FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')                    MOUNTPOINT('/&SYSNAME.')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.SYSR9A.ROOT.HFS')          FILESYSTEM('OMVS.SYSR9.ROOT.HFS')
TYPE(HFS) MODE(READ)                        TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                     MOUNTPOINT('/$VERSION')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')            FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')                MOUNTPOINT('/&SYSNAME./dev')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')            FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')                MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 47. BPXPRMxx setup for multiple systems sharing file systems and using different release levels

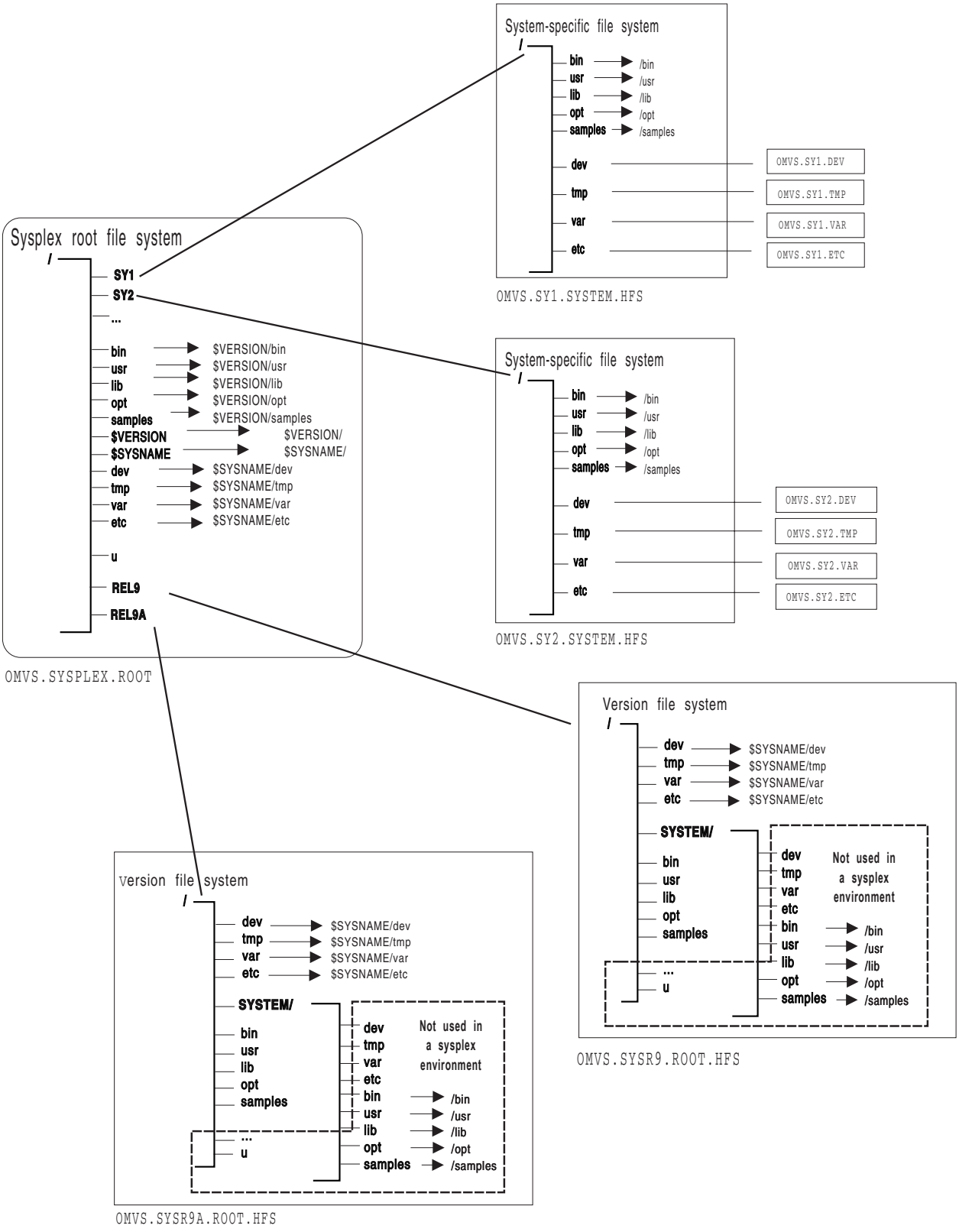

Figure 48. Sharing file systems between multiple systems using different release levels

In this scenario, for example, if **ls –l /bin/** is issued on SY1, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to

/SYSR9A on SY1, which makes the path name **/SYSR9A/bin**. The contents of this directory will now be displayed. If **ls –l /bin/** is issued on SY2, the contents of **/SYSR9/bin** will display.

From SY2 you can display information on SY1 by fully qualifying the directory.

Example: To view SY1's **/bin** directory:

```
ls –l /SY1/bin/
```

```
One BPXPRMxx member to define file systems for the entire sysplex
Using different releases

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('&SYSR1.')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT FILESYSTEM('OMVS.USER.HFS')
MOUNTPOINT('u') AUTOMOVE
TYPE(HFS) MODE(RDWR)


MOUNTFILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.&SYSR1..ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 49. One BPXPRMxx parmlib member for multiple systems sharing file systems and using different release levels

In order to use one BPXPRMxx parmlib file system member, we have used another system symbolic like &SYSR1. This system symbolic is used in the VERSION parameter and also as a qualifier in the version file system data set name.

Automount policies

Rule: You must keep the automount policies consistent across all the participating systems in the sysplex. The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

Keeping your automount policy consistent on all systems

Your automount policy most likely resided in the **/etc/auto.master** and **/etc/u.map files**. For those using shared file systems, each participating system has a separate **/etc** file system. In order for the automount policy to be consistent across participating systems, the same copy of the automount policy must exist in every system's **/etc/auto.master** and **/etc/u.map** files.

AUTOMOUNT is the preferred method of managing the **/u** directory. You do not need a mount statement for **/u** in the BPXPRMxx parmlib member.

For example, both SY1 and SY2 have the following files:

- **/etc/auto.master**

```
/u     /etc/u.map
```

- **/etc/u.map**

```
name       *
type       HFS
filesystem OMVS.<uc_name>.HFS
mode       rdwr
duration   60
delay      60
```

When the automount daemon initializes on SY1, it will read its local **/etc/auto.master** file to identify what directories to manage; in this case, it is **/u**. Next, the automount daemon will use the policy specified in the local **/etc/u.map** file to mount file systems with the specified naming convention under **/u**. The automount daemon on SY2 will perform similar actions. Because all mounted file systems are available to all participating systems in the sysplex, your automount policy must be consistent. This is true for the file system name specified in **/etc/u.map** and the values for other parameters in **/etc/u.map** and **/etc/auto.master**.

Moving file systems in a sysplex

You may need to change ownership of the file system for recovery or re-IPLing.

Tips:

- To check for file systems that have already been mounted, use the **df** command from the shell.
- The SETOMVS command used with the FILESYS, FILESYSTEM, mount point and SYSNAME parameters can be used to move a file system in a sysplex, or you can use the **chmount** command from the shell.

  Restriction: Do not move two types of file systems:

  – System-specific file systems
  – File systems that are being exported by DFS. You have to unexport them from DFS first and then move them

Examples:

1. To move ownership of the file system that contains **/u/wjs** to SY1:

```
chmount -d SY1 /u/wjs
```

2. To move ownership of the payroll file system from the current owner to SY2 using SETOMVS, issue:

   ```
   SETOMVS FILESYS,FILESYSTEM='POSIX.PAYROLL.HFS',SYSNAME=SY2
   ```

   or (assuming the mount point is over directory **/PAYROLL**)

   ```
   SETOMVS FILESYS,mountpoint='/PAYROLL',SYSNAME=SY2
   ```

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at **/dev** for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Also, opened FIFO files are automatically closed before the file system containing the FIFO is moved. They are closed because the in-storage FIFO data on the old system is not moved and is no longer accessible on new owning system.

Moving file systems to a back-level system

If you move a file system to a back-level system, existing NFS client connections to the files in that file system may be broken if Share Reservations are used. With Share Reservations, remote NFS clients can open files on z/OS in such a way that no one else can open that file until the first program finishes and closes the file. For more information about Share Reservations, see the BPX1VOP callable service in *z/OS UNIX System Services File System Interface Reference*.

Restrictions:

- A file system cannot be moved to a back-level system while there are active Share Reservations on any file in the file system. You will have to move the file system to a sysplex member at the z/OS V1R7 release level or later. Alternatively, you can stop the applications at the NFS clients who have put reservations on the files, or wait for them to finish.
- Share Reservations that attempt to deny reading or writing for files in a read-only file system are accepted but will not be enforced.
- File systems cannot be remounted from read-write to read-only or from read-only to read-write while there are Share Reservations established on any file in that file system.

If an NFS client is going to open a file that has Share Reservations set:

- That file must be owned by a system at the z/OS V1R7 level or higher before it can be opened.
- If the file is owned by a remote system that supports Share Reservations, they will be enforced at the owner for all opens within the sysplex.
- If the file is owned by a remote system at a lower level, the client's open will fail. The reason code of the failure will indicate that the file system has to be moved to a sysplex member that is at the z/OS V1R7 release or later.

If the system goes down and there are Share Reservations on a file owned by a remote system:

1. If the file system is taken over by another z/OS V1R7 or later system, the reservations are reestablished at the new owner and enforced there.
2. If the file system is taken over by an owner that does not support Share Reservations, the NFS client's open is invalidated and subsequent operations from that client for this open are rejected. If you move the file system to a sysplex member that supports Share Reservations, the file can be reopened as

it was before. You can use the AUTOMOVE parameter of the MOUNT command to restrict these takeovers to the systems that do support Share Reservations.

Shared file system implications during system failures and recovery

File system recovery in a shared file system environment takes into consideration file system specifications such as the sysplex awareness capability, the AUTOMOVE value, and whether or not the file system is mounted read-only or read-write.

Takeover is always attempted for file systems that are sysplex-aware, regardless of the AUTOMOVE value. Most systems in the sysplex already have the file system locally mounted, so the ownership is simply moved and file system access continues as it was. Table 17 on page 90 describes the recovery actions that occur for each combination of settings.

Generally, when an owning system fails, ownership of a file system that is mounted AUTOMOVE is moved to another system and the file system remains usable. However, if a file system is mounted read-write and the owning system fails, then all file system operations for files in that file system will fail. This happens because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered. The BPX1CLO and BPX1OPN callable services are discussed in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be started again.

In some situations, even though a file system is mounted AUTOMOVE, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes unowned; if this happens, you will see message BPXF213E. This is true if the file system is mounted either read-write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that are owned by another system are still usable. However, all file operations for the unowned file system will fail until a new owner is established. The shared file system support will continue to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for shared file system. If a subsequent recovery attempt succeeds, the file system transitions from the unowned to the active state.

Applications using files in unowned file systems will need to close (BPX1CLO) those files and reopen (BPX1OPN) them after the file system is recovered.

Sysplex-unaware file systems that are mounted NOAUTOMOVE will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Note that since the file system still exists in the file system hierarchy, the file system mount point is still in use. File systems that are mounted below a NOAUTOMOVE file system will not be accessible via path name when the NOAUTOMOVE file system becomes available.

Do not mount AUTOMOVE file systems within NOAUTOMOVE file systems. When a NOAUTOMOVE file system becomes unowned and there are AUTOMOVE file systems mounted within it, those AUTOMOVE file systems will retain a level of availability, but only for files that are already open. When the NOAUTOMOVE file

system becomes unowned, it will not be possible to perform path name lookup through it to the file systems mounted within it, which will make those file systems unavailable for new access. When ownership is restored to the unowned file system, access to the file systems mounted within it is also restored.

Managing the movement of data

File systems can be managed so as to maximize their availability when systems exit the participating group. You have more control over this when the outage is planned, but there are steps you can take to help manage the placement of data in the event of a system failure.

Recovery processing for the file systems that are owned by a failed system is managed internally by all the systems in the participating group. If you want special considerations for the placement of certain file systems, you can use the options provided by the various mount services to specify the original owner and subsequent owners for a particular file system.

| "Customizing BPXPRMxx for a shared file system" on page 87 describes the behavior of the various AUTOMOVE options.

| Table 21 shows the AUTOMOVE options that you can use with the MOUNT command to manage sysplex-unaware file systems. (Table 22 on page 105 covers the AUTOMOVE options for sysplex-aware file systems.)

Table 21. Automove options supported by the MOUNT command for sysplex-unaware file systems

UNMOUNT	Attempts will not be made to keep the file system active when the current owner fails. The file system will be unmounted when the owner is no longer active in the participating group, as well as all the file systems mounted within it. It is suggested for use on parmlib mounts for system-specific file systems, such as those that would be mounted at **/etc, /dev**, **/tmp** and **/var**.
NOAUTOMOVE	Attempts will not be made to keep the file system active when the current owner fails. The file system will remain in the hierarchy for possible recovery when the original owner reinitializes. Use this option on mounts for system-specific file systems if you want to have automatic recovery when the original owner rejoins the participating group. When the NOAUTOMOVE option is used, the file system becomes unowned when the owning system exits the participating group. The file system remains unowned until the last owning system restarts, or until the file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use. An unowned file system is a mounted file system that does not have an owner. Because it still exists in the file system hierarchy, it can be recovered or unmounted.
AUTOMOVE (no system list)	Recovery of the file system is to be performed when the current owner fails. This option is suggested for use on mounts of file systems that are critical to operation across all the systems in the participating group. AUTOMOVE is the default.

Table 21. Automove options supported by the MOUNT command for sysplex-unaware file systems (continued)

AUTOMOVE with a system list	AUTOMOVE(EXCLUDE|INCLUDE,*sysname1*,*sysname2*,...*sysnameN*) specifies managed recovery of the file system if the current owner fails. • Use the EXCLUDE list to prevent recovery of a file system from transferring ownership to a particular system, or set of systems, in the participating group. When the current owner fails, recovery of the file system is performed to an owner outside the exclude list. • Use the INCLUDE list to ensure that recovery of a file system will transfer ownership only to a particular system or set of systems in the participating group. Recovery of the file system is performed in priority order only by the list of systems specified in the INCLUDE list. **Restriction:** Only use this option on mounts of file systems that are critical to operation across a subset of systems in the participating group, or when you do not want certain systems in the participating group to have ownership of the file system. If recovery processing fails to establish a new owner for the file system, the file system is unmounted, along with all the file systems mounted within it.

Table 22 shows the AUTOMOVE options that you can use with the MOUNT command to manage sysplex-aware file systems.

Table 22. Automove options supported by the MOUNT command for sysplex-aware file systems

UNMOUNT	Attempt will be made to keep the file system active when the current owner fails. The file system and all file systems that are mounted beneath it will be unmounted if the file system cannot be taken over by a new owning system.
AUTOMOVE	Recovery of the file system is to be performed when the current owner fails. This option is suggested for use on mounts of file systems that are critical to operation across all the systems in the participating group. AUTOMOVE is the default.

Note: AUTOMOVE or UNMOUNT are the only options you can use for file systems that are mounted in a mode for which they are capable of being directly mounted to the PFS on all systems (sysplex-aware). If you specify any other option on a MOUNT, it is ignored and you will see message BPXF234I.

Most of the z/OS UNIX interfaces that provide for mounting file systems (such as TSO, shell, ISHELL, and BPX2MNT) support some form of the options described in "Customizing BPXPRMxx for a shared file system" on page 87. See the associated documentation for the exact syntax.

Guideline: To ensure that the root is always available, use the default, which is AUTOMOVE.

For file systems that are mostly used by DFS or SMB clients, consider specifying NOAUTOMOVE on the MOUNT statement. Then the file systems will not change ownership if the system is suddenly recycled, and they will be available for automatic re-export by DFS or SMB. Specifying NOAUTOMOVE is suggested because a file system can only be exported by the DFS or SMB server at the

system that owns the file system. Once a file system has been exported by DFS or SMB, it cannot be moved until it has been unexported from DFS or SMB. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS or SMB clients. When an owning system recycles and a DFS- or SMB-exported file system has been taken over by one of the other systems, DFS or SMB cannot automatically re-export that file system. The file system will have to be moved from its current owner back to the original DFS or SMB system, the one that has just been recycled, and then exported again.

Shared file system implications during a planned shutdown of z/OS UNIX

These sections contain the procedures to use when shutting down z/OS UNIX.

- Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN in the Managing Operations chapter.
- Steps for shutting down z/OS UNIX usinG F BPXOINIT,SHUTDOWN=... in the Managing Operations chapter.

It is important that you understand the system actions that result when you use those procedures.

The current automove option dictates if and how the participating group recovers file system ownership from an exited system. It has no effect on the manual movement of the file system. However, when you are using the procedures for shutting down z/OS UNIX to prepare for a planned system outage, the automove option does apply. This can be explained with the following rationale:

- A system failure does not provide any means for manual intervention. The automove option provides a set of rules for automatic recovery.
- A request to move a file system manually is a deliberate action on behalf of an authorized user or administrator, and should override any rules for automatic recovery.
- Using tools to prepare for a system outage is also a deliberate action on behalf of an authorized user or administrator, but you are using these tools in an environment that can be customized to allow for additional manual intervention. You can synchronize data before the system outage, and then manage the planned outage in the same way as the unplanned outage, by making use of the automatic recovery rules that are supplied by the automove options. If you prefer some other action, you can perform manual intervention to move specific file system ownership before you use these methods for shutdown preparation.

Use F OMVS,SHUTDOWN to shut down file systems. If this is not appropriate for your installation, use the F BPXOINIT,SHUTDOWN=... procedure.

State of file systems after shutdown

File systems on the system where the shutdown was issued are immediately unmounted. As a result, data is synched to disk. For shared file systems, one of the following actions is done on the file systems that are owned by the system where the command was issued.

- Unmount if automounted or if a file system was mounted on an automounted file system.
- Move to another system if an AUTOMOVE(YES) was specified.
- Unmount for all other file systems.

File systems that are not owned by the system on which the shutdown was issued are not affected.

File system initialization

When you are preparing to bring a system back into the participating group after it has left, it is helpful to understand the coordination that occurs among the systems that are already participating in the group. You might see delays in the availability of the entering system because of activity occurring elsewhere in the sysplex. Although it is possible to bring up multiple systems simultaneously, when they reach the point of z/OS UNIX initialization, their processing is serialized so as to allow only one system at a time to initialize z/OS UNIX.

Other examples of activities occurring on other active systems that can cause the initializing system to experience delays are

- Unmounting a file system
- Changing ownership of a file system
- Recovering for systems that have left the participating group

Before it rejoins the participating group, a system processes all the file systems that are listed in the current hierarchy of the participating group. It also attempts to reclaim any unowned file systems that it previously owned when it was part of the participating group. It does not attempt to reclaim those file systems that were successfully moved or recovered to another system in the sysplex.

During initialization, any new MOUNT statements in the BPXPRMxx parmlib member are processed, which makes those file systems available for use within the participating group after they are successfully mounted.

While a system is initializing in a sysplex, critical file systems that are necessary for initialization to complete successfully might become unavailable due to a system outage. When a system is removed from the sysplex, there is a window of time during which any file systems it owned will become inaccessible to other systems. This window of time occurs while other systems are being notified of the system's exit from the sysplex and before they start the cleanup for that system.

Ideally, ownership of critical file systems will have been moved to other systems before the system exits. If that has not happened, there will be a window of time during which these critical file systems are unowned. If the initializing system requires access to these critical file systems during this window, there will likely be mount failures that prevent the initialization from completing successfully. To avoid this situation, you must make sure that any system that is being removed from the sysplex does not own any critical file systems.

Locking files in the sysplex

You can lock all or part of a file that you are accessing for read-write purposes by using the byte range lock manager (BRLM).

With z/OS V1R6, the lock manager is initialized on every system in the sysplex. This is known as distributed BRLM, and it is the only supported byte range locking method when all systems are at the V1R6 level. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system. Distributed BRLM was formerly an option on previous levels of z/OS, and central BRLM was formerly the default.

When a system failure occurs, all byte range locks are lost for files in file systems owned by that system. To maintain locking integrity for those locked files that are still open on surviving systems, z/OS UNIX prevents further locking or I/O on those

files. In addition, the applications are signaled, in case they never issue locking requests or I/O. Running applications that did not issue locking requests and did not have files open are not affected.

After a failure where byte range locks are lost, z/OS UNIX provides the following information to processes that have used byte range locking:

- Access to open files for which byte range locks are held by any process will result in an I/O error. The file must be closed and reopened before use can continue.
- A signal is issued to any process which has made use of byte range locking. By default, a SIGTERM signal is issued against every such process, and an EC6 abend with reason code 0D258038 will terminate the process. If you do not want the process to be terminated, the process can use BPX1PCT (the physical file system control callable service) to specify a different signal for z/OS UNIX to use for notifying the process that the BRLM has failed. Any signal can be used for this purpose, thus allowing the user or application the ability to catch or ignore the signal and react accordingly.

  z/OS MVS System Codes describes the system completion code EC6 and its associated reason codes. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about BPX1PCT.

Using distributed BRLM

With z/OS V1R6, a file system can be moved while byte range locks are held for files in the file system. When a file system changes owners, the corresponding locking history changes BRLM servers at the same time. (This is not the case when a system failure occurs, as was discussed in “Locking files in the sysplex” on page 29.) For this reason, distributed BRLM is now the only supported method when all systems are at the z/OS V1R6 level.

If you are already running with a z/OS UNIX CDS indicating that distributed BRLM is enabled, there is no change required to activate distributed BRLM for z/OS V1R6. Likewise, if your sysplex only has systems at the z/OS V1R6 level, there is no change required, because distributed BRLM is the default. z/OS V1R6 systems ignore the z/OS UNIX CDS DISTBRLM setting.

However, if you migrate to z/OS V1R6 by running mixed levels in a sysplex, you should enable distributed BRLM before IPLing the z/OS V1R6 system because a z/OS V1R6 system may attempt to activate distributed BRLM when the central BRLM server leaves the sysplex, regardless of the z/OS UNIX CDS setting. The inconsistency between distributed BRLM being active and central BRLM being defined in the z/OS UNIX CDS can cause an EC6-BadOmvsCds abend on downlevel systems. It is a notification-only abend indicating that the CDS should be updated. z/OS UNIX will still operate normally, and distributed BRLM will be active in the sysplex.

Before bringing the first z/OS V1R6 system into the sysplex, enable distributed BRLM by using the IXCL1DSU utility to update the BPXMCDS couple data set and then activate it. Message BPXF235I is issued when the switch from central BRLM to distributed BRLM occurs.

See “Creating a couple data set (CDS)” on page 84 for an example of the COUPLExx parmlib member. For more information about the BPXMCDS couple data set and the IXCL1DSU utility, see *z/OS MVS Setting Up a Sysplex*.With V1R6, if you run your IXCL1DSU job to create a z/OS UNIX couple data set, distributed BRLM is set up as a default.

Requirement: Remove the central BRLM server from the sysplex via an IPL or OMVS shutdown (F OMVS,SHUTDOWN). This eliminates all file locking history in the sysplex and allows the new distributed BRLM servers to start with a clean locking history.

Mounting file systems using symbolic links

You can mount different file systems at a logical mount point that resolves to a different path name on different systems.

While $VERSION/ can be used to differentiate a path based on the version level of a system and $SYSNAME/ can be used to differentiate on each system, you can use special identifiers to mount file systems using symbolic links. These are $SYSSYMR/template and $SYSSYMA/template.

Restrictions:

1. Like $VERSION/ and $SYSNAME/, the identifiers need to be at the beginning of the link name.
2. Only the first occurrence of $SYSSYMR/ or $SYSSYMA/ in the link name will be recognized as an identifier for which the remaining text requires substitutions. Any other identifiers after the first one will remain as is in the resolved linkname.
3. Text must follow a $SYSSYMR/ or $SYSSYMA/ in order for it to be recognized as a valid identifier with text containing symbols to be resolved.
4. Any system symbol in the symbolic link text that is recognized by the ASASYMBM service will be resolved. However, only static system symbols should be used in order to avoid unexpected results. These symbols are assigned a value at initialization. For information about system symbols, see *z/OS MVS Initialization and Tuning Reference*.

Tip: You can use D SYMBOLS to display the current settings of system symbols.

Examples

These examples assume that the standard MVS symbol &SYSR1. resolves to OSV315 on SY1 and resolves to OSV315B on SY2.

1. If the symbolic link is /x/y/sym1, and the symbolic link contains $SYSSYMR/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to OSV315/resdir. Because it is a relative path name (the identifier was $SYSSYMR/), the resulting path name will be /x/y/OSV315/resdir.

   Example: On a mount, passing /x/y/sym1 as the input mount point path name, the mount point would be: /x/y/OSV315/resdir on SY1.

   - If the symbol &SYSR1. resolves to OSV315B on SY2, a lookup of the same path name would result in a mount point of /x/y/OSV315B/resdir.
   - On a v_readlink syscall, passing the VnToken for the symbolic link, the output linkname would be OSV315/resdir on SY1 or OSV315B/resdir on SY2.
2. If the symbolic link is /x/y/sym1, and the symbolic link contains $SYSSYMA/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to /OSV315/resdir. Because it is an absolute path name (the identifier was $SYSSYMA/), the resulting path name will be /OSV315/resdir.

   Example: On a mount, passing /x/y/sym1 as the input mount point path name, the mount point would be /OSV315/resdir on SY1.

   - If the symbol &SYSR1. resolves to OSV315B on SY2, a lookup of the same path name from SY2 would result in a mount point of /OSV315B/resdir.

- On a v_readlink syscall, passing the VnToken for the symbolic link, the output linkname would be /OSV315/resdir on SY1 and /OSV315B/resdir on SY2.

Mounting file systems using NFS client mounts

With the z/OS NFS server, the client has remote access to z/OS UNIX files from a client workstation. Using the Network File System, the client can mount all or part of the file system and make it appear as part of its local file system. From the workstation, the client user can create, delete, read, write, and treat the host-located files as part of the workstation's own file system.

In a similar way, the z/OS NFS client gives users remote access to files on an NFS server. Using NFS, the user can mount all or part of the remote file system and make it appear as part of the local z/OS file hierarchy. From there, the user can create, delete, read, write, and treat the remotely located files as part of their own file system.

In a sysplex, the NFS Client-NFS Server relationship is as follows: the data that becomes accessible is accessible from any place in the sysplex as long as at least one of the systems has connectivity to the NFS server.

Rule: Entries in the NFS Server Export Data Set can control which UNIX directories can be mounted by client users. When specifying path names in this data set, you must specify fully qualified path names. That is, the use of symbolic links in this data set are not supported.

File system availability

In the shared file system environment, file system availability and accessibility depend on a number of important factors. These factors can vary depending on how a file system is mounted and the capability of the file system to manage itself in a sysplex environment. After you set up the shared file system environment for cross-system communication ("Procedures for establishing shared HFS in a sysplex" on page 7), it will be helpful to understand how file system availability is provided to your systems, and what kinds of actions can cause interruptions to that availability.

Minimum setup required for file system availability

Rules:

- For DASD file systems, at least one system in the shared file system group needs to have a physical I/O path to the volume where the file system resides and the volume is varied online. Without connectivity from at least one system, the file system will not be available to any of the systems in the shared file system group. Connectivity from one system can provide shared file system accessibility to the file system for all other systems in the shared file system group.
- All systems need to have the physical file system (PFS) started. Accomplish this by placing the appropriate FILESYSTYPE statement in the BPXPRMxx parmlib member that is used in the configuration. Additionally, any necessary subsystems required by the PFS must be started and configured, especially if this system is to function as the file system owner. For example, the NFS Client PFS requires that the TCP/IP subsystem be started and a network connection configured.

Read-write connections for non-sysplex aware file systems: Most physical file systems (PFSes) allow only one connection for update at a time. Such file systems are called *non-sysplex aware for update*. This is directly related to the mount mode

of the file system. With HFS, for example, only one system can actually connect to the file system with a mode of RDWR. That system is called the *file system owner*.

The other systems that want to participate in shared file systems will also request a RDWR mount, but their access will be provided via cross-system messaging with the file system owner which has already established the read-write connection. These systems are called *file system clients*. When the file system owner becomes unavailable (for example, through system shutdown), it will be important for another system (one of the file system clients) to have the file system volume varied online so that a new owner can be established. This helps ensure maximum file system availability in the shared file system group.

Read-write connections for sysplex-aware file systems: Some PFSes can handle multiple concurrent connections for update. They are capable of managing the serialization of such requests. Such file systems are called *sysplex aware for update*. Most network file systems have this capability. NFS Client is one such file system type.

For a read-write mount to NFS Client, each system in the shared file system group will make a direct connection to NFS. The first system to make such a connection is still called the file system owner. All subsequent systems to make a direct connection are considered non-owners, rather than clients. This type of multiple direct connection for read-write access allows for maximum I/O performance by eliminating the need to send requests to the file system owner.

However, sometimes a non-owning system cannot make a direct connection to the PFS even after meeting the minimum requirements (for example, sometimes requests to NFS Client time out before they are satisfied). That system might be given a cross-system messaging connection, making it a client to the file system. While this is not the optimal mount mode for this type of file system, it does allow access to the file system.

Read-only connections for non-sysplex aware file systems: There may be some physical file systems that do not support multiple concurrent connections for read-only access. These are called *non-sysplex aware for readonly*, and are handled the same as the read-write connections for non-sysplex aware file systems.

Read-only connections for sysplex-aware file systems: Physical file systems that support multiple concurrent connections for read-only access are called *sysplex aware for readonly*. The HFS physical file system falls into this category. Such file systems are handled the same as the read-write connections for sysplex aware file systems. The read-only connections are attempted locally for each system in the shared file system group, but if the file system volume is not online to a system, then the system becomes a client to the file system via cross-system messaging with the owner.

Situations that can interrupt availability

Some situations may cause interruptions to file system availability on one or more systems. Following is a list of some of the most common causes. It is not meant to be an exhaustive list.

- **Loss of the file system owner.** If the file system owner leaves the shared file system group (through system failure, soft shutdown, VARY, XCF, OFFLINE, or some other means), an attempt may be made to establish another file system owner if requested by the AUTOMOVE specification of the mount. If a new file system owner cannot be established, the file system will become unowned. It will

be unavailable until the original owner can reclaim it, or until another owner is established through subsequent automated recovery actions performed by shared file system.

- **PFS termination.** If a PFS terminates on one system, it can affect file system availability on other systems.
  - Prior to z/OS V1R2, if a PFS terminates on one system, all file systems of that type are unmounted across the sysplex.
  - In z/OS V1R2 and later, if a PFS terminates on one system, any file systems of that type that are owned by other systems are not affected. File systems of that type are moved to new owners whenever possible if they are owned by the system where the PFS is terminating and are automovable. These file systems remain accessible to other systems. If they cannot be moved to new owners, they are unmounted across the sysplex. It may not be possible to move a file system due to a lack of connectivity from other systems, or if the file system containing the mount point for the file system needed to be moved but could not be.
- **VARY volume,OFFLINE**. When the volume for a file system is varied offline, it will make that file system inaccessible to that system. However, if the volume is online to other systems, it may still be accessible to those systems and to other systems via cross-system messaging. This would be the case for sysplex-aware file systems for read-write or read-only access. Unlike loss of the file system owner, varying a file system volume offline will not result in any attempt by the system to restore accessibility to systems on which it is lost.

Tuning z/OS UNIX performance in a sysplex

The intersystem communication required to provide the additional availability and recoverability associated with z/OS UNIX shared file system support, affects response time and throughput on R/W file systems being shared in a sysplex.

For example, assume that a user on SY1 requests a read on a file system mounted R/W and owned by SY2. Using shared file system support, SY1 sends a message requesting this read to SY2 via an XCF messaging function:

```
SY1 ===> (XCF messaging function) ===> SY2
```

After SY2 gets this message, it issues the read on behalf of SY1, and gathers the data from the file. It then returns the data via the same route the request message took:

```
SY2 ===> (XCF messaging function) ===> SY1
```

Thus, adding z/OS UNIX to a sysplex increases XCF message traffic. To control this traffic, closely monitor the number and size of message buffers and the number of message paths within the sysplex. It is likely that you will need to define additional XCF paths and increase the number of XCF message buffers above the minimum default. For more information about signaling services in a sysplex environment, see *z/OS MVS Setting Up a Sysplex*.

You should also be aware that because of I/O operations to the CDS, every mount request requires additional system overhead. Mount time increases as a function of the number of mounts, the number of members in a sysplex, and the size of the CDS. You will need to consider the effect on your recovery time if a large number of mounts are required on any system participating in a shared file system.

DFS considerations

A file system can only be exported by the DFS server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported by DFS.

To recover from system outages, you need to weigh sysplex availability against availability to the DFS and Server Message Block (SMB) clients. When an owning system recycles and a DFS-exported file system has been taken over by one of the other systems, DFS will not be able to automatically reexport that file system. The file system will have to be moved from its current owner back to the original DFS system, the one that has just been recycled, and then reexported.

Tip: For file systems that are mostly for use by DFS clients, you should consider specifying NOAUTOMOVE on the MOUNT statement. If you specify NOAUTOMOVE, the file systems will not be taken over if the system is recycled, and they will be available for automatic reexport by DFS.

Chapter 4. Changes for z/OS UNIX System Services Command Reference

chmount — Change the mount attributes of a file system

Format

chmount [**–DRrw**] [**-d***sysname*] [**–d** *destsys*] [**–a** **yes**|**no**|**unmount**|**include,***sysname1,...,sysnameN*|**exclude,***sysname1,...,sysnameN*] *pathname...*

Description

The **chmount** shell command, located in **/usr/sbin**, changes the mount attributes of a specified file system.

Rule: A **chmount** user must have UID(0) or at least have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

Options

–a **yes**|**no**|**unmount**|**include,***sysname1,...,sysnameN*|**exclude,***sysname1,...,sysnameN*
The **–a** option specifies the AUTOMOVE attribute of the file system in a sysplex environment where systems are exploiting the shared file system capability.

–a yes allows the system to automatically move logical ownership for the file system as needed. This is the default.

–a no prevents ownership movement in some situations.

–a umount unmounts the file system in some situations.

–a include,*sysname1,...,sysnameN* specifies a list of systems, in priority order, to which the file system's ownership can be moved. **include** can be abbreviated to **i**.

–a exclude,*sysname1,...,sysnameN* specifies a list of systems, in priority order, to which the file system's ownership cannot be moved. **exclude** can be abbreviated to **e**.

See *z/OS UNIX System Services Planning* for details about the behavior of the AUTOMOVE options.

–D Reassigns logical ownership of a file system to any available file system participating in shared file system.

–d *destsys*
To designate a specific reassignment, use **–d** *destsys*, where *destsys* becomes the logical owner of a file system in a shared file system environment.

–R Changes the attributes of a specified file system and all file systems mounted below it in the file system hierarchy.

–r Switches the specified file system to read-only mode.

–w Switches the specified file system to read-write mode.

pathname... specifies the pathnames to use for locating the file systems that need attributes changed.

Example

To move ownership of the file system that contains **/u/wjs** to SY1:

```
chmount -d SY1 /u/wjs
```

Usage Note

The pathname for **chmount/unmount** is a node so symlinks can not be followed unless a trailing slash is added to the symbolic link name. For example, if **/etc** has been converted into a symbolic link, **/etc -> $SYSNAME/etc**, issuing **chmount -w /etc** (without the trailing slash) will result in trying to **chmount -w /etc -> $SYSNAME/etc**. This may result in RACF® errors depending on the security access for the symlinked file. However, adding the trailing slash, by specifying **chmount -w /etc/** the symlink will be followed and RACF will determine the access from the symlinked file.

Exit Values

0 Successful completion

Related Information

mount, **unmount**

---

mount — Logically mount a file system

Format

mount [**–t** *fstype*] [**–rv**] [**–a yes**|**include,***sysname1,...,sysnameN* |**exclude,***sysname1,...,sysnameN* |**no**|**unmount**] [**–o** *fsoptions*] [**–d** *destsys*] [**–s nosecurity**|**nosetuid**] **–f** *fsname pathname*[**-w***n*]

mount –q [**–d** *destsys*][**–v**] *pathname*

File Tag Specific Option:

mount [**–c** *ccsid*,**text**|**notext**]

Description

The **mount** shell command, located in **/usr/sbin**, is used to mount a file system or list all mounts over a file system.

Note: A **mount** user must have UID(0) or at least have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

Options

–a yes|**include,***sysname1,...,sysnameN* |**exclude,***sysname1,...,sysnameN* |**no**|**unmount**

The **–a** option specifies the AUTOMOVE attribute of the file system in a sysplex environment where systems are exploiting the shared file system capability.

–a yes allows the system to automatically move logical ownership for the file system as needed. This is the default.

–a include,*sysname1,...,sysnameN* specifies a list of systems, in priority order, to which the file system's ownership can be moved. **include** can be abbreviated to **i**.

–a exclude,*sysname1,...,sysnameN* specifies a list of systems, in priority order, to which the file system's ownership cannot be moved. **exclude** can be abbreviated to **e**.

–a no prevents ownership movement in some situations.

–a umount unmounts the file system in some situations.

See *z/OS UNIX System Services Planning* for details about the behavior of the AUTOMOVE options.

–d *destsys*
Specifies the name of the system in a shared file system environment that will be the logical owner of the mount. Note, if **–q** is specified, the **mount –q** output will only list mounts that are owned by *destsys*.

–f *fsname*
Names the file system to be mounted. All file system names must be unique. File system names are case sensitive. However, if the file system type is HFS, *fsname* will be translated to uppercase. The file system name has a maximum length of 44 characters, any additional characters will be truncated. Options **–q** and **–f** are mutually exclusive, but one must be specified.

-w*n* Specifies the amount of time the mount will wait in seconds for async mounts to complete. If *n* is specified as a 0 the wait will be indefinite. This option flag is tolerated on any form of the mount command and is ignored if not appropriate (no wait needs to be done).

–o *fsoptions*
Specifies an option string to be passed to the file system type. NFS, for example, uses this to identify the remote server and the object on that server. The format and content are specified by the physical file system that is to perform the logical mount. You can specify lowercase or uppercase characters. Enclose the string in single quotes.

–q Prints a list of pathnames for the mountpoints of file systems mounted over a another file system, including that system. Options **–q** and **–f** are mutually exclusive, but one must be specified. If **–v** is not specified, only pathnames for mountpoints are printed. Note that the output of **mount –q** can be used by the **unmount** utility as input. See “Examples” on page 118.

–r Specifies mounting a file system read-only.

–s nosecurity|nosetuid
Specifies that a file system is unsecured. Setuid, setgid, APF and program controlled attributes are ignored when you use **nosetuid**. To additionally disable authorization checking, use **nosecurity**. Minimum unique abbreviations can be used for the option arguments.

Note: When an HFS is mounted with the NOSECURITY option enabled, any new files or directories that are created will be assigned an owner of UID 0, no matter what UID issued the request.

–t *fstype*
Identifies the file system type. *fstype* may be entered in mixed case but will be treated as upper case. If this option is not specified, the default is **–t** HFS.

–v Verbose output. Includes additional information, if available, on output. If **–v** is specified on the **mount** command and the mount fails, the file system name that had the mount failure will be included in the failure information.

pathname specifies the pathname for the mountpoint.

File Tag Specific Option

–c *ccsid*,**text|notext**

Specifies the file tag that will be implicitely set for untagged files in the mounted file system.

ccsid Identifies the coded character set identifier to be implicitly set for the untagged file. *ccsid* is specified as a decimal value from 0 to 65535. However, when text is specified, the value must be between 0 and 65535. Other than this, the value is not checked as being valid and the corresponding code page is not checked as being installed.

For more information on file tagging, see *z/OS UNIX System Services Planning*.

text Specifies that each untagged file is implicitly marked as containing pure text data that can be converted.

notext Specifies that none of the untagged files in the file system are automatically converted during file reading and writing.

Examples

1. The output of **mount –q** can be used for the input of **unmount**. For example:

   ```
   mount -q /ict/hfsfir
   ```

   can be used as input:

   ```
   unmount $(mount -q /ict/hfsdir)
   ```

2. To mount an HFS file system over **/u/wjs** with a sync interval of 120 seconds:

   ```
   mount -f omvs.hfs.user.wjs -o 'SYNC(120)' /u/wjs
   ```

3. To display a list of pathnames for all mountpoints under **/u**:

   ```
   mount -q /u
   ```

Usage Notes®

1. Systems exploiting shared file system will have I/O to an OMVS couple data set. Because of these I/O operations to the CDS, each mount request requires additional system overhead. You will need to consider the affect that this will have on your recovery time if a large number of mounts are required on any system participating in shared file system.
2. The **–a unmount** is not available to automounted file systems.
3. The **–a no** specification will only be accepted on z/OS V1R3 systems and later.

File System Recovery and mount

File system recovery in a shared file system environment takes into consideration file system specifications such as **–a yes|no|unmount** and whether or not the file system is mounted read-only or read/write.

Generally, when an owning system fails, ownership over its **–a yes** mounted file system is moved to another system and the file is usable. However, if a file system is mounted read/write and the owning system fails, then all file system operations for files in that file system will fail. This is because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered. (The BPX1CLO and BPX1OPN callable services are discussed in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.)

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be submitted again. Otherwise, the file system is usable.

In some situations, even though a file system is mounted with the **–a yes** option, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes ″unowned″ (the system will issue message BPXF213E when this occurs). This is true if the file system is mounted either read/write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that are owned by another system are still usable.

However, all file operations for the unowned file system will fail until a new owner is established. The shared file system support will continue to attempt recovery of **–a yes** mounted file systems on all systems in the sysplex that are enabled for shared file system. Should a subsequent recovery attempt succeed, the file system transitions from the unowned to the active state.

Applications using files in unowned file systems will need to close (BPX1CLO) those files and re-open (BPX1OPN) them after the file system is recovered.

File systems that are mounted with the **–a no** option will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Note that since the file system still exists in the file system hierarchy, the file system mount point is still in use.

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

File systems associated with a 'never move' PFS will be unmounted during dead system recovery. For example, TFS is a 'never move' PFS and will be unmounted, as well as any file systems mounted on it, when the owning system leaves the sysplex.

As stated in "Usage Notes®" on page 118, **–a unmount** is not available to automounted file systems. However, during dead system recovery processing for an automounted file system (whose owner is the dead system), the file system will be unmounted if it is not being referenced by any other system in the sysplex.

For more information on mounts and the AUTOMOVE and NOAUTOMOVE parameters, see "mount — Logically mount a file system" on page 116.

Exit Values

0 Successful completion

Related Information

chmount, **unmount**

MOUNT — Logically mount a file system

Format

```
MOUNT FILESYSTEM(file_system_name)
      MOUNTPOINT(pathname)
      TYPE(file_system_type)
      MODE(RDWR|READ)
      PARM(parameter_string)
      TAG(NOTEXT|TEXT,ccsid)
      SETUID|NOSETUID
      WAIT|NOWAIT
      SECURITY|NOSECURITY
      SYSNAME (sysname)
      AUTOMOVE|AUTOMOVE(indicator,sysname1,sysname2,...,sysnameN)|
      NOAUTOMOVE|UNMOUNT
```

Above, *indicator* is either INCLUDE or EXCLUDE, which can also be abbreviated as I or E.

Description

For hierarchical file systems, you can use the MOUNT command to logically mount, or add, a mountable file system to the file system hierarchy. You can unmount any mounted file system using the UNMOUNT command.

Note: A mount user must have UID (0) or at least have READ access to the BPX.SUPERUSER FACILITY class.

filesystem(file_system_name)
Specifies the name of the file system to be added to the file system hierarchy.

file_system_name
For the hierarchical file system (HFS), this is the fully qualified name of the MVS HFS data set that contains the file system. It cannot be a partitioned data set member.

The file system name specified must be unique among previously mounted file systems. The file system name supplied is changed to all uppercase characters. You can enclose it in single quotes, but they are not required.

If file system('"file_system_name"') is specified, the file system name will not be translated to uppercase.

MOUNTPOINT(pathname)
Specifies the pathname of the mount point directory, the place within the file hierarchy where the file system is to be mounted. This operand is required.

pathname
Specifies the mount point pathname. The pathname must be enclosed in single quotes. The name can be a relative pathname or an absolute pathname. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually

specify an absolute pathname. It can be up to 1023 characters long. Pathnames are case-sensitive, so enter the pathname exactly as it is to appear.

Rules:

1. The mount point must be a directory. ***Any files in that directory are inaccessible while the file system is mounted.***
2. Only one file system can be mounted to a mount point at any time.

TYPE(file_system_type)
Specifies the type of file system that will perform the logical mount request. The system converts the TYPE operand value to uppercase letters. This operand is required.

file_system_type
This name must match the TYPE operand of the FILESYSTYPE statement that activates this physical file system in the BPXPRMxx parmlib member. The file_system_type value can be up to 8 characters long.

MODE(RDWR|READ)
Specifies the type of access the file system is to be opened for.

RDWR
Specifies that the file system is to be mounted for read and write access. RDWR is the default if MODE is omitted.

READ
Specifies that the file system is to be mounted for read-only access.

The HFS allows a file system that is mounted using the MODE(READ) option to be shared as read-only with other systems that share the same DASD.

PARM(parameter_string)
Specifies a parameter string to be passed to the file system type. The format and content are specified by the physical file system that is to perform the logical mount. You can specify lowercase or uppercase characters. Enclose the string in single quotes.

parameter_string
Specifies a parameter string value that can be up to 1024 characters long. The parameter string must be enclosed in single quotes; it is case-sensitive.

For an HFS file system, the following can be specified:
`PARM('SYNC(t),NOWRITEPROTECT')`

SYNC(t)
`t` is a numeric value that specifies the number of seconds that should be used to override the sync interval default for this file system during a specific mount. If SYNC is not specified at mount time, then the sync interval default value will be used (a value of 60 seconds). The same rules apply to the argument to the SYNC keyword at mount time as apply to the argument of the SYNCDEFAULT keyword at HFS initialization time. For more information on the SYNCDEFAULT keyword, see *z/OS UNIX System Services Planning*.

NOWRITEPROTECT
The HFS has a Write Protection mechanism that adds some overhead to HFS processing. This overhead can be avoided by turning off the write protection by specifying NOWRITEPROTECT in the PARM field of the MOUNT command.

NOSPARSE | NOSPARSE(DUMP)
Will cause HFS to create a dump when it either attempts to read metadata from disk for a file and detects that the subject file is sparse or if an application attempts to write to a page beyond the end of the file causing the file to become sparse. Only one dump will be created for each of the possible reason codes while a file system is mounted.

NOSPARSE(LOGREC)
Will cause HFS to write a LOGREC record instead of creating a dump for the same conditions as for the Dump case.

TAG(NOTEXT|TEXT,*ccsid*)
Specifies whether the file tags for untagged files in the mounted file system are implicitly set. File tagging controls the ability to convert a file's data during file reading and writing. Implicit, in this case, means that the tag is not permanently stored with the file. Rather, the tag is associated with the file during reading or writing, or when stat() type functions are issued. Either TEXT or NOTEXT, and ccsid must be specified when TAG is specified.

Note: When the file system is unmounted, the tags are lost.

NOTEXT
Specifies that none of the untagged files in the file system are automatically converted during file reading and writing.

TEXT
Specifies that each untagged file is implicitly marked as containing pure text data that can be converted.

ccsid
Identifies the coded character set identifier to be implicitly set for the untagged file. *ccsid* is specified as a decimal value from 0 to 65535. However, when TEXT is specified, the value must be between 0 and 65535. Other than this, the value is not checked as being valid and the corresponding code page is not checked as being installed.

SETUID|NOSETUID
Specifies whether the SETUID and SETGID mode bits on executables in this file system are respected. Also determines whether the APF extended attribute or the Program Control extended attribute is honored.

SETUID
Specifies that the SETUID and SETGID mode bits be respected when a program in this file system is run. SETUID is the default.

NOSETUID
Specifies that the SETUID and SETGID mode bits not be respected when a program in this file system is run. The program runs as though the SETUID and SETGID mode bits were not set. Also, if you specify the NOSETUID option on MOUNT, the APF extended attribute and the Program Control extended attribute are not honored.

WAIT|NOWAIT
Specifies whether to wait for an asynchronous mount to complete before returning.

WAIT
Specifies that MOUNT is to wait for the mount to complete before returning. WAIT is the default.

NOWAIT
Specifies that if the file system cannot be mounted immediately (for example, a network mount must be done), then the command will return with a return code indicating that an asynchronous mount is in progress.

SECURITY|NOSECURITY
Specifies whether security checks are to be enforced for files in this file system.

Note: When an HFS is mounted with the NOSECURITY option enabled, any new files or directories that are created will be assigned an owner of UID 0, no matter what UID issued the request.

SECURITY
Specifies that normal security checking will be done. SECURITY is the default.

NOSECURITY
Specifies that security checking will not be enforced for files in this file system. A user may access or change any file or directory in any way.

Security auditing will still be performed if the installation is auditing successes.

The SETUID, SETGID, APF, and Program Control attributes may be turned on in files in this file system, but they will not be honored while it is mounted with NOSECURITY.

SYSNAME (sysname)
For systems participating in shared file system, SYSNAME specifies the particular system on which a mount should be performed. This system will then become the owner of the file system mounted. This system must be IPLed with SYSPLEX(YES). IBM recommends that you specify SYSNAME(&SYSNAME.) or omit the SYSNAME parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

sysname
sysname is a 1–8 alphanumeric name of a system participating in shared file system.

AUTOMOVE | AUTOMOVE(indicator,sysname1,...,sysnameN) | NOAUTOMOVE | UNMOUNT
These parameters only apply in a sysplex where systems are exploiting the shared file system capability. They specify what is to happen to the ownership of a file system when a shutdown, PFS termination, dead system takeover, or file system move occurs. The default setting is AUTOMOVE, where the file system will be randomly moved to another system (no system list used).

indicator is either INCLUDE or EXCLUDE, which can also be abbreviated as I or E.

AUTOMOVE
AUTOMOVE indicates that ownership of the file system can be automatically moved to another system participating in shared file system. AUTOMOVE is the default.

AUTOMOVE(INCLUDE,sysname1,sysname2,...,sysnameN) or AUTOMOVE(I,sysname1,sysname2,...,sysnameN)
The INCLUDE indicator with a system list provides an ordered list of systems to which the file system's ownership could be moved. *sysnameN* may be a system name or an asterisk (*). The asterisk acts as a wildcard to

allow ownership to move to any other participating system and is only permitted in place of a system name as the last entry of a system list.

Note: Use of the asterisk is not supported prior to z/OS Version 1 Release 6.

AUTOMOVE(EXCLUDE,sysname1,sysname2,...,sysnameN) or AUTOMOVE(E,sysname1,sysname2,...,sysnameN)
The EXCLUDE indicator with a system list provides a list of systems to which the file system's ownership should not be moved.

NOAUTOMOVE
NOAUTOMOVE prevents movement of file system's ownership in some situations.

UNMOUNT
UNMOUNT allows the file system to be unmounted in some situations.

Guidelines:

1. You should define your version and sysplex root file systems as AUTOMOVE, and define your system-specific file systems as UNMOUNT.
2. Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system underneath it as AUTOMOVE; in this case, the file system defined as AUTOMOVE will not be recovered after a system failure until the failing system is restarted.

For more information about shared file systems and the associated version and sysplex root file systems, as well as details about the behavior of the AUTOMOVE options, see *z/OS UNIX System Services Planning*.

Usage Notes

1. The directory **/samples** contain sample MOUNT commands (called **mountx**).
2. When the mount is done asynchronously (NOWAIT was specified and return code 4 was returned), you can determine if the mount has completed with one of the following:
   - The **df** shell command
   - The **DISPLAY OMVS,F** operator command (see *z/OS MVS System Commands*)
   - The MOUNT table option on the File Systems pulldown in the ISPF Shell (accessed by the ISHELL command)
3. In order to mount a file system as the system root file system, the caller must be a superuser. Also, a file system can only be mounted as the system root file system if the root file system was previously unmounted.
4. If you have previously unmounted the root file system, a 'dummy file system' or SYSROOT will be displayed as the current root file system. During the time when SYSROOT is displayed as the root, any operation that requires a valid file system will fail. When you subsequently mount a new root file system on mountpoint **/**, that new file system will replace SYSROOT. When a new root file system has been mounted, you should terminate any current dubbed users or issue a **chdir** using a full pathname to the appropriate directory. This way, the users can access the new root file system. Otherwise, an error will occur when a request is made requiring a valid file system.
5. Systems exploiting shared file system will have I/O to an OMVS couple data set. Because of these I/O operations to the CDS, each mount request requires additional system overhead. You will need to consider the affect that this will

have on your recovery time if a large number of mounts are required on any system participating in shared file system.

6. The TAG parameter is intended for file systems that don't support storing the file tag, such as NFS remote file systems.
7. Do not use the TAG parameter simultaneously with the NFS Client Xlate option. If you do, the mount will fail.
8. The UNMOUNT keyword is not available to automounted file systems.
9. The UNMOUNT specification will only be accepted on z/OS V1R3 systems and later.

File System Recovery and TSO MOUNT

File system recovery in a shared file system environment takes into consideration file system specifications such as AUTOMOVE | NOAUTOMOVE | UNMOUNT, and whether or not the file system is mounted read-only or read/write.

Generally, when an owning system fails, ownership over its AUTOMOVE mounted file system(s) is moved to another system and the file is usable. However, if a file system is mounted read/write and the owning system fails, then all file system operations for files in that file system will fail. This is because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and re-opened (BPX1OPN) when the file system is recovered. (The BPX1CLO and BPX1OPN callable services are discussed in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.)

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be re-attempted. Otherwise, the file system is usable.

In some situations, even though a file system is mounted AUTOMOVE, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes unowned (the system will issue message BPXF213E when this occurs). This is true if the file system is mounted either read/write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that are owned by another system are still usable.

However, all file operations for the unowned file system will fail until a new owner is established. The shared file system support will continue to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for shared file system. Should a subsequent recovery attempt succeed, the file system transitions from the unowned to the active state.

Applications using files in unowned file systems will need to close (BPX1CLO) those files and re-open (BPX1OPN) them after the file system is recovered.

File systems that are mounted NOAUTOMOVE will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Note that since the file system still exists in the file system hierarchy, the file system mount point is still in use.

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

File systems associated with a 'never move' PFS will be unmounted during dead system recovery. For example, TFS is a 'never move' PFS and will be unmounted, as well as any file systems mounted on it, when the owning system leaves the sysplex.

As stated in "Usage Notes" on page 124, the UNMOUNT keyword is not available to automounted file systems. However, during dead system recovery processing for an automounted file system (whose owner is the dead system), the file system will be unmounted if it is not being referenced by any other system in the sysplex.

Return Codes

0 Processing successful.
4 Processing incomplete. An asynchronous mount is in progress.
12 Processing unsuccessful. An error message has been issued.

Examples

1. To mount the HFS data set HFS.WORKDS on the directory **/u/openuser**, enter:

   ```
   MOUNT filesystem('HFS.WORKDS') MOUNTPOINT('/u/openuser') TYPE(HFS)
   ```

2. The following example mounts the HFS directory **/u/shared_data**, which resides on the remote host named **mvshost1**, onto the local directory **/u/jones/mnt**. The command may return before the mount is complete, allowing the mount to be processed in parallel with other work. The SETUID and SETGID bits are honored on any executable programs:

   ```
   MOUNT filesystem('MVSHOST1.SHARE.DATA') MOUNTPOINT('/u/jones/mnt')
         TYPE(NFSC) PARM('mvshost1:/hfs/u/shared_data') NOWAIT SETUID
   ```

3. Examples for using the TAG parameter are:

   TAG(TEXT,819) identifies text files containing ASCII (ISO-8859-1) data.

   TAG(TEXT,1047) identifies text files containing EBCDIC (ISO-1047) data.

   TAG(NOTEXT,65535) tags files as containing binary or unknown data.

   TAG(NOTEXT,0) is the equivalent of not specifying the TAG parameter at all.

   TAG(NOTEXT,273) tags files with the German code set (ISO-273), but is ineligible for automatic conversion.

Chapter 5. Changes for MVS System Commands

SETOMVS Command

Use the SETOMVS command to change dynamically the options that z/OS UNIX System Services currently is using. These options are originally set in the BPXPRMxx parmlib member during initial program load (IPL). For more information on the BPXPRMxx parmlib member, see *z/OS UNIX System Services Planning*.

Changes to all of the system-wide limits take effect immediately. When a process limit is updated, all processes that are using the system-wide process limit have their limits updated. All process limit changes take effect immediately except those processes with a user-defined process limit (defined in the OMVS segment or set with a SETOMVS PID= command). Exceptions are MAXASSIZE and MAXCPUTIME, which are not changed for active processes.

Note: If a process-level limit is lowered with the SETOMVS command, some processes may immediately hit 100% usage. Depending on the process limit specified and what the process is doing, this could cause some processes to fail.

Syntax

The complete syntax for the SETOMVS command is:

SETOMVS	SETOMVS EXTENSIONS (sysplex exclusive)
SETOMVS [AUTHPGMLIST='authprogramlist'|NONE] [,FORKCOPY=(COPY|COW)] [,IPCSEMNIDS=ipcsemnids] [,IPCSEMNOPS=ipcsemnops] [,IPCSEMNSEMS=ipcsemnsems] [,IPCMSGQBYTES=ipcmsgqbytes] [,IPCMSGNIDS=ipcmsgnids] [,IPCSHMMPAGES=ipcshmmpages] [,IPCSHMNIDS=ipcshmnids] [,IPCSHMNSEGS=ipcshmnsegs] [,IPCSHMSPAGES=ipcshmspages] [,IPCMSGQMNUM=ipcmsgqmnum] [,LIMMSG=[NONE|SYSTEM|ALL]] [,MAXASSIZE=maxassize] [,MAXCORESIZE=maxcoresize] [,MAXCPUTIME=maxcputime] [,MAXFILEPROC=maxfileproc] [,MAXFILESIZE=(maxfilesize|NOLIMIT)] [,MAXMMAPAREA=maxmmaparea] [,MAXPROCSYS=maxprocsys] [,MAXPROCUSER=maxprocuser] [,MAXPTYS=maxptys] [,MAXSHAREPAGES=maxsharepages] [,MAXTHREADS=maxthreads] [,MAXTHREADTASKS=maxthreadtasks] [,MAXUIDS=maxuids] [,PID=pid,processlimitname=newvalue] [,PRIORITYGOAL=(n) | NONE] [,RESET=(xx)] [,STEPLIBLIST='stepliblist'] [,SUPERUSER=superuser] [,SYNTAXCHECK=(xx)] [,TTYGROUP=ttygroup] [,USERIDALIASTABLE='useridaliastable'] [,VERSION='string']	SETOMVS FILESYS ,FILESYSTEM=filesystem ,AUTOMOVE=YES|NO|UNMOUNT| indicator(sysname1 ,sysname2,...,sysnameN) or SETOMVS FILESYS ,FILESYSTEM=filesystem ,SYSNAME=sysname|* or SETOMVS FILESYS ,MOUNTPOINT=mountpoint ,AUTOMOVE=YES|NO|UNMOUNT| indicator(sysname1 ,sysname2,...,sysnameN) or SETOMVS FILESYS ,MOUNTPOINT=mountpoint ,AUTOMOVE=YES|NO|UNMOUNT| indicator(sysname1 ,sysname2,...,sysnameN) or SETOMVS FILESYS ,MOUNTPOINT=mountpoint ,SYSNAME=sysname|* or SETOMVS FILESYS ,FROMSYS=sysname ,SYSNAME=sysname|* **Notes:** 1. **FILESYSTEM**, **FROMSYS**, and **MOUNTPOINT** are mutually exclusive parameters. When you specify **FILESYS**, you must supply one of these three parameters. 2. **SETOMVS RESET=(xx)** has been changed to allow SETOMVS RESET=xx as well as SETOMVS RESET=(xx). The parentheses are now optional.

Rather than defining parameter limit values in their full decimal or hexadecimal form, you can use the following 1–character multiplier (denomination values) suffix to specify them. The system also uses this value in displays when it returns responses to respective D OMVS commands.

Notes:

1. Only those SETOMVS parameters that support this ″C″ suffix specifically note that support and refer to Table 25 on page 143.
2. Values that **contain** a multiplier are limited to 8 digits (nnnnnnnnC) and those values are limited to X'00FF FFFF' (16 777 215 decimal). Limits that support values above the bar have a range of 1M-16383P. However, do not exceed a parameter-specific maximum value.
3. Values that **do not contain** a multiplier are limited to X'7FFF FFFF' (2 147 483 647 decimal).

Table 23. 1–Character Parameter Limit Multipliers

Denomination Value	1–Character Abbreviation	Bytes
null	n/a	1
Kilo	**K**	1,024
Mega	**M**	1,048,576

Table 23. 1–Character Parameter Limit Multipliers (continued)

Giga	**G**	1,073,741,824
Tera	**T**	1,099,511,627,776
Peta	**P**	1,125,899,906,842,624

Parameters

AUTOMOVE =YES|NO|UNMOUNT|indicator(sysname1,sysname2,...,sysnameN), FILESYS=filesys, FILESYSTEM=filesystem, FROMSYS=sysname, MOUNTPOINT=mountpoint, SYSNAME=sysname|*, and VERSION='nnnn' are parameters that are used in a sysplex environment where systems are exploiting shared file system. For more information on sharing file systems in a sysplex and the behavior of the AUTOMOVE options, see *z/OS UNIX System Services Planning*.

The parameters are:

AUTOMOVE=YES|NO|UNMOUNT|indicator(sysname1,sysname2,...,sysnameN)
AUTOMOVE only applies in a sysplex where systems are participating in shared file system. These parameters indicate what happens to the ownership of the file system when a shutdown, PFS termination, dead system takeover, or file system move occurs.

AUTOMOVE=YES allows the system to automatically move logical ownership of the file system as needed. AUTOMOVE=YES is the default; you can specify it as AUTOMOVE.

AUTOMOVE=NO prevents ownership movement in some situations.

AUTOMOVE=UNMOUNT unmounts the file system in some situations.

AUTOMOVE=indicator(sysname1,sysname2,...,sysnameN) specifies a list of systems to which the file system's ownership should or should not be moved when ownership of the file system changes.

- If *indicator* is specified as INCLUDE (or I), the system list must provide a comma-delimited, priority-ordered list of systems to which ownership of the file system can be moved. For example, AUTOMOVE=INCLUDE(SYS1, SYS4, SYS9). You can specify an asterisk (*) for the last (or only) system name to indicate any active system. For example, AUTOMOVE=INCLUDE(SYS1, SYS4, *).

  Note: Do not use an asterisk in a mixed sysplex environment where any system is not at z/OS Version 1 Release 6 or later, as doing so will produce unpredictable results. The asterisk is not supported prior to z/OS Version 1 Release 6.
- If *indicator* is specified as EXCLUDE (or E), the system list must provide a comma-delimited list of systems to which the file system must not be moved. For example, AUTOMOVE=EXCLUDE(SYS3, SYS5, SYS7).

Restriction: The AUTOMOVE parameter is not permitted when using SETOMVS to move a file system.

Guideline: To ensure that the root file system is always available, use the default AUTOMOVE value (AUTOMOVE=YES).

For more information about the behavior of the AUTOMOVE option, see *z/OS UNIX System Services Planning*.

FILESYS=filesys
In a sysplex environment, this parameter alerts the parser that commands that change mount attributes are to follow.

For examples on the use of this parameter when making move or change requests, see *z/OS UNIX System Services Planning*.

FILESYSTEM=filesystem
In a sysplex environment, **FILESYSTEM** is the 44 character alphanumeric field that denotes the name of the filesystem to be changed or moved. This filesystem name must be in the following form: 'OMVS.USER.JOE'.

Note: The filesystem name must be in quotes, and mixed-case filesystem names are supported.

FILESYSTEM, **MOUNTPOINT**, and **FROMSYS** are mutually exclusive parameters.

For examples on the use of this parameter when making move or change requests, see *z/OS UNIX System Services Planning*.

FROMSYS=sysname
In a sysplex environment, this parameter indicates the system where all the filesystems will be moved from. The filesystems will be moved to the system identified by the **sysname** keyword. **FILESYSTEM**, **MOUNTPOINT**, and **FROMSYS** are mutually exclusive parameters.

MOUNTPOINT=mountpoint
In a sysplex environment, MOUNTPOINT is the mountpoint specification. For example:

```
'/usr/d1'
```

It is case sensitive. This is the mountpoint where the filesystem is mounted. If specified, the filesystem associated with this mountpoint will be moved or changed. **FILESYSTEM**, **MOUNTPOINT**, and **FROMSYS** are mutually exclusive parameters.

For examples on the use of this parameter when making move or change requests, see *z/OS UNIX System Services Planning*.

AUTHPGMLIST='authprogramlist'|NONE
Points to an hfs file containing a list of pathnames, MVS program names, or both that allow an additional level of authorization for program-controlled or for apf-authorized programs. See *z/OS UNIX System Services Planning* for information on constructing this file. The default is NONE.

FORKCOPY = COPY | COW
Specifies how user storage is copied from the parent process to the child process during a **fork()** system call.

If you specify **FORKCOPY=COW**, all **fork()** calls are processed in copy-on-write (COW) mode if the suppression-on-protection hardware feature is available. Before the storage is modified, both the parent and child processes refer to the same view of the data. The parent storage is copied to the child as soon as storage is modified, either by the parent or the child.

Using copy-on-write causes the system to use the extended system queue area (ESQA) to manage page sharing.

If you specify **FORKCOPY=COPY**, **fork()** immediately copies the parent storage to the child, regardless of whether the suppression-on-protection feature is available. Use this option to avoid any additional ESQA use in support of **fork()**.

Follow these guidelines:

- If the run-time library is in the link pack area, specify **FORKCOPY=COPY**.
- If the run-time library is not in the link pack area, specify **FORKCOPY=COW**.

If you do not specify **FORKCOPY**, the default is **FORKCOPY=COW**.

IPCSEMNIDS = ipcsemnids
Specifies the maximum number of unique semaphore sets in the system. The range is from 1 to 20 000. The default is 500.

IPCSEMNOPS = ipcsemnops
Specifies the maximum number of operations for each semaphore operation call. The range is from 0 to 32 767. The default is 25. This is a system-wide limit.

IPCSEMNSEMS = ipcsemnsems
Specifies the maximum number of semaphores for each semaphore set. The range is from 0 to 32 767. The default is 25.

IPCMSGQBYTES = ipcmsgqbytes
Specifies the maximum number of bytes in a single message queue. The range is from 0 to 1 048 576. The default is 262 144.

IPCMSGNIDS = ipcmsgnids
Specifies the maximum number of unique message queues in the system. The range is from 1 to 20 000. The default is 500.

IPCSHMMPAGES = ipcshmmpages
Specifies the maximum number of pages for a shared memory segment. The range is from 1 to 4P. The default is 25600.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 128, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

IPCSHMNIDS = ipcshmnids
Specifies the maximum number of unique shared memory segments in the system. The range is from 1 to 20 000. The default is 500.

IPCSHMNSEGS = ipcshmnsegs
Specifies the maximum number of shared memory segments attached for each address space. The range is from 0 to 1 000. The default is 10.

IPCSHMSPAGES = ipcshmspages
Specifies the maximum number of pages for shared memory segments in the system. The range is from 0 to 2 621 440. The default is 262 144.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 128, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

IPCMSGQMNUM = ipcmsqgmnum
Specifies the maximum number of messages for each message queue in the system. The range is from 0 to 20 000. The default is 10 000.

LIMMSG=(NONE|SYSTEM|ALL)
Specifies how console messages that indicate when system parmlib limits are reaching critical levels are to be displayed:

NONE No console messages are to be displayed when any of the parmlib limits have been reached.

SYSTEM
Console messages are to be displayed for all processes that reach system limits. In addition, messages are to be displayed for each process limit of a process if:
- The process limit or limits are defined in the OMVS segment of the owning User ID
- The process limit or limits have been changed with a SETOMVS PID=*pid*,*process_limit*

ALL Console messages are to be displayed for the system limits and for the process limits, regardless of which process reaches a process limit.

Default: NONE

MAXASSIZE = maxassize
Specifies the RLIMIT_AS hard limit resource value that processes receive when they are dubbed a process. RLIMIT_AS indicates the address space region size. The soft limit is obtained from MVS. If the soft limit value from MVS is greater than the MAXASSIZE value, the hard limit is set to the soft limit.

This value is also used when processes are initiated by a daemon process using an **exec** after **setuid()**. In this case, both the RLIMIT_AS hard and soft limit values are set to the MAXASSIZE value.

Refer to the description of **setrlimit()** in *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about RLIMIT_AS.

The range is from 10 485 760 (10MB) to 2 147 483 647 ; the default is 41 943 040 (40MB).

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 128, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

MAXCORESIZE = maxcoresize
Specifies the RLIMIT_CORE soft and hard limit resource values that processes receive when they are dubbed a process. RLIMIT_CORE indicates the maximum core dump file size (in bytes) that a process can create. Also, it specifies the limit when they are initiated by a daemon proces using an **exec** after **setuid()**.

Refer to the description of **setrlimit()** in *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about RLIMIT_CORE.

The range is from 0 to 2 147 483 647; the default is 4 194 304 (4MB).

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 128, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

MAXCPUTIME = maxcputime

Specifies the RLIMIT_CPU hard limit resource values that processes receive when they are dubbed a process. RLIMIT_CPU indicates the CPU time that a process is allowed to use, in seconds. The soft limit is obtained from MVS. If the soft limit value from MVS is greater than the MAXCPUTIME value, the hard limit is set to the soft limit. This value is also used when processes are initated by a daemon process using an **exec** after **setuid()**. In this case, both the RLIMIT_CPU hard and soft limit values are set to the MAXCPUTIME value.

Refer to the description of **setrlimit()** in *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about RLIMIT_CPU.

The range is from 7 to 2 147 483 647. The default is 1 000.

Specifying a value of 2 147 483 647 indicates unlimited CPU time.

MAXFILEPROC = maxfileproc

Specifies the maximum number of files that a single user is allowed to have concurrently active or allocated. The range is 3 to 131 071.

MAXFILESIZE = (maxfilesize | NOLIMIT)

Specifies the RLIMIT_FSIZE soft and hard limit resource values that processes receive when they are dubbed a process. RLIMIT_FSIZE indicates the maximum file size (in 4KB increments) that a process can create. Also, it specifies the limit when they are initiated by a daemon process using an **exec** after **setuid()**.

The range is from 0 to 524 228. If you specify 0, no files will be created by the process. Omitting this statement or specifying NOLIMIT indicates an unlimited file size.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 128, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

MAXMMAPAREA = maxmmaparea

Specifies the maximum amount of data space storage (in pages) that can be allocated for memory mappings of HFS files. Storage is not allocated until memory mappings are active.

The range is from 1 to 16 777 216. The default is 4 096.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 128, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

MAXPROCSYS = maxprocsys
Specifies the maximum number of processes that z/OS UNIX System Services will allow to be active at the same time. The range is 5 to 32 767; the default and the value in BPXPRMXX is 200.

MAXPROCUSER = maxprocuser
Specifies the maximum number of processes that a single OMVS user ID (UID) is allowed to have active at the same time, regardless of how the process became a UNIX System Services process. The range is 3 to 32 767;

MAXPTYS = maxptys
Specifies the maximum number of pseudo-TTY (pseudoterminal) sessions that can be active at the same time. The range is 1 to 10 000; the default and the value in BPXPRMXX is 256.

MAXPTYS lets you manage the number of interactive shell sessions. When you specify this value, each interactive session requires one pseudo-TTY pair. You should avoid specifying an arbitrarily high value for MAXPTYS. However, because each interactive user may have more than one session, we recommend that you allow 4 pseudo-TTY pairs for each user (MAXUIDS * 4). The MAXPTYS value influences the number of pseudo-TTY pairs that can be defined in the file system.

MAXSHAREPAGES = maxsharepages
Specifies the maximum number of shared storage pages that can be concurrently in use by UNIX System Services functions. This can be used to control the amount of ESQA consumed, since the shared storage pages cause the consumption of ESQA storage.

The range is from 0 to 32 768 000. The default is 131 072 pages.

Notes:

1. You can set a denomination (or multiplier) value when defining the MAXSHAREPAGES value. The suffix, ″C″ can have a 1-character value as presented in Table 25 on page 128, but must not exceed the parameter-specific upper limit. MVS retains the denomination value and uses it within a subsequent D OMVS command
2. Use care when you adjust MAXSHAREPAGES on an active system. Dynamically decreasing the number of pages available to EQSA while there is a workload can cause errors, because the EQSA limit can be suddenly reached when the MAXSHAREPAGES limit is no longer as large. As a result, shared programs are not able to be loaded, and new forks are not able to be created. This situation can exist until the workload adjusts to the new lower limit.

MAXTHREADS = maxthreads
Specifies the maximum number of pthread_created threads, including those running, queued, and exited but not detached, that a single process can have currently active. Specifying a value of 0 prevents applications from using pthread_create. The range is 0 to 100 000; the default and the value in BPXPRMXX is 200.

MAXTHREADTASKS = maxthreadtasks
Specifies the maximum number of MVS tasks created with pthread_create (BPX1PTC) that a single user may have concurrently active in a process. The range is 1 to 32 768; the default and the value in BPXPRMXX is 50.

MAXTHREADTASKS lets you limit the amount of system resources available to a single user process.

- The minimum value of 1 prevents a process from performing any pthread_creates.
- A high MAXTHREADTASKS value may affect storage and performance. Each task requires additional storage for:
  - The control blocks built by the z/OS UNIX kernel
  - The control blocks and data areas required by the runtime library
  - System control blocks such as the TCB and RB

Individual processes can alter these limits dynamically.

MAXUIDS = maxuids
Specifies the maximum number of unique OMVS user IDs (UIDs) that can use UNIX System Services at the same time. The UIDs are for interactive users or for programs that requested UNIX System Services. The range is 1 to 32 767; the default and the value in BPXPRMXX is 200.

MAXUIDS lets you limit the number of active UIDs. Select a MAXUIDS by considering:

- Each UNIX System Services user is likely to run with 3 or more concurrent processes. Therefore, UNIX System Services users require more system resources than typical TSO/E users.
- If the MAXUIDS value is too high relative to the MAXPROCSYS value, too many users can invoke the shell. All users may be affected, because forks may begin to fail.

  For example, if your installation can support 400® concurrent processes — MAXPROCSYS(400) — and each UID needs an average of 4 processes, then the system can support 100 users. For this operating system, specify MAXUIDS(100).

In assigning a value to MAXUIDS, consider if the security administrator assigned the same OMVS UID to more than one TSO/E user ID.

MEMLIMIT = maxmemlimit
Specifies the maximum amount (*maxmemlimit*) of allocated, non-shared, 1–megabyte storage segments above the bar allowed for the address space. Both the hard and soft RLMIT_MEMLIMIT values are set to this value, and the address space *memlimit* is modified to reflect his value.

Note: You can set a denomination (or multiplier) value when defining this value (nnnnnnnnC), where nnnnnnnn ranges from 1M — 16383P (noting values are rounded up) and C can have a 1–character value as presented in Table 25 on page 128. Also, be aware that SMF set override limits to the values you set here.

MVS retains the denomination value and uses it within a subsequent D OMVS command

PID=pid,processlimitname=value
Dynamically changes a process-level limit for the process represented by *pid*.

PRIORITYGOAL = (n) | NONE

Specify from 1 to 40 service classes. These classes can be from 1 to 8 characters. If you do not specify this statement, or if you specify NONE, no array is created for it. All service classes specified on the PRIORITYGOAL option must also be specified in your workload manager service policy.

Generally, we do not recommend that you set PRIORITYGOAL.

RESET = (xx)

Specifies the parmlib member containing parameters to apply immediately to the running z/OS UNIX System Services environment. The variable specifies the character suffix of the BPXPRMxx member to use to change the environment. It can be any properly constructed BPXPRMxx member. This parameter accepts only the single keyword and parmfile specification. It does not accept additional keywords separated by commas.

The SETOMVS RESET command is similar to the SET™ OMVS command. The following table shows the acceptable parameters for each.

Notes:

1. **SETOMVS RESET** accepts only a single parameter; **SET OMVS** accepts more than one parameter.
2. **SETOMVS RESET=(xx)** has been changed to allow **SETOMVS RESET=xx** as well as **SETOMVS RESET=(xx)**. The parentheses are now optional.
3. **SETOMVS RESET=(xx)** redefines a PFS; it does not start it. Do not use **SETOMVS RESET=(xx)** unless absolutely necessary.

For more detailed information about the RESET parameter see *z/OS UNIX System Services Planning*.

Table 24. Acceptable Parameter Statements and Their Applicability

Parameter Statement	SET OMVS= (xx, yy, ...)	SETOMVS RESET= (xx)
AUTOMOVE	No	No
CTRACE	No	No
FILESYS	No	No
FILESYSTEM	No	No
FILESYSTYPE	Yes	Yes
FORKCOPY	Yes	Yes
FROMSYS	No	No
IPCMSGNIDS	Yes	Yes
IPCMSGQBYTES	Yes	Yes
IPCMSGQMNUM	Yes	Yes
IPCSEMNIDS	Yes	Yes
IPCSEMNOPS	Yes	Yes
IPCSEMNSEMS	Yes	Yes
IPCSHMMPAGES	Yes	Yes
IPCSHMNIDS	Yes	Yes
IPCSHMNSEGS	Yes	Yes
IPCSHMSPAGES	Yes	Yes
MAXASSIZE	Yes	Yes
MAXCORESIZE	Yes	Yes

Table 24. Acceptable Parameter Statements and Their Applicability (continued)

Parameter Statement	SET OMVS= (xx, yy, ...)	SETOMVS RESET= (xx)
MAXCPUTIME	Yes	Yes
MAXFILEPROC	Yes	Yes
MAXFILESIZE	Yes	Yes
MAXMMAPAREA	Yes	Yes
MAXPROCSYS	Yes	Yes
MAXPROCUSER	Yes	Yes
MAXPTYS	Yes	Yes
MAXSHAREPAGES	Yes	Yes
MAXTHREADS	Yes	Yes
MAXTHREADTASKS	Yes	Yes
MAXUIDS	Yes	Yes
MEMLIMIT	Yes	Yes
MOUNT	Yes	Yes
MOUNTPOINT	No	No
NETWORK	Yes	Yes
PRIORITYGOAL	Yes	Yes
PRIORITYPG	Yes	Yes
ROOT	Yes	Yes
RUNOPTS	No	No
STARTUP_EXEC	No	No
STARTUP_PROC	No	No
STEPLIBLIST	Yes	Yes
SUBFILESYSTYPE	Yes	Yes
SUPERUSER	Yes	Yes
SYSCALL_COUNTS	Yes	Yes
SYSNAME	No	No
SYSPLEX	No	No
TTYGROUP	Yes	Yes
USERIDALIASTABLE	Yes	Yes
VERSION	Yes	Yes

STEPLIBLIST = 'stepliblist'
Specifies the path name of a hierarchical file system (HFS) file. This file is intended to contain a list of data sets that are sanctioned by the installation for use as step libraries during the running of set-user-ID and set-group-ID executable programs.

SUPERUSER = superuser
This statement specifies a superuser name. You can specify a 1-to-8-character name that conforms to restrictions for an OS/390 user ID. The user ID specified on SUPERUSER must be defined to the security product and should have a UID of 0 assigned to it. The user ID specified with **setuid()** is used when a daemon switches to an unknown identity with a UID of 0.

The default is SUPERUSER(BPXROOT).

SYNTAXCHECK=(xx)
Specifies that the operator wishes to check the syntax of the designated parmlib member. For example, to check the syntax of BPXPRMZ1 the operator enters:

```
SETOMVS SYNTAXCHECK=(Z1)
```

The system returns a message indicating either that the syntax is correct or that syntax errors were found and written into the hard copy log. This command parses the parmlib member in the same manner, and with the same messages as during IPL.

Note: **SYNTAXCHECK** checks syntax as well as the existence of HFS and zFS data sets specified in the catalog. Mount points are not verified.

SYSCALL_COUNTS = (YES | NO)
Specifies whether to accumulate syscall counts so that the RMF™ data gatherer can record this information. The default is NO.

If you specify YES, the path length for the most frequently used kernel system calls increases by more than 150 instructions.

SYSNAME=sysname|*
sysname is the 1-8 alphanumeric name of a system participating in shared file system. This system must be IPLed with SYSPLEX(YES). **sysname** specifies the particular system on which a mount should be performed. This system will then become the owner of the file system mounted. If ***(asterisk) is specified, it represents any other randomly selected system taking part in shared file system. The asterisk specification is not available with the **FROMSYS** parameter.

For examples of the use of this parameter when making move or change requests, see ″shared file system in a Sysplex″ in *z/OS UNIX System Services Planning*.

TTYGROUP = ttygroup
This specifies a 1-to-8-character name that must conform to the restrictions for an OS/390 group name. Slave pseudoterminals (ptys) and OCS rtys are given this group name when they are first opened. This group name should be defined to the security product and have a unique GID. No users should be connected to this group.

The name is used by certain **setgid()** programs, such as **talk** and **write**, when attempting to write to another user's pty or rty.

The default is TTYGROUP(TTY).

USERIDALIASTABLE = 'useridaliastable'
Enables installations to associate alias names with MVS user IDs and group names. If specified, the alias names are used in z/OS UNIX System Services processing for the user IDs and group names listed in the table.

Specifying USERIDALIASTABLE causes performance to degrade slightly. The more names that you define, the greater the performance degradation. Installations are encouraged to continue using uppercase-only user IDs and group names.

The USERIDALIASTABLE statement specifies the pathname of a hierarchical file system (HFS) file. This file is intended to contain a list of MVS user IDs and group names with their associated alias names.

VERSION = 'nnnn'

The VERSION statement applies only to systems that are exploiting shared file systems. VERSION allows multiple releases and service levels of the binaries to coexist and participate in shared file systems. A directory with the value *nnnn* specified on VERSION is dynamically created at system initialization under the sysplex root that is used as a mount point for the version file system. This directory, however, is only dynamically created if the root file system for the sysplex is mounted read/write.

Note: *nnnn* is a case-sensitive character string no greater than 8 characters in length. It indicates a specific instance of the version HFS. The most appropriate values for *nnnn* are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer. For example, if the system is at V2R9, you can specify REL9 for VERSION.

When SYSPLEX(YES) is specified, you must also specify the VERSION parameter.

The VERSION value is substituted in the content of symbolic links that contain $VERSION. For scenarios describing the use of the version HFS, see ″shared file system in a Sysplex″ in *z/OS UNIX System Services Planning*.

When testing or changing to a new Maintenance Level (PTF), you can change the VERSION value dynamically by using the SETOMVS command:

```
SETOMVS VERSION='string'
```

You can also change the settings of this parameter via SET OMVS=(xx) and SETOMVS RESET=(xx) parmlib specifications.

Note: We do not recommend changing version dynamically if you have any users logged on or running applications; replacing the system files for these users may be disruptive.

Chapter 6. Changes for MVS Initialization and Tuning Reference

BPXPRMxx (z/OS UNIX System Services parameters)

BPXPRMxx contains the parameters that control the z/OS UNIX System Services (z/OS UNIX) environment and the file systems. IBM suggests that you have two BPXPRMxx parmlib members, one defining the values to be used for system setup and the other defining the file systems. This makes it easier to migrate from one release to another, especially when using the ServerPac method of installation.

To specify which BPXPRMxx parmlib member to start with, the operator can include OMVS=xx in the reply to the IPL message or OMVS=xx in the IEASYSxx parmlib member. The two alphanumeric characters, represented by xx, are appended to BPXPRM to form the name of the BPXPRMxx parmlib member.

If OMVS=xx is not specified in the reply to the IPL message or is not in the IEASYSxx member, or if OMVS=DEFAULT is specified, defaults are used for each parameter and the kernel services are started in minimum mode. For more information about running in minimum mode and full function mode, see *z/OS UNIX System Services Planning*. If the operator specifies OMVS=xx in the IPL reply to the message, it overrides the OMVS=xx specified in IEASYSxx.

Note: The START OMVS,OMVS=xx command is not valid when issued from the command console. OMVS=xx is not valid in parmlib COMMNDxx.

You can use multiple parmlib members to start OMVS. This is shown by the following reply to the IPL message:

```
R 0,CLPA,SYSP=R3,LNK=(R3,R2,L),OMVS=(AA,BB,CC)
```

The parmlib member BPXPRMCC would be processed first, followed by and overridden by BPXPRMBB, followed by and overridden by BPXPRMAA. This means that any parameter in BPXPRMAA has precedence over the same parameter in BPXPRMBB and BPXPRMCC.

For example, if you specify MAXFILESIZE in all three parmlib members, the value MAXFILESIZE in BPXPRMAA will be the value used to start kernel services.

You can also specify multiple BPXPRMxx parmlib members using the OMVS keyword in IEASYSxx. For example:

```
OMVS=(AA,BB,CC)
```

If MOUNT statements are specified in each parmlib member, the files are mounted in the following order: BPXPRMAA, BPXPRMBB, and BPXPRMCC.

To modify BPXPRMxx parmlib settings without re-IPLing, you can use the SETOMVS operator command, or you can dynamically change the BPXPRMxx parmlib members that are in effect by using the the SET OMVS operator command. See “Dynamically Changing the BPXPRMxx Values” in *z/OS UNIX System Services Planning* for more information. See *z/OS MVS System Commands* for more information about the SETOMVS and SET OMVS commands.

Syntax rules for BPXPRMxx

When customizing BPXPRMxx, the following rules apply:

- Statements that contain limiting keywords (like MAXUIDS, which limits the number of concurrent z/OS UNIX users), should not be duplicated in the same BPXPRMxx member. You can have duplicates of limiting keywords across BPXPRMxx members, but only the last occurrence is used. Resource defining keywords (like MOUNT, which specifies a file system that z/OS UNIX is to logically mount onto the root file system or another file system) are cumulative. Resource defining keywords can be duplicated in the same BPXPRMxx member. Each time you specify a resource defining keyword, its value is added to the previous values.
- If a statement that has a default is omitted, the default is used.
- Use columns 1 through 71 for data; columns 72 through 80 are ignored.
- Enter one or more statements on a line, or use several lines for one statement.
- Use blanks as delimiters. Multiple blanks are interpreted as a single blank. Blanks are allowed between parameters and values; for example, MAXPROCSYS(500) and MAXPROCSYS (500) are allowed and have the same meaning.
- Comments may appear in columns 1-71 and must begin with "/*" and end with "*/".
- Enter values in uppercase, lowercase, or mixed case. The system converts the input to uppercase, except for values enclosed in single quotes, which are processed without changing the case.
- Values that require single quotes and that are the only ones allowed to be in single quotes are:
  - STEPLIBLIST
  - USERIDALIASTABLE
  - FILESYSTEM in the ROOT and MOUNT statements
  - MOUNTPOINT in the MOUNT statement
  - PARM in the FILESYSTYPE, ROOT, MOUNT, and SUBFILESYSTYPE statements
  - RUNOPTS
  - VERSION
  - AUTHPGMLIST
- Enclose values in single quotes, using the following rules:
  - Two single quotes next to each other on the same line are considered as a single quote. For example, `John''s file` is considered to be `John's file`. One quote in column 71 and another in column 1 of the next line are *not* considered as a single quote. This input is treated as two strings or an error.
  - Because some values can be up to 1023 characters, a value can require multiple lines. Place one quote at the beginning of the value, stop the value in column 72 of each line, continue the value in column 1 of the next line, and complete the value with one quote. For example:

```
column                                                          column
1                                                                  71
|                                                                   |
   MOUNT FILESYSTEM('HFS.WORKDS') MOUNTPOINT('/u/john/namedir1/namedir2
/namedir3/namedir4') TYPE(HFS) MODE(RDWR)
```

Rather than defining parameter limit values in their full decimal or hexadecimal form, you can use the following 1–character multiplier (denomination values) suffix when specifying them. This value will also be used in displays when the system returns responses to respective D OMVS commands.

Notes:

1. Only those SETOMVS parameters that support this ″C″ suffix specifically note that support and refer to Table 25 on page 128.

2. Values that **contain** a multiplier are limited to 8 digits (nnnnnnnnC) and those values are limited to X'00FF FFFF' (16 777 215 decimal).
3. Values that **do not contain** a multiplier are limited to X'7FFF FFFF' (2 147 483 647 decimal).

Table 25. 1–Character Parameter Limit Multipliers

Denomination Value	1–Character Abbreviation	Bytes
null	n/a	1
Kilo	**K**	1,024
Mega	**M**	1,048,576
Giga	**G**	1,073,741,824
Tera	**T**	1,099,511,627,776
Peta	**P**	1,125,899,906,842,624

Syntax of BPXPRMxx

```
{AUTOCVT(ON|OFF)}

{MAXPROCSYS(nnnnn)}

{MAXPROCUSER(nnnnn)}

{MAXUIDS(nnnnn)}

{MAXFILEPROC(nnnnnn)}

{MAXTHREADTASKS(nnnnn)}

{MAXTHREADS(nnnnnn)}

{MAXPTYS(nnnnn)}

{MAXFILESIZE(nnnnn|NOLIMIT)}

{MAXCORESIZE(nnnnn)}

{MAXASSIZE(nnnnn)}

{MAXCPUTIME(nnnnn)}

{MAXMMAPAREA(nnnnn)}

{MAXSHAREPAGES(nnnnn)}

{RESOLVER_PROC(nnnnn|DEFAULT|NONE)}

{SHRLIBRGNSIZE(nnnnn)}

{SHRLIBMAXPAGES(nnnnn)}

{PRIORITYGOAL(service_class_name1,...service_class_name40|NONE)}

{IPCMSGNIDS(nnnnn)}

{IPCMSGQBYTES(nnnnn)}

{IPCMSGQMNUM(nnnnn)}

{IPCSEMNIDS(nnnnn)}
```

```
                        {IPCSEMNOPS(nnnnn)}

                        {IPCSEMNSEMS(nnnnn)}

                        {IPCSHMMPAGES(nnnnn)}

                        {IPCSHMNIDS(nnnnn)}

                        {IPCSHMNSEGS(nnnnn)}

                        {IPCSHMSPAGES(nnnnn)}

                        {FORKCOPY(COW|COPY)}

                        {SUPERUSER(user_name)}

                        {TTYGROUP(group_name)}

                        {CTRACE(parmlib_member_name)}

                        {STEPLIBLIST('/etc/steplib')}

                        {USERIDALIASTABLE('/etc/tablename')}

                        {SERV _LPALIB('dsname', 'volser')}

                        {SERV _LINKLIB('dsname', 'volser')}
                        {FILESYSTYPE TYPE(type_name)
                               ENTRYPOINT(entry_name)
                               PARM('parm')}
                               ASNAME(proc_name,'start_parms')

                        {SYSPLEX(YES|NO)}

                        {VERSION('nnnn')}

                        {ROOT  FILESYSTEM('fsname') or DDNAME(ddname)
                               TYPE(type_name)
                               MODE(access)
                               PARM('parameter')
                               SETUID|NOSETUID
                               SYSNAME(sysname)
                               TAG(NOTEXT|TEXT,ccsid)}
                               AUTOMOVE|NOAUTOMOVE}
                               MKDIR('pathname')

                        {MOUNT FILESYSTEM('fsname') or DDNAME(ddname)
                               TYPE(type_name)
                               MOUNTPOINT('pathname')
                               MODE(access)
                               PARM('parameter')
                               SETUID|NOSETUID
                               SECURITY|NOSECURITY
                               SYSNAME(sysname)
                               TAG(NOTEXT|TEXT,ccsid)}
|                              AUTOMOVE[(INCLUDE,sysname1,sysname2,...,[sysnameN|*])
|                               |(EXCLUDE,sysname1,sysname2,...,sysnameN)]
|                               |NOAUTOMOVE|UNMOUNT
                      }
                               MKDIR('pathname')

                        {NETWORK DOMAINNAME(sockets_domain_name)
                               DOMAINNUMBER(sockets_domain_number)
                               MAXSOCKETS(nnnnn)
                               TYPE(type_name)
                               INADDRANYPORT(starting_port_number)
                               INADDRANYCOUNT(number_of_ports_to_reserve)}
```

```
{SUBFILESYSTYPE NAME(transport_name)
       TYPE(type_name)
       ENTRYPOINT(entry_name)
       PARM('parameter')
       DEFAULT}

{STARTUP_PROC(procname)}

{STARTUP_EXEC('dsname(membername)',class)}

{RUNOPTS('string')}

{SYSCALL_COUNTS(YES|NO)}

{MAXQUEUEDSIGS(nnnnnn)}
{LIMMSG(NONE|SYSTEM|ALL)}
{AUTHPGMLIST('/etc/authfile')|NONE}
{SWA(ABOVE|BELOW)}
```

Syntax example of BPXPRMxx

```
AUTOCVT(OFF)
MAXPROCSYS(400)
MAXPROCUSER(16)
MAXUIDS(200)
MAXFILEPROC(20)
MAXTHREADTASKS(100)
MAXTHREADS(500)
MAXPTYS(100)
MAXFILESIZE(1000)
/*--- or --- MAXFILESIZE(300M) -----*/
MAXCORESIZE(4194304)
/*--- or --- MAXCORESIZE(1K) -------*/
MAXASSIZE(41943040)
/*--- or --- MAXASSIZE(10M) --------*/
MAXCPUTIME(1000)
MAXMMAPAREA(4096)
/*--- or --- MAXMMAPAREA(16M) ------*/
 MAXSHAREPAGES(32768)
/*--- or --- MAXSHAREPAGES(10K) ----*/
PRIORITYGOAL(CICS4,CICS4,CICS4,CICS3,CICS2,CICS1,TSO2,TSO1,BAT3,BAT2)
IPCMSGNIDS(500)
IPCMSGQBYTES(262144)
IPCMSGQMNUM(100000)
IPCSEMNIDS(500)
IPCSEMNOPS(25)
IPCSEMNSEMS(25)
IPCSHMMPAGES(256)
/*--- or --- IPCSHMMPAGES(55M) -----*/
IPCSHMNIDS(500)
IPCSHMNSEGS(10)
IPCSHMSPAGES(262144)
/*--- or --- IPCSHMSPAGES(300K) ----*/
FORKCOPY(COW)
SUPERUSER(BPXROOT)
TTYGROUP(TTY)
CTRACE(CTCBPX23)

STEBLIBLIST('/etc/steplib')
USERIDALIASTABLE('/etc/tablename')
SYSPLEX(YES)
 VERSION('REL9')
 FILESYSTYPE TYPE(HFS)
      ENTRYPOINT(GFUAINIT)
      PARM('SYNCDEFAULT(0) FIXED(2) VIRTUAL(128)')
ROOT  FILESYSTEM('OMVS.ROOT')
      TYPE(HFS)
      MODE(RDWR)
      SYSNAME(SY1)
      TAG(NOTEXT,0)
      AUTOMOVE

MOUNT FILESYSTEM('OMVS.USER.JONES')
      TYPE(HFS)
      MOUNTPOINT('/u/jones')
      MODE(RDWR)
      SYSNAME(SY1)
      TAG(TEXT,1047)
      AUTOMOVE(INCLUDE,SYS1,SYS2,*)
```

```
FILESYSTYPE TYPE(INET)
      ENTRYPOINT(EZBPFINI)
NETWORK DOMAINNAME(AF_INET)
      DOMAINNUMBER(2)
      MAXSOCKETS(64000)
      TYPE(INET)
STARTUP_PROC(OMVS)
STARTUP_EXEC('OMVS.ROOT(REXX01)',A)
RUNOPTS('RTLS(ON) LIBRARY(SYSCEE) VERSION(OS24)')
SYSCALL_COUNTS(YES)
MAXQUEUEDSIGS(1000)
RESOLVER_PROC(DEFAULT)
AUTHPGMLIST('/etc/authfile')
SWA(ABOVE)
```

Notes:

1. This is an example only; the values presented here are not optimal nor recommended values for your installation.
2. The **bold comments** present alternate specifications of those parameters that support the use of multipliers. Refer to Table 25 on page 143 for information concerning the use of multipliers.

IBM-supplied default for BPXPRMxx

There is no default BPXPRMxx parmlib member. A sample parmlib member BPXPRMXX is provided in SYS1.SAMPLIB.

Statements and parameters for BPXPRMxx

For guidance information about selecting values for the statements, see the chapter on customizing z/OS UNIX in *z/OS UNIX System Services Planning*.

AUTOCVT(ON|OFF)

Activates and deactivates automatic conversion of I/O data using coded character sets for the program and its associated files.

The coded character set identifiers (CCSIDs) are specified by the program or by setting the appropriate environment variables at run time. The system AUTOCVT indicator can be overridden by individual programs at a thread level; AUTOCVT is a controlling switch only for existing programs that do not explicitly establish their own conversion environment.

Default: OFF

You can use the SETOMVS or SET OMVS commands to change the value of AUTOCVT between ON and OFF. Changing this conversion mode does not affect conversion of opened files for which I/O has already started.

When AUTOCVT(ON) is set, every read and write operation for a file must be checked to see if conversion is necessary. Thus, there is a performance penalty involved, even if no conversion occurs. It is, therefore, preferable to keep AUTOCVT(OFF) and have each program enabled, if possible, for conversion. To do this, set the compile or run time environment variables that control conversion or by issuing fcntl().

Note: If you are using SYSPLEX(YES) and mixed releases of z/OS UNIX, you can IPL specifying OMVS=(delta,common) for each unique release, where ″delta″ identifies the member containing the new keywords for that release, and ″common″ identifies the common keywords for all releases.

Automatic conversion can also be controlled individually by a program with one of the following flags in the thread Thli control block (BPXYTHLI):

```
ThliCvtOn - Activates automatic conversion for this thread.
ThliCvtOff - Deactivates automatic conversion for this thread.
```

Both bits must not be on at the same time.

Automatic conversion is accomplished between programs and files that are tagged with different CCSIDs when a conversion table exists for that CCSID pair in the system. CCSID values are defined in *Character Data Representation Architecture*.

MAXPROCSYS(nnnnn)
Specifies the maximum number of processes that the system allows.

Value Range: *nnnnn* is a decimal value from 5 to 32767.

Default: 200

You can use the SETOMVS or SET OMVS command to dynamically increase or decrease the value of MAXPROCSYS. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

If you are using SETOMVS or SET OMVS to change the value, the new value must be within a certain range, or you will get an error message. The range that you can use has a minimum value of 5; the maximum value is based on the following calculation:

```
MIN(32767,MAX(4096,3*initial value))
```

The initial value is the MAXPROCSYS value that was specified during BPXPRMxx initialization. You cannot use a value less than 5. If you want to use a value greater than the current maximum (as calculated by the formula) but lower than the initial maximum (32767), you will have to change the value in BPXPRMxx and re-IPL. For an example of how to calculate the maximum value in the range, see “Dynamically Changing Certain BPXPRMxx Parameter Values” in *z/OS UNIX System Services Planning*.

For planning information, see **MAXPROCSYS** in *z/OS UNIX System Services Planning*.

MAXPROCUSER(nnnnn)
Specifies the maximum number of processes that a single z/OS UNIX user ID can have concurrently active, regardless of how the processes were created. MAXPROCUSER is the same as the CHILD_MAX variable in the POSIX standard.

A value of 25 is required for FIPS 151-2 compliance and a value of 16 is required for POSIX.1 (ISO/IEC 9945-1:1990[E] IEEE Std 1003.1-1990) standard compliance.

The number of processes is tracked by user ID (UID). When a user attempts to create a new process, the limit value for the user (defined by either the user profile or the default OPTN value) is compared to the value maintained for the user’s UID. If the user maximum is larger than the current process count for the UID, the user can create another process. If not, the user is not allowed to create a new process. For example, if user “A”, with a user-defined limit of 10, tries to create a process and the UID limit is already 12, user “A” is not allowed to create the new process. Since only 12 processes are currently created, user “B”, with a user-defined limit of 20, is allowed to create a new process.

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the MAXPROCUSER values. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

There are certain cases when a new process can be created with a UID that has already reached the MAXPROCUSER number of processes. Super users (with UID=0) are not limited by MAXPROCUSER. Additionally, if an undubbed address space is dubbed because of a request to use a kernel resource and is dubbed with the default OMVS segment, it will not be limited by the MAXPROCUSER limit either.

For planning information, see **MAXPROCUSER** in *z/OS UNIX System Services Planning*.

Value Range: *nnnnn* is a decimal value from 3 to 32767.

Default: 25

MAXUIDS(nnnnn)
Specifies the maximum number of z/OS UNIX user IDs (UIDs) that can operate concurrently.

Value Range: *nnnnn* is a decimal value from 1 to 32767.

Default: 200

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the value of MAXUIDS. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For planning information, see **MAXUIDS** in *z/OS UNIX System Services Planning*.

MAXFILEPROC(nnnnnn)
Specifies the maximum number of descriptors for files, sockets, directories, and any other file system objects that a single process can have concurrently active or allocated. MAXFILEPROC is the same as the OPEN_MAX variable in the POSIX standard.

Value Range: *nnnnnn* is a decimal value from 3 to 131072.

Default: 2000

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the value of MAXFILEPROC. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For planning information, see **MAXFILEPROC** in *z/OS UNIX System Services Planning*.

MAXTHREADTASKS(nnnnn)
Specifies the maximum number of MVS tasks that a single process can have concurrently active for pthread_created threads.

Value Range: *nnnnn* is a decimal value from 0 to 32768.

Default: 1000

You can change the value of MAXTHREADTASKS dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For planning information, see **MAXTHREADTASKS** in *z/OS UNIX System Services Planning*.

MAXTHREADS(nnnnnn)
Specifies the maximum number of pthread_created threads, including running, queued, and exited but undetached, that a single process can have concurrently active. Specifying a value of 0 prevents applications from using pthread_create.

Value Range: *nnnnnn* is a decimal value from 0 to 100000.

Default: 200

You can change the value of MAXTHREADS dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For planning information, see **MAXTHREADS** in *z/OS UNIX System Services Planning*.

MAXPTYS(nnnnn)
Specifies the maximum number of pseudoterminals (pseudo-TTYs or PTYs) for the system.

Value Range: *nnnnn* is a decimal value from 1 to 10000.

Default: 800

You can use the SETOMVS or SET OMVS command to dynamically increase the value of MAXPTYS. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

If you are using SETOMVS or SET OMVS to change the value, the new value must be within a certain range. If it is outside the range, you will get an error message. To use a value that is outside this range, you must change the MAXPTYS specification in BPXPRMxx and re-IPL. The range's minimum value is 1 and the maximum is based on the following calculation:

```
MIN(10000,MAX(256,2*initial value)
```

The initial value is the MAXPTYS value that was specified during BPXPRMxx initialization. For an example of how to calculate the maximum value in the range, see "Dynamically Changing Certain BPXPRMxx Parameter Values" in *z/OS UNIX System Services Planning*.

For planning information, see **MAXPTYS** in *z/OS UNIX System Services Planning*.

MAXFILESIZE(nnnnn|NOLIMIT)
Specifies the RLIMIT_FSIZE soft and hard resource values that a process receives when it is identified as a process. RLIMIT_FSIZE indicates the maximum file size (in 4KB increments) that a process can create. It also specifies the limit when they are initiated by a daemon process using an **exec()** after a **setuid()**. For more information about RLIMIT_FSIZE, see the description of **setrlimit()** in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Value Range: *nnnnn* is a decimal value from 0 to 2147483647, which indicates an unlimited file size. If MAXFILESIZE is not specified or MAXFILESIZE(NOLIMIT) is specified, there will be no limit to the size of files created, except for the architectural limit of the system.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 143, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

If you specify 0, the process does not create any files. Omitting this statement indicates an unlimited file size.

Default: 1000

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the value of MAXFILESIZE. To make a permanent change, edit the BPXPRMxx member that will be used in IPLs.

MAXCORESIZE(nnnnn)
Specifies the RLIMIT_CORE soft and hard resource values that a process receives when it is identified as a process. RLIMIT_CORE indicates the maximum core dump file size (in bytes) that a process can create. It also specifies the limit when they are initiated by a daemon process using an **exec()** after a **setuid()**. For more information about RLIMIT_CORE, see the description of **setrlimit()** in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Value Range: *nnnnn* is a decimal value from 0 to 2147483647 (2 gigabytes).

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 143, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

Default: 4194304 (4 megabytes) Specifying a value of 2147483647 (2 gigabytes) indicates an unlimited core file size.

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the value of MAXCORESIZE. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

MAXASSIZE(nnnnn)
Specifies the RLIMIT_AS resource values that a process receives when it is identified as a process. RLIMIT_AS indicates the address space region size. For more information about RLIMIT_AS, refer to the description of **setrlimit** in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

The soft limit is obtained from MVS; if it is greater than the MAXASSIZE value, the soft limit is set to the hard limit. This value is also used when processes are initiated by a daemon process using an **exec()** after **setuid()**. In this case, both the RLIMIT_ AS hard and soft limit values are set to the MAXASSIZE specified value.

When processes are initiated by a daemon process using an **exec()** after **setuid()**, this value is used. Therefore, MAXASSIZE will be the region size for

all processes created via rlogin or telnet. In this case, both the RLIMIT_AS hard and soft limit values are set to the MAXASSIZE value.

A superuser can override this value by specifying a new region size in the spawn inheritance structure on **__spawn()**. Or you can change the value of MAXASSIZE dynamically by using the SETOMVS or SET OMVS command.This change only affects the new processes created after the change was made.

Note: The IEFUSI user exit can modify the region size of an address space. Users are strongly discouraged from altering the region size of address spaces in the OMVS subsystem category.

Value Range: *nnnnn* is a decimal value from 10485760 (10 megabytes) to 2147483647 (2 Gigabytes).

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 143, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command.

Default: 209715200

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the value of MAXASSIZE. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For planning information, see **MAXASSIZE** in *z/OS UNIX System Services Planning*.

MAXCPUTIME(nnnnn)

Specifies the RLIMIT_CPU resource values that a process receives when it is identified as a process. RLIMIT_CPU indicates the CPU time, in seconds, that a process can use. For more information about RLIMIT_CPU, refer to the description of **setrlimit()** in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

If the soft limit value from MVS is greater than the MAXCPUTIME value, the soft limit is set to the hard limit. This value is also used when processes are initiated by a daemon process using an **exec()** after **setuid()**. In this case, both the RLIMIT_CPU hard and soft limit values are set to the MAXCPUTIME value.

A superuser can override this value by specifying a new time limit in the spawn inheritance structure on **__spawn()**.

For processes running in or forked from TSO or BATCH, the MAXCPUTIME value has no effect. The TIME limit is inherited from the parent. If a TIME parameter is specified on the JCL for the started task, then that value is used. If not, then the TIME value is taken from the JES default TIME value.

For processes created by the **rlogind** command or other daemons, MAXCPUTIME is the time limit for the address space.

Specifying a MAXCPUTIME or CPUTIMEMAX of 86400 seconds disables the JWT timeout the same way that JCL TIME=1440 does.

Value Range: *nnnnn* is a decimal value from 7 to 2147483647 seconds.

Default: 1000

Use the SETOMVS or SET OMVS command to dynamically increase the value of MAXCPUTIME. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For planning information, see **MAXCPUTIME** in *z/OS UNIX System Services Planning*.

MAXMMAPAREA(nnnnn)

Specifies the maximum amount of data space storage space (in pages) that can be allocated for memory mappings of HFS files. Storage is not allocated until the memory mapping is active.

Using memory map services causes a large amount of system memory to be consumed. For each page (4KB) that is memory-mapped, 96 bytes of ESQA are consumed when a file is not shared with any other users. When a file is shared by multiple users, each user after the first causes 32 bytes of ESQA to be consumed for each shared page. Assuming that the default of 40960 pages is taken, and assuming that no sharing is done by **mmap()** users, a maximum of 384KB of ESQA could be consumed. The ESQA storage is consumed when the **mmap()** function is invoked rather than when the page is accessed by the memory mapping application program.

If you have applications using the __MAP_MEGA option, you can map very large files without the system overhead in ESQA. For more information, see "Extended System Queue Area (ESQA)" in *z/OS UNIX System Services Planning*.

Value Range: *nnnnn* is a decimal value from 1 to 16777216.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 143, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

Default: 40960

You can change the value of MAXMMAPAREA dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For planning information, see **MAXMMAPAREA** in *z/OS UNIX System Services Planning*.

MAXSHAREPAGES(nnnnn)

Specifies the maximum amount of shared system storage pages that can be used by z/OS UNIX functions. The purpose of MAXSHAREPAGES is to limit the amount of ESQA storage necessary to maintain the shared pages. For a detailed description of how MAXSHAREPAGES affects ESQA usage, please refer to *z/OS UNIX System Services Planning*.

The usage of shared pages is helpful but not critical to the loading of user shared library modules, ptrace and fork; it serves to increase performance but does not affect functionality. As the amount of shared pages being used reaches certain limits, less functions are allowed to continue using them. User shared library loads, ptrace and fork stop using shared pages when the limit reaches 60% (the only time shared storage is used by the fork service is when

FORKCOPY(COW) is specified), mmap stops at 80%, and shmat, the most critical function, uses shared pages until their total capacity has been reached. If while running a 64–bit program, you allocate shared memory segments above the bar by using the shmget() service, the shared page limit is not affected.

Because each page of shared storage requires the associated consumption of extended system queue area (ESQA) storage, limiting the shared storage usage provides a way to limit the ESQA usage by z/OS UNIX users. If you use the __IPC_MEGA or __MAP_MEGA options, then the shared pages limits are not affected because MEGA does not affect the system ESQA overhead.

Note: You should evaluate adjusting MAXSHAREPAGES on an active system. Dynamically decreasing the number of pages available to ESQA for active work can cause errors. This is due to the fact that for those jobs, the ESQA limit may now be reached or exceeded. It is possible that shared programs will not be able to be loaded and fork() may not succeed. This situation will exist until the workload adjusts to the new lower limit.

Value Range: *nnnnn* is a decimal value from 0 to 32768000 specifying a number of 4K pages.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 143, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

Default: 131072

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the MAXSHAREPAGES value. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

SHRLIBRGNSIZE(nnnnn)

Specifies the maximum size of the shared library region for address spaces that load system shared library modules. For these address spaces, the size specified is allocated from high private storage and is used for the loading of system shared library modules. This storage is not allocated in an address space until it loads a system shared library module. This parameter applies to modules loaded from system shared libraries, which allocate storage on megabyte boundaries. Therefore, this storage does not count against the MAXSHAREPAGES limit, and does not consume ESQA.

Value Range: *nnnnn* is a decimal value between 16777215 (16 megabytes) and 1610612735 (1.5 gigabytes).

Default: 67108863

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the SHRLIBRGNSIZE value. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

SHRLIBMAXPAGES(nnnnn)

This parameter is intended to control the maximum number of pages that can be allocated in the system to contain user shared library modules. This value, in conjunction with MAXSHAREPAGES, can be used to control the amount of

ESQA consumed by user shared library modules. Please refer to *z/OS UNIX System Services Planning* for additional details.

Value Range: *nnnnn* is a decimal value between 1 and 16777215 specifying a number of 4K pages.

Default: 4096

Use the SETOMVS or SET OMVS command to dynamically increase or decrease the SHRLIBMAXPAGES value. To make a permanent change, edit the BPXPRMXX member that will be used for IPLs.

PRIORITYGOAL(service_class_name1,...service_class_name40)
Specifies a list of 1 to 40 service class names of 8 characters or less separated by commas, which are used in association with the **setpriority**, **nice** and **chpriority** callable services when the system is running in goal mode. These functions allow a program to alter the priority of one or more processes.

Generally, it is recommended that you not set PRIORITYGOAL unless the nice(), setpriority() or chpriority() values must be enabled.

If the list has less than 40 entries, the system propagates the last service class specified into the remaining unspecified entries in the table. For example:

```
PRIORITYGOAL(CICS4,CICS4,CICS4,CICS3,CICS2,CICS1,TSO2,TSO1,BAT3,BAT2)
```

If you do not specify this statement, arrays are not created for it. All service classes specified on the PRIORITYGOAL statement must also be specified in your workload manager service policy.

PRIORITYGOAL(NONE) means that there are no values. If you do not specify PRIORITYGOAL, that means that there are no values.

If you do not want to allow users to increase the priority but still want to enable the **nice()** and **setpriority()** functions, define a range of service classes with priority increments on a base that is normal for the users. Using these functions lets the user order the priority of processes, but will not let a user improve performance over that of other users.

Value Range: *service_class_name* is a 1 to 8 character value.

Default: None

You can dynamically change the values of PRIORITYGOAL by using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

IPCMSGNIDS(nnnnn)
Specifies the maximum number of unique system-wide message queues.

Value Range: *nnnnn* is a decimal value from 1 to 20000.

Default: 500

You can change the value of IPCMSGNIDS dynamically using the SETOMVS or SET OMVS command. The new minimum is the current value. The new maximum is calculated as follows:

```
MIN(initial maximum,MAX(4096,3*initial value))
```

You can increase but not decrease the value, as described in *z/OS UNIX System Services Planning*.

IPCMSGQBYTES(nnnnn)
Specifies the maximum number of bytes in a single message queue.

Value Range: *nnnnn* is a decimal value from 0 to 2147483647.

Note: The high end of this range is not obtainable due to storage constraints. The actual maximum range varies due to storage allocation and system usage.

Default: 262144

You can change the value of IPCMSGQBYTES dynamically using the SETOMVS or SET OMVS command.

IPCMSGQMNUM(nnnnn)
Specifies the maximum number of system-wide messages for each queue.

Value Range: *nnnnn* is a decimal value from 0 to 2147483647.

Note: The high end of this range is not obtainable due to storage constraints. The actual maximum range varies due to storage allocation and system usage.

Default: 10000

You can change the value of IPCMSGQMNUM dynamically using the SETOMVS or SET OMVS command.

IPCSEMNIDS(nnnnn)
Specifies the maximum number of unique system-wide semaphore sets.

Value Range: *nnnnn* is a decimal value from 1 to 20000.

Default: 500

You can change the value of IPCSEMNIDS dynamically using the SETOMVS or SET OMVS command, as described in *z/OS UNIX System Services Planning*.

IPCSEMNOPS(nnnnn)
Specifies the maximum number of operations for each **semop** call.

Value Range: *nnnnn* is a decimal value from 0 to 32767.

Default: 25

You can change the value of IPCSEMNOPS dynamically using the SETOMVS or SET OMVS command.

IPCSEMNSEMS(nnnnn)
Specifies the maximum number of semaphores for each semaphore set.

Value Range: *nnnnn* is a decimal value from 0 to 32767.

Default: 1000

You can change the value of IPCSEMNSEMS dynamically using the SETOMVS or SET OMVS command.

IPCSHMMPAGES(nnnnn)
Specifies the maximum number of pages for shared memory segments.

Value Range: *nnnnn* is a decimal value from 1 to 4 petabytes (that is, 4 * 1 125 899 906 842 624).

Note: If you obtain memory segments below the 2–gigabyte address range then a realistic maximum is about 1.5 gigabytes; the actual maximum range varies due to storage allocation and system usage. If you obtain memory segments above the 2–gigabyte address range, the maximum is dependent on the IEASYS HVSHARE parameter which specifies the size of the high virtual shared area.

Default: 25600

You can change the value of IPCSHMMPAGES dynamically using the SETOMVS or SET OMVS command.

IPCSHMNIDS(nnnnn)
Specifies the maximum number of unique system-wide shared memory segments.

Value Range: *nnnnn* is a decimal value from 1 to 20000.

Default: 500

You can change the value of IPCSHMNIDS dynamically using the SETOMVS or SET OMVS command. The new minimum is the same as the current value. The new maximum is calculated as follows:

```
MIN(initial maximum,MAX(4096,3*initial value))
```

You can increase but not decrease the value, as described in *z/OS UNIX System Services Planning*.

IPCSHMNSEGS(nnnnn)
Specifies the maximum number of attached shared memory segments for each address space.

Value Range: *nnnnn* is a decimal value from 0 to 1000.

Default: 10

You can change the value of IPCSHMNSEGS dynamically using the SETOMVS or SET OMVS command.

IPCSHMSPAGES(nnnnn)
Specifies the maximum number of system-wide shared pages created by calls to the **fork** and 31–bit**shmat** functions.

You can increase, but not decrease, the value, as described in *z/OS UNIX System Services Planning*. Shared memory segments obtained above the 2–gigabyte range in 64–bit programs do not affect this limit.

Value Range: *nnnnn* is a decimal value from 0 to 2621440.

Note: You can set a denomination (or multiplier) value when defining this value. The suffix, ″C″ can have a 1–character value as presented in Table 25 on page 143, but must not exceed the parameter-specific upper limit.

MVS retains the denomination value and uses it within a subsequent D OMVS command

Default: 262144

You can change the value of IPCSHMSPAGES dynamically using the SETOMVS or SET OMVS command. The new minimum is the same as the current value. The new maximum is calculated as follows:

```
MIN(initial maximum,MAX(4096,3*initial value))
```

You can increase but not decrease the value, as described in *z/OS UNIX System Services Planning*.

Because each page of shared storage requires the associated consumption of extended system queue area (ESQA) storage, limiting the shared storage usage provides a way to limit the ESQA usage by z/OS UNIX users. If you use the __IPC_MEGA or __MAP_MEGA options, then the shared pages limits are not affected because MEGA does not affect the system ESQA overhead.

FORKCOPY(COW|COPY)

Specifies how user storage is to be copied from the parent process to the child process during a **fork()** system call.

FORKCOPY(COW) specifies that all **fork()** calls are processed with the copy-on-write mode if the suppression-on-protection (SOP) hardware feature is available. Before the storage is modified, both the parent and child process refer to the same view of the data. The parent storage is copied to the child only if either the parent or the child modifies the storage. FORKCOPY(COW) causes the system to use the ESQA to manage page sharing.

FORKCOPY(COPY) specifies that **fork()** immediately copies the parent storage to the child, whether the SOP is available or not. Use this option to avoid any additional ESQA use in support of fork.

Follow these guidelines:

- If the run-time library is in the link pack area, specify FORKCOPY(COPY).
- If the run-time library is not in the link pack area, specify FORKCOPY(COW).

Default: COW

You can change the value of FORKCOPY dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member used for IPLs.

SUPERUSER(user_name)

Superuser name, which must conform to the restrictions for the z/OS user ID. The user name must also be defined to RACF (or another security product) and must have a z/OS UNIX user ID (UID) of 0. For example, in RACF, specify OMVS(UID(0)) on the ADDUSER command.

When a daemon issues a **setuid()** to set a UID to 0 and the user ID is not known, **setuid()** uses the user ID from the SUPERUSER statement.

Never permit the userid BPXROOT to the BPX.DAEMON profile (described in “Setting Up the BPX.* FACILITY Class Profiles“ in *z/OS UNIX System Services Planning*). This warning applies even if you use a name other than BPXROOT.

Value Range: *user_name* is a 1 to 8 character value.

Default: BPXROOT

Use the SETOMVS or SET OMVS command to dynamically change the value of SUPERUSER. To make a permanent change, edit the BPXPRMxx member that is used for IPLs.

TTYGROUP(group_name)
Specifies the z/OS group name given to slave pseudoterminals (PTYs) and OCS remote terminals (RTYs). This group name should be defined to the security product and must have a unique group ID (GID). No users should be connected to this group.

The group_name is used by certain **setgid()** programs, such as talk and write, when writing to another user's PTY or RTY.

Value Range: *group_name* is a 1 to 8 character value.

Default: TTY

You can change the value of TTYGROUP dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for future IPLs.

CTRACE(parmlib_member_name)
Specifies the parmlib member that contains the initial tracing options to be used for the z/OS UNIX component. Use this statement to provide tracing while the kernel is starting and to avoid having to issue a TRACE operator command to set tracing options.

Default: CTIBPX00

STEPLIBLIST('/etc/steplib')
Specifies the pathname of a hierarchical file system (HFS) file. This file is intended to contain a list of MVS data sets that are sanctioned by the installation for use as step libraries for programs that have the set-user-ID and set-group-ID bit set.

Use the SETOMVS or SET OMVS command to dynamically change the value of STEPLIBLIST. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For additional information, see **STEPLIBLIST** in *z/OS UNIX System Services Planning*.

USERIDALIASTABLE('/etc/tablename')
Specifies the pathname of a hierarchical file system (HFS) file. This file is intended to contain a list of z/OS user IDs and group names with their corresponding alias names. The alias names can contain any characters in the portable filename character set.

You can change USERIDALIASTABLE dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

Once a user is logged into the system, changing the user ID or group name alias table does not change the alias name immediately. If a change needs to be activated sooner, you can use the SETOMVS or SET OMVS command to change the table more quickly.

For planning information, see **USERIDALIASTABLE** in *z/OS UNIX System Services Planning*.

SERV_LPALIB('dsname', 'volser')
Specifies the target service library where the UNIX System Services modules that are normally built into LPA are located. The load library specified must be an APF Authorized Library or the F OMVS,ACTIVATE=SERVICE command fails when referencing the library.

Value Range: *dsname* is a 1-to-44 character value representing a valid MVS load library data set name. The alphabetic characters in the load library name must be uppercase. *volser* is a 1-to-6 character value representing a valid volume serial number for the volume that contains the specified MVS load library. The alphabetic characters in the volume serial number must be uppercase.

You can change the value of SERV_LPALIB dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for future IPLs.

SERV_LINKLIB('dsname', 'volser')
Specifies the target service library where the UNIX System Services modules that are normally loaded from SYS1.LINKLIB into the private area of the OMVS address space are located. The load library specified must be an APF Authorized Library of the F OMVS,ACTIVATE=SERVICE command fails when referencing the library.

Value Range: *dsname* is a 1-to-44 character value representing a valid MVS load library data set name. The alphabetic characters in the load library name must be uppercase. *volser* is a 1-to-6 character value representing a valid volume serial number for the volume that contains the specified MVS load library. The alphabetic characters in the volume serial number must be uppercase.

You can change the value of SERV_LINKLIB dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for future IPLs.

FILESYSTYPE TYPE(type_name) ENTRYPOINT(entry_name) PARM('parm') ASNAME(proc_name[,'start_parms'])
Specifies the type of file system that is to be started. BPXPRMxx can contain more than one FILESYSTYPE statement.

When SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating shared file system group must be defined for all systems participating in shared file system. The easiest way to accomplish this is by having a single BPXPRMxx member that contains file system information for each system participating in shared file system. If you decide to define a BPXPRMxx for each system, the FILESYSTYPE statements must be identical on each system. For more information on shared file system, see *z/OS UNIX System Services Planning*.

Note that any facilities required for a particular FILESYSTYPE must be initiated on all systems participating in shared file system. For example, NFS requires TCP/IP, so if you specify an NFS FILESYSTYPE, you must also initialize TCP/IP on NFS initialization.

The SETOMVS RESET command can be used to dynamically specify new FILESYSTYPE statements. To make a permanent change, edit the BPXPRMxx member used for IPLs. For more information, see "Dynamically Adding FILESYSTYPE Statements in BPXPRMxx" in *z/OS UNIX System Services Planning*.

The parameters are:

TYPE(type_name)
Specifies the name of the file system type that is to control the file system.

In the FILESYSTYPE statement, specify a name for the TYPE file system. For example, you could use the following, or assign your own names:

- ZFS for a zSeries® file system (ZFS)
- HFS for a hierarchical file system (HFS)
- TFS for a temporary file system (TFS)
- UDS for z/OS UNIX domain (AF_UNIX) sockets
- INET for network (AF_INET and AF_INET6) sockets
- CINET for common INET (AF_INET and AF_INET6) sockets
- AUTOMNT for an automounted file system
- DFSC for accessing global namespace.
- NFS for accessing remote files.

For planning information, see **FILESYSTYPE** in *z/OS UNIX System Services Planning*.

TYPE is a required parameter. The name is 1 to 8 characters; the system converts the name to uppercase.

ENTRYPOINT(entry_name)
Specifies the name of the load module containing the entry point into the file system type.

ENTRYPOINT is a required parameter. The name is 1 to 8 characters; the system converts the name to uppercase. Refer to the documentation for the specific physical file system for valid entry point names.

PARM('parm')
Provides a parameter to be passed directly to the file system type. The parameter format and content are specified by the file system type.

PARM is an optional parameter. The parameter is up to 500 characters long; the characters can be in uppercase, lowercase, or both. The parameter must be enclosed in single quotes.

If the physical file system specified does not expect a PARM operand, it ignores all PARM operands.

SYNCDEFAULT(t), VIRTUAL(max), FIXED(min) and **FSFULL(threshold,increment)** are valid only when **ENTRYPOINT** is GFUAINIT.

Note: If a syntax error is found in any of these four parameters (**SYNCDEFAULT, VIRTUAL, FIXED,** or **FSFULL**), an error message is issued and **all four parameters** are set to the default values.

SYNCDEFAULT(t)

t specifies the number of seconds used as a default for the sync daemon interval. When the sync daemon is active, the meta data for a file system is hardened. Setting *t* to 0 indicates that the file system should harden meta data synchronously with syscall requests.

Sync interval values are rounded up to the next 30-second value. For example, specifying 31 seconds results in a sync interval of 60 seconds.

The maximum value that can be specified for *t* is 65534. Values between 65534 and 99999 are rejected.

A value of 99999 specifies that no sync daemon intervals are specified, and thus, the meta data is not hardened.

Default: 60 seconds

VIRTUAL(max)

max specifies the maximum amount of virtual storage (in megabytes) that HFS data and meta data buffers should use.

If you do not set a value for *max*, the system assigns a default value that is equal to half the amount of real storage available to the system at HFS initialization. See *z/OS UNIX System Services Planning* for more information about the VIRTUAL(max) parameter.

FIXED(min)

min specifies the amount of virtual storage (in megabytes) that is fixed at HFS initialization time and remains fixed even if HFS activity drops to zero. *min* must be less than or equal to **VIRTUAL(max)**.

min cannot exceed 50% of real storage available to the system. If the allowed amount of storage is exceeded, an informational message is issued and *min* is set to 50% of real storage. The minimum limit can be changed dynamically by invoking the **confighfs** shell command. See *z/OS UNIX System Services Command Reference* for more information about the **confighfs** shell command.

Default: 0

FSFULL(threshold,increment)

threshold specifies the percentage of the HFS capacity at which an operator message is generated. The default is 100%.

increment specifies the percentage of change above the HFS capacity at which an operator messages is generated. Messages are generated by either an increase or decrease greater than *increment*. The default is 5%.

You can specify *threshold* and *increment* values for all HFS file systems. The values can also be set on the MOUNT command for a specific file system. Parameters on the MOUNT command override parmlib values. If no values are specified in either place, no threshold checking is done. If a threshold value is specified but no increment is given, the increment defaults to 5%. The increment value applies both to upgrading the message when the file system continues to fill and to removing the message when more space becomes available due to either deleting files, or to extending the file system. The values are in terms of percent full. The values applied to a file system can be changed only when the file system is mounted.

ASNAME(proc_name[,'start_parms'])

proc_name specifies the name of a procedure in SYS1.PROCLIB that is to be used to start the address space that is initialized by the physical file system (PFS). Specify ASNAME for any PFS that does not run in the kernel address space. The name you specify is also used for the name of the address space.

start_parms is an optional quoted string that is to be appended to the *proc_name* when the address space is started. The string may be up to 100 characters long. The *start_parms* are not validated; they are just passed to the system when the address space is started with an internal command. Refer to the START command in *z/OS MVS System Commands* or the ASCRE macro in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

By default the address space started with ASNAME is started under JES, but this may be changed by including the additional start_parms 'SUB=MSTR'.

ASNAME is an optional parameter. *proc_name* is 1 to 8 characters; the system converts the name to uppercase. If you do not specify ASNAME, or specify *proc_name* as the name of the kernel address space, the PFS is initialized in the kernel address space.

Refer to the documentation for the specific physical file system for valid ASNAME operands.

SYSPLEX(YES|NO)

For z/OS UNIX, the SYSPLEX statement specifies whether a system should join the SYSBPX XCF group to share HFS resources across the sysplex. If SYSPLEX(YES) is specified, the system participates in shared file system. If SYSPLEX(NO) is specified, the system *does not* participate in shared file system. If the SYSPLEX statement is not provided, the default is SYSPLEX(NO). Also, to participate in shared file system, the systems must be at R9 level or later.

For more information on shared file system, see *z/OS UNIX System Services Planning*. IBM suggests that you review this chapter before using any shared file system specific parameters: SYSPLEX(YES|NO), VERSION, AUTOMOVE|NOAUTOMOVE and SYSNAME.

Note: You cannot adjust the SYSPLEX field dynamically. There is no SETOMVS, SET OMVS, or SETOMVS RESET=(xx) capability. To change the value of SYSPLEX, you must re-IPL the system.

Default: NO

VERSION('nnnn')

The VERSION statement applies only to systems that are exploiting shared file systems. VERSION allows multiple releases and service levels of the binaries to coexist and participate in shared file system. A directory with the value *nnnn* specified on VERSION is dynamically created at system initialization under the sysplex root that is used as a mount point for the version file system. This directory, however, is only dynamically created if the sysplex root HFS is mounted read/write.

Note: *nnnn* is a case-sensitive character string no greater than 8 characters in length. It indicates a specific instance of the version file system. The most appropriate values for *nnnn* are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer. For example, if the system is at V2R9, you can specify REL9 for VERSION.

When SYSPLEX(YES) is specified, you must also specify the VERSION parameter.

The VERSION value is substituted in the content of symbolic links that contain $VERSION. For scenarios describing the use of the version HFS, see *z/OS UNIX System Services Planning*.

When testing or changing to a new Maintenance Level (PTF), the VERSION value can be changed dynamically by using the SETOMVS command:

```
SETOMVS VERSION='string'
```

You can also change the settings of this parameter via SET OMVS=(xx) and SETOMVS RESET=(xx) parmlib specifications.

Note: We do not recommend changing VERSION dynamically if you have any users logged on or running applications; replacing the system files for these users may be disruptive.

ROOT FILESYSTEM('fsname') DDNAME(ddname) TYPE(type_name) MODE(access) PARM('parameter') SETUID|NOSETUID AUTOMOVE|NOAUTOMOVE SYSNAME(sysname) TAG(NOTEXT|TEXT,*ccsid*) MKDIR('pathname')

Specifies a file system that z/OS UNIX is to logically mount as the root file system.

Note: The ROOT statement is optional. If not specified, a TFS file system is mounted as the root.

To change the value of the ROOT statement without having to re-IPL, you can use the TSO/E MOUNT and UNMOUNT commands.

The root file system can be unmounted using the TSO/E UNMOUNT command or ISHELL. Ensure that you specify the IMMEDIATE option.

The parameters are:

FILESYSTEM('fsname')
The name of the root file system. The name must be unique in the system.

Either FILESYSTEM or DDNAME is required; do not specify both. The name is 1 to 44 characters; the characters can be in uppercase, lowercase, or both. The name must be enclosed in single quotes. An HFS data set name must conform to the rules of MVS data set names.

DDNAME(ddname)
The ddname on the JCL DD statement that defines the root file system. To use the DDNAME parameter, a DD statement for the HFS data set containing the root file system should be placed in the z/OS UNIX cataloged procedure.

Either FILESYSTEM or DDNAME is required; do not specify both. The ddname is 1 to 8 characters; the system converts the ddname to uppercase.

TYPE(type_name)
Specifies the name of a file system type identified in a FILESYSTYPE statement. The TYPE(type_name) parameter must be the same as the TYPE(type_name) parameter on a FILESYSTYPE statement.

TYPE is a required parameter. The name is 1 to 8 characters; the system converts the name to uppercase.

MODE(access)
Specifies access to the root file system by all users:
- READ: Users can only read the root file system.
- RDWR: Users can read and write in the root file system.

Default: RDWR

PARM('parameter')
Provides a parameter to be passed directly to the file system type. The parameter format and content are specified by the file system type.

PARM is an optional parameter. The parameter is up to 500 characters long; the characters can be in uppercase, lowercase, or both. The parameter must be enclosed in single quotes.

If the physical file system specified does not expect a PARM operand, it ignores all PARM operands. Refer to the documentation for the specific physical file system for valid entry point names.

SYNC(t), NOWRITEPROTECT, NOSPARSE, FSFULL, and **SYNCRESERVE** are valid only when ENTRYPOINT is GFUAINIT.

Note: If a syntax error is found in any of these parameters (**SYNC(t), NOWRITEPROTECT, NOSPARSE, FSFULL,** and **SYNCRESERVE**), an error message is issued and all five parameters are set to the default values.

SYNC(t)

t specifies the number of seconds used as a default for the sync daemon interval. When the sync daemon is active, the meta data for a file system is hardened. Setting *t* to 0 indicates that the file system should harden meta data synchronously with syscall requests.

Sync interval values are rounded up to the next 30-second value. For example, specifying 31 seconds results in a sync interval of 60 seconds.

The maximum value that can be specified for *t* is 65535. Values between 65535 and 99999 are rejected.

A value of 99999 specifies that no sync daemon intervals are specified, and thus, the meta data is not hardened.

Default: 60 seconds

NOWRITEPROTECT

– This keyword overrides the WRITEPROTECT function. When **NOWRITEPROTECT** is specified, the file system is not protected from being read/write mounted by multiple systems simultaneously. Read/write mounting by multiple systems corrupts the file system.

  Extreme care should be taken when specifying this keyword. It should only be used when there is no possibility of the file system being mounted by multiple systems.

  Use of the **NOWRITEPROTECT** keyword avoids an additional file system read operation that is required at Sync time to support the WRITEPROTECT function.

– **Default:** WRITEPROTECT

NOSPARSE(DUMP|LOGREC)

– When NOSPARSE is specified on the MOUNT statement, HFS will not allow any files in that file system to be sparse. A file becomes sparse when all of the data cannot be written. For example, suppose we are only able to write the first 10,000 bytes of a file, and then the system has to lseek out to offset 50,000 and resume writing from there. The file is considered sparse because bytes 10,000-50,000 were never written to the file. If the user attempts to read bytes 10,000 to 50,000, binary 0's will be returned as the value. NOSPARSE handles this by causing the HFS to create a dump or a LOGREC record when either of the following situations occur:

  - HFS attempts to read metadata from disk for a file and detects that the subject file is sparse, or
  - an application attempts to write to a page beyond the end of the file, causing the file to become sparse.

– DUMP will cause HFS to create a dump. Only one dump will be created for each of the possible reason codes while a file system is mounted. DUMP is the default if you specify NOSPARSE without the DUMP or LOGREC keywords.
– LOGREC will cause HFS to write a LOGREC record instead of creating a dump.
– **Default:** DUMP

FSFULL(threshold,increment)

threshold specifies the percentage of the HFS capacity at which an operator message is generated. The default is 100%.

increment specifies the percentage of change above the HFS capacity at which an operator messages is generated. Messages are generated by either an increase or decrease greater than *increment*. The default is 5%.

You can specify *threshold* and *increment* values for all HFS file systems. The values can also be set on the MOUNT command for a specific file system. Parameters on the MOUNT command override parmlib values. If no values are specified in either place, no threshold checking is done. If a threshold value is specified but no increment is given, the increment defaults to 5%. The increment value applies both to upgrading the message when the file system continues to fill and to removing the message when more space becomes available due to either deleting files, or to extending the file system. The values are in terms of percent full. The values applied to a file system can be changed only when the file system is mounted.

SYNCRESERVE(nn)

– This keyword controls the number of pages HFS should reserve for sync processing of the file system metadata.*nn* represents the percentage of the file system space which is to be reserved for the sync shadow write mechanism. *nn* is a decimal number between 1 and 50. There is no reason to ever reserve more than 50% of the file system space, because the reserved space must always be less than the actual index size and the index size plus the reserved space cannot be greater than the file system space.
– When this parm is specified on the MOUNT statement, it will override the HFS internal reserved page estimation algorithm. Therefore, this parameter should only be used if it is found that the internal algorithm is not providing the desired results.

SETUID|NOSETUID

SETUID specifies that the **setuid()** and **setgid()** mode bit on an executable file will be supported.

NOSETUID specifies that the **setuid()** and **setgid()** mode bit on an executable file will not be supported. The UID or GID will not be changed when the program is executed and the APF and Program Control extended attributes are not honored. The entire HFS is uncontrolled.

Default: SETUID

AUTOMOVE|NOAUTOMOVE

The AUTOMOVE|NOAUTOMOVE parameters apply only in a sysplex where systems are participating in shared file system. The parameters indicate what happens to the ownership of the file system when a shutdown, PFS termination, dead system takeover, or file system move occurs.

AUTOMOVE indicates that ownership of the file system automatically changes to another system participating in shared file system. NOAUTOMOVE indicates that ownership of the file system is not moved if the owning system goes down; as a result, the file system becomes inaccessible.

Note: When specifying NOAUTOMOVE, though the file system becomes inaccessible when the owning system goes down, it still exists in the file system hierarchy. The file system will remain unowned until the original owing system re-IPLs.

IBM suggests that you use AUTOMOVE for the sysplex root file system and the version file system. For file systems that are associated with a single system, specify UNMOUNT; this includes **/etc**, **/tmp**, **/var**, **/dev**, and other system-specific file systems. For descriptions of the sysplex root, system-specific, and version file systems, see "Shared file system in a Sysplex" in *z/OS UNIX System Services Planning*.

Default: AUTOMOVE

SYSNAME(sysname)
For a description, see SYSNAME on the MOUNT statement. To ensure that the root is always available, use the default.

Default: The name of the system that the command is processed on.

TAG (NOTEXT|TEXT,*ccsid*)
Specifies whether implicit file tags are assigned to untagged files in the mounted file system. File tagging controls whether a file's data can be converted during file reading and writing. "Implicit" in this case means that the tag is not permanently stored with the file. Instead, the tag is associated with the file during reading and writing, or when stat() type functions are issued. Either **TEXT**, or **NOTEXT**, and *ccsid* must be specified when **TAG** is specified.

NOTEXT specifies that none of the files in the file system are automatically converted during file reading and writing.

TEXT specifies that each untagged file is implicitly marked as containing pure text data that can be converted.

ccsid names the coded character set identifier to be implicitly set for the untagged file. *ccsid* is specified as a decimal value from 0 to 65536. However, when **TEXT** is specified, the values of 0 and 65536 are illegal because those values imply no conversion. Other than this, the value is not checked as being valid and the corresponding code page is not checked as being installed.

For example:

- TAG(TEXT,819) identifies text files containing ASCII (ISO-8859–1) data.
- TAG(TEXT,1047) identifies text files containing EBCDIC ((ISO-1047) data.
- TAG(NOTEXT,65536) tags files as containing binary or unknown data.
- TAG(NOTEXT,0) is the equivalent of not specifying the TAG parameter.
- TAG(NOTEXT,273) tags file with the German code set (ISO-273), but is ineligible for automatic conversion.

Default: NOTEXT

MKDIR('pathname')

Specifies the name of a directory that the system will create dynamically after the file system has been successfully mounted. This allows the system to create mountpoints that can be referenced by subsequent MOUNT statements. MKDIR is an optional keyword.

You can specify more than one MKDIR keyword on each ROOT statement. The directories will be created in the order they are listed.

You must specify a relative path name; the path name cannot start with a slash (/). Enclose the path name in single quotes.

The path name is relative to the file system mount point specified on the MOUNTPOINT keyword.

The path name can contain intermediate directories, but these intermediate directories must already exist in the file system hierarchy. If these intermediate directories do not exist, you can use additional MKDIR keywords to create the necessary intermediate directories.

The directory to be created must reside in a file system that is has been mounted as RDWR. The permission bits for the created directory will be 755. The UID and GID will be inherited from this directory's parent. These attributes will be overlaid when this directory is used as a mount point.

We recommend that you do not use the MKDIR keyword for file systems that mount asynchronously, such as the NFS file system. When a file system will be mounted asynchronously, message BPXF025I is issued to the system log. Similarly, do not use MKDIR with the SYSNAME keyword, when SYSNAME identifies a remote system to perform the mount. The results are unpredictable.

A failure to create a directory does not cause a failure of the mount. A message is written to the system log when a problem occurs during the creation of the directory. If the directory already exists, no message is written.

Note: MKDIR will only work on systems that are at z/OS Version 1 Release 5 level or higher. If you are using MKDIR in a sysplex that is sharing a common BPXPRMxx member, make sure that all systems are at least at the z/OS V1R5 level.

| **MOUNT FILESYSTEM('fsname') DDNAME(ddname) TYPE(type_name)**
| **MOUNTPOINT('pathname') MODE(access) PARM('parameter')**
| **TAG(NOTEXT|TEXT,*ccsid*) SETUID|NOSETUID SECURITY|NOSECURITY**
| **AUTOMOVE[(INCLUDE,sysname1,sysname2,...,[sysname*n*|*])**
| **|(EXCLUDE,sysname1,sysname2,...,sysname*n*)]**
|NOAUTOMOVE|UNMOUNT
SYSNAME(sysname) MKDIR('pathname')

Specifies a file system that z/OS UNIX is to logically mount onto the root file system or another file system.

Mount statements are processed in the sequence in which they appear. If they are cascading, the system will mount the first file system first. Make sure that a mount point exists before the file system is mounted. If you mount a file system over an existing directory containing files, you will cover up the existing files.

If a MOUNT statement uses a DDNAME parameter to identify the HFS data set, allocate that HFS data set in the OMVS cataloged procedure. See the section

on customizing the OMVS cataloged procedure to run the kernel initialization program in the "Customizing z/OS UNIX" chapter in *z/OS UNIX System Services Planning*.

The MOUNT statement is optional; the BPXPRMxx member can contain one or more MOUNT statements.

The MOUNT parameters are:

FILESYSTEM('fsname')
The name of the file system. The name must be unique in the system.

Either FILESYSTEM or DDNAME is required; do not specify both. The name is 1 to 44 characters; the characters can be in uppercase, lowercase, or both. The name must be enclosed in single quotes. An HFS data set name must conform to the rules of MVS data set names.

DDNAME(ddname)
The ddname on the JCL DD statement that defines the file system. To use the DDNAME parameter, a DD statement for the HFS data set containing the mountable file system should be placed in the OMVS cataloged procedure.

Either FILESYSTEM or DDNAME is required; do not specify both. The name is 1 to 8 characters; the system converts the ddname to uppercase.

TYPE(type_name)
Specifies the name of a file system type identified in a FILESYSTYPE statement. The TYPE(type_name) parameter must be the same as the TYPE(type_name) parameter on a FILESYSTYPE statement.

TYPE is a required parameter. The name is 1 to 8 characters; the system converts the name to uppercase.

MOUNTPOINT('pathname')
Specifies the pathname, or a symlink that resolves to the pathname of the directory onto which the file system is to be mounted.

Mount point restrictions are:

- The mount point must be a directory.
- Any files in the directory are not accessible while the file system is mounted.
- Only one mount can be active at any time for a mount point.
- A file system can be mounted at only one directory at any time.

MOUNTPOINT is required. The pathname is up to 1023 characters long; the characters can be in uppercase, lowercase, or both. The pathname must be enclosed in single quotes.

MODE(access)
Specifies access to the mounted file system by all users:
- READ: Users can only read the file system being mounted.
- RDWR: Users can read and write in the file system being mounted.

Default: RDWR

TAG (NOTEXT|TEXT,*ccsid*)
Specifies whether implicit file tags are assigned to untagged files in the mounted file system. File tagging controls whether a file's data can be converted during file reading and writing. ″Implicit″ in this case means that the tag is not permanently stored with the file. Instead, the tag is associated

with the file during reading and writing, or when stat() type functions are issued. Either **TEXT**, or **NOTEXT**, and *ccsid* must be specified when **TAG** is specified.

NOTEXT specifies that none of the files in the file system are automatically converted during file reading and writing.

TEXT specifies that each untagged file is implicitly marked as containing pure text data that can be converted.

ccsid names the coded character set identifier to be implicitly set for the untagged file. *ccsid* is specified as a decimal value from 0 to 65536. However, when **TEXT** is specified, the values of 0 and 65536 are illegal because those values imply no conversion. Other than this, the value is not checked as being valid and the corresponding code page is not checked as being installed.

For example:

- TAG(TEXT,819) identifies text files containing ASCII (ISO-8859–1) data.
- TAG(TEXT,1047) identifies text files containing EBCDIC ((ISO-1047) data.
- TAG(NOTEXT,65536) tags files as containing binary or unknown data.
- TAG(NOTEXT,0) is the equivalent of not specifying the TAG parameter.
- TAG(NOTEXT,273) tags file with the German code set (ISO-273), but is ineligible for automatic conversion.

Default: NOTEXT

PARM('parameter')

Provides a parameter to be passed directly to the file system type. The parameter format and content are specified by the file system type.

PARM is an optional parameter. The parameter is up to 500 characters long; the characters can be in uppercase, lowercase, or both. The parameter must be enclosed in single quotes.

If the physical file system specified does not expect a PARM operand, it ignores all PARM operands. Refer to the documentation for the specific physical file system for valid entry point names.

SYNC(t), NOWRITEPROTECT, NOSPARSE, FSFULL, and **SYNCRESERVE** are valid only when ENTRYPOINT is GFUAINIT.

Note: If a syntax error is found in any of these parameters (**SYNC(t), NOWRITEPROTECT, NOSPARSE, FSFULL,** and **SYNCRESERVE**), an error message is issued and all five parameters are set to the default values.

SYNC(t)

t specifies the number of seconds used as a default for the sync daemon interval. When the sync daemon is active, the meta data for a file system is hardened. Setting *t* to 0 indicates that the file system should harden meta data synchronously with syscall requests.

Sync interval values are rounded up to the next 30-second value. For example, specifying 31 seconds results in a sync interval of 60 seconds.

The maximum value that can be specified for *t* is 65535. Values between 65535 and 99999 are rejected.

A value of 99999 specifies that no sync daemon intervals are specified, and thus, the meta data is not hardened.

Default: 60 seconds

NOWRITEPROTECT

– This keyword overrides the WRITEPROTECT function. When **NOWRITEPROTECT** is specified, the file system is not protected from being read/write mounted by multiple systems simultaneously. Read/write mounting by multiple systems corrupts the file system.

  Extreme care should be taken when specifying this keyword. It should only be used when there is no possibility of the file system being mounted by multiple systems.

  Use of the **NOWRITEPROTECT** keyword avoids an additional file system read operation that is required at Sync time to support the WRITEPROTECT function.

– **Default:** WRITEPROTECT

NOSPARSE(DUMP|LOGREC)

– When NOSPARSE is specified on the MOUNT statement, HFS will not allow any files in that file system to be sparse. A file becomes sparse when all of the data cannot be written. For example, suppose we are only able to write the first 10,000 bytes of a file, and then the system has to lseek out to offset 50,000 and resume writing from there. The file is considered sparse because bytes 10,000-50,000 were never written to the file. If the user attempts to read bytes 10,000 to 50,000, binary 0's will be returned as the value. NOSPARSE handles this by causing the HFS to create a dump or a LOGREC record when either of the following situations occur:

  - HFS attempts to read metadata from disk for a file and detects that the subject file is sparse, or
  - an application attempts to write to a page beyond the end of the file, causing the file to become sparse.

– DUMP will cause HFS to create a dump. Only one dump will be created for each of the possible reason codes while a file system is mounted. DUMP is the default if you specify NOSPARSE without the DUMP or LOGREC keywords.

– LOGREC will cause HFS to write a LOGREC record instead of creating a dump.

– **Default:** DUMP

FSFULL(threshold,increment)

threshold specifies the percentage of the HFS capacity at which an operator message is generated. The default is 100%.

increment specifies the percentage of change above the HFS capacity at which an operator messages is generated. Messages are generated by either an increase or decrease greater than *increment*. The default is 5%.

You can specify *threshold* and *increment* values for all HFS file systems. The values can also be set on the MOUNT command for a specific file system. Parameters on the MOUNT command override parmlib values. If no values are specified in either place, no threshold checking is done. If a threshold value is specified but no increment is given, the increment defaults to 5%. The increment value applies both to upgrading the message when the file system continues to fill and to removing the message when more space

becomes available due to either deleting files, or to extending the file system. The values are in terms of percent full. The values applied to a file system can be changed only when the file system is mounted.

SYNCRESERVE(nn)

– This keyword controls the number of pages HFS should reserve for sync processing of the file system metadata.*nn* represents the percentage of the file system space which is to be reserved for the sync shadow write mechanism. *nn* is a decimal number between 1 and 50. There is no reason to ever reserve more than 50% of the file system space, because the reserved space must always be less than the actual index size and the index size plus the reserved space cannot be greater than the file system space.

– When this parm is specified on the MOUNT statement, it will override the HFS internal reserved page estimation algorithm. Therefore, this parameter should only be used if it is found that the internal algorithm is not providing the desired results.

SETUID|NOSETUID

SETUID specifies that the **setuid()** and **setgid()** mode bit on an executable file will be supported.

NOSETUID specifies that the **setuid()** and **setgid()** mode bit on an executable file will not be supported. The UID or GID will not be changed when the program is executed and the APF and Program Control extended attributes are not honored. The entire HFS is uncontrolled.

Default: SETUID

SECURITY|NOSECURITY

SECURITY specifies that security checks should be performed.

NOSECURITY specifies that security checks should not be performed.

Default: SECURITY

AUTOMOVE[(INCLUDE,sysname1,sysname2,...,[sysnameN|*])
|(EXCLUDE,sysname1,sysname2,...,sysnameN)]
| NOAUTOMOVE | UNMOUNT

The AUTOMOVE, NOAUTOMOVE and UNMOUNT parameters apply only in a sysplex where systems are participating in shared file system. The parameters indicate what happens to the ownership of the file system when a shutdown, PFS termination, dead system takeover, or file system move occurs.

AUTOMOVE indicates that ownership of the file system automatically changes to another system participating in shared file system. You can specify AUTOMOVE on its own to allow the system to randomly select a new owner for the file system. You can direct the system how to choose a new owner for the file system by using the indicators INCLUDE (I) or EXCLUDE (E).

Specify INCLUDE with a system list to provide a prioritized list of systems to which the file system can be moved should the owning system go down. For example, AUTOMOVE(INCLUDE,SYS1,SYS4,SYS9) tells the system that the file system can be moved to SYS1, SYS4, or SYS9, in that order, or AUTOMOVE(INCLUDE,SYS1,SYS4,*) tells the system that the file system can be moved to SYS1 or SYS4, in that order, then to any other

available system. This selection is not totally random; rather, MVS attempts to move the file system to a new server system in which the file system is actively in use.

Notes:

1. If specified, the asterisk must be listed **last** or listed as the **only** system name in the INCLUDE system list. The asterisk is not supported prior to z/OS Version 1 Release 6.
2. Be certain all systems in the sysplex are at z/OS Version 1 Release 6 or higher. Specifying AUTOMOVE with an asterisk in the INCLUDE system list in a mixed (down-level) environment will produce unpredictable results.

Specify EXCLUDE with a system list to provide a list of systems to which the file system cannot be moved. For example, AUTOMOVE(EXCLUDE,SYS3,SYS5,SYS7) tells the system that the file system can be moved to any system except SYS3, SYS5, and SYS7. If the file system cannot be moved as you have directed in the system list, the file system will be unmounted when the owning system goes down.

NOAUTOMOVE indicates that ownership of the file system is not moved in some situations; as a result, the file system becomes inaccessible.

Note: When specifying NOAUTOMOVE, although the file system becomes inaccessible when the owning system unexpectedly goes down, it still exists in the file system hierarchy. The file system will remain unowned until the original owning system re-IPLs. Changing the AUTOMOVE value in BPXPRM*xx* of an unowned file system before re-IPLing will not change the AUTOMOVE value of this file system since it is already mounted. To change the AUTOMOVE value, the file system must be unmounted before the IPL.

UNMOUNT indicates that the file system should be unmounted in some situations.

See *z/OS UNIX System Services Planning* for more information about the behavior of the AUTOMOVE options.

Guidelines:

1. Use AUTOMOVE for the version file system and the sysplex root file system.
2. For file systems that are associated with a single system, specify UNMOUNT; this includes **/etc, /tmp, /var, /dev,** and the system-specific file system. For descriptions of the sysplex root, system-specific, and version file systems, see the chapter on sharing file systems in a sysplex in *z/OS UNIX System Services Planning*.
3. For sysplex-unaware file systems that are mostly exported by the DFS or SMB server to their remote clients, consider specifying NOAUTOMOVE on the MOUNT statement. By doing so, the file systems will not change ownership if the system is suddenly recycled and they will be available for automatic re-export by DFS or SMB.

   Tip: Consider specifying NOAUTOMOVE because a file system can only be exported by the DFS or SMB server at the system that owns the file system. A file system can only be exported by the DFS or SMB server at the system that owns the file system. Once a file system has

been exported by DFS, it cannot be moved until it has been unexported by DFS. The same holds true of file systems exported by SMB. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS or SMB clients. When an owning system recycles and a file system exported by DFS or SMB has been taken over by one of the other systems, DFS or SMB cannot automatically re-export that file system. When an owning system is recycled and an exported file system has been taken over by one of the other systems, that file system will not be automatically reexported. The file system will have to be moved from its current owner back to the original system, the one that has just been recycled, and then exported again.

Default: AUTOMOVE

SYSNAME(sysname)
For systems participating in shared file system, SYSNAME specifies the particular system on which a mount should be performed. This system will then become the owner of the file system mounted. This system must be IPLed with SYSPLEX(YES).

Default: The name of the system, if IPLed with SYSPLEX(YES), that the mount is processed on.

Note: In OS/390 R9 and later, to ensure that the root is always available, use the defaults for SYSNAME and AUTOMOVE.

MKDIR('pathname')
Specifies the name of a directory that the system will create dynamically after the file system has been successfully mounted. This allows the system to create mountpoints that can be referenced by subsequent MOUNT statements. MKDIR is an optional keyword.

You can specify more than one MKDIR keyword on each ROOT statement. The directories will be created in the order they are listed.

You must specify a relative path name; the path name cannot start with a slash (/). Enclose the path name in single quotes.

The path name is relative to the file system mount point specified on the MOUNTPOINT keyword.

The path name can contain intermediate directories, but these intermediate directories must already exist in the file system hierarchy. If these intermediate directories do not exist, you can use additional MKDIR keywords to create the necessary intermediate directories.

The directory to be created must reside in a file system that has been mounted as RDWR. The permission bits for the created directory will be 755. The UID and GID will be inherited from this directory's parent. These attributes will be overlaid when this directory is used as a mount point.

We recommend that you do not use the MKDIR keyword for file systems that mount asynchronously, such as the NFS file system. When a file system will be mounted asynchronously, message BPXF025I is issued to the system log. Similarly, do not use MKDIR with the SYSNAME keyword, when SYSNAME identifies a remote system to perform the mount. The results are unpredictable.

A failure to create a directory does not cause a failure of the mount. A message is written to the system log when a problem occurs during the creation of the directory. If the directory already exists, no message is written.

Note: MKDIR will only work on systems that are at z/OS Version 1 Release 5 level or higher. If you are using MKDIR in a sysplex that is sharing a common BPXPRMxx member, make sure that all systems are at least at the z/OS V1R5 level.

For additional information, see **MOUNT** in *z/OS UNIX System Services Planning*.

NETWORK DOMAINNAME(sockets_domain_name) DOMAINNUMBER(sockets_domain_number) MAXSOCKETS(number) TYPE(type_name) INADDRANYPORT(starting_port_number) INADDRANYCOUNT(number_of_ports_to_reserve)

Specifies that a socket physical file system domain should be readied for use. The TYPE in this statement matches the TYPE on the previous FILESYSTYPE statement.

Use the SETOMVS RESET command to dynamically change the MAXSOCKET value or add a new NETWORK. To make a permanent change, edit the BPXPRMxx member used for IPLs. For more information, see “Dynamically Adding FILESYSTYPE Statements in BPXPRMxx“ in *z/OS UNIX System Services Planning*.

Provide a NETWORK statement for each socket file system domain to be initialized.

- For AF_UNIX file systems, always include a FILESYSTYPE statement specifying ENTRYPOINT(BPXTUINT) and a NETWORK statement with a matching TYPE, usually TYPE(UDS), on both.
- For TCP/IP sockets, always include a FILESYSTYPE statement specifying ENTRYPOINT(EZBPFINI) and a NETWORK statement with a matching TYPE, usually TYPE(INET), on both.
- To activate an Internet Protocol Version 6 (IPv6) socket on a system, you must configure both the AF_INET domain and the AF_INET6 domain. You cannot code a NETWORK statement for domain name AF_INET6 without coding a NETWORK statement for domain name AF_INET.
- For CINET sockets, include a FILESYSTYPE statement with ENTRYPOINT (BPXTCINT) and a NETWORK statement with a matching TYPE, usually TYPE(CINET), that specifies INADDRANYPORT and INADDRANYCOUNT. See “Specifying INADDRANYPORT and INADDRANYCOUNT“ in *z/OS UNIX System Services Planning* for more information.

DOMAINNAME(sockets_domain_name)

The 1 to 16 character name by which this socket file system domain is to be known.

DOMAINNUMBER(sockets_domain_number)

A number that matches the value defined for this domain name. The currently supported values for this field are:

1 AF_UNIX
2 AF_INET
19 AF_INET6

The following table shows some supported domain names, domain numbers, and their associated entry point names. See the documentation

for the physical file system you are using to get the correct entry point name.

Table 26. Supported Domains

Domain name	Domain number	Entry point
AF_UNIX	1	BPXTUINT
AF_INET	2	EZBPFINI, BPXTCINT
AF_INET6	19	EZBPFINI, BPXTCINT

MAXSOCKETS(nnnnn)
Specifies the maximum number of sockets supported by this file system for this address family. You can specify a value from 0 to 16777215. This is an optional parameter. The maximum value that this field can have is defined by each domain. If a value larger than the maximum is specified, an informational message is issued and the value used is the maximum. If this parameter is omitted, a default value of 100 is used.

Note: Ensure that this number is large enough for socket connections for all applications using your z/OS UNIX environment. This upper limit is set when the NETWORK statement is processed during IPL. It can only be changed if the NETWORK statement is changed using the SETOMVS RESET command.

When activating IPv6 on a system, you can specify separate MAXSOCKETS values for domains AF_INET and AF_INET6. If you do not specify a MAXSOCKETS value for the AF_INET6 domain, the default will be the MAXSOCKETS value specified or defaulted to for the AF_INET domain.

If you are using AnyNet® sockets over SNA or AF_UNIX, a high MAXSOCKETS value may use too many resources. You should use a low value instead.

TYPE(type_name)
Specifies the name of a file system type identified in a FILESYSTYPE statement. The TYPE(type_name) must be the same as the TYPE(type_name) parameter on a FILESYSTYPE statement.

TYPE is a required parameter. The name is 1 to 8 characters; the system converts the name to uppercase.

INADDRANYPORT(starting_port_number)
Specifies the starting port number for the range of port numbers that the system reserves for use with PORT 0, INADDR_ANY binds. This value is only needed for CINET.

Value Range: *starting_port_number* is a decimal value from 1024 to 65534. Ports 1 — 1023 are well-known ports that cannot be reserved for use with PORT 0, INADDR_ANY binds.

Default: If neither INADDRANYPORT or INADDRANYCOUNT is specified, the default for INADDRANYPORT is 63000. Otherwise, no ports are reserved (0).

Note: If you do not want to support INADDRANY with CINET, you should specify INADDRANYPORT(*xx*), where *xx* is a valid value, without specifying INADDRANYCOUNT.

Note: When activating IPv6 on a system, the INADDRANYPORT is shared across domains. The INADDRANYPORT value is taken from the NETWORK statement for the AF_INET domain. Any INADDRANYPORT value specified for the AF_INET6 domain is ignored.

INADDRANYCOUNT(number_of_ports_to_reserve)
Specifies the number of ports that the system reserves, starting with the port number specified in the INADDRANYPORT parameter. This value is only needed for CINET.

Value Range: *number_of_ports_to_reserve* is a decimal value from 1 to 4000.

Default: If neither INADDRANYPORT or INADDRANYCOUNT is specified, the default for INADDRANYCOUNT is 1000. Otherwise, no ports are reserved (0).

RESOLVER_PROC(*procname*|DEFAULT|NONE)
Specifies how the resolver address space is processed during z/OS UNIX initialization. The resolver is used by TCP/IP applications for name-to-address or address-to-name resolution. In order to create a resolver address space, a system must be configured with an AF_INET or AF_INET6 domain.

procname is the name of the address space for the resolver and the procedure member name in the appropriate proclib. *procname* is one to eight characters long. The procedure must reside in a data set that is specified by the MSTJCLxx parmlib member's IEFPDSI DD card specification.

DEFAULT causes an address space named RESOLVER to start, using the system default procedure of IEESYSAS. The address space is started with SUB=MSTR so that it will run under the MASTER address space instead of the JES address space.

NONE specifies that no address space is to be started. If you are using z/OS Communications Server IP the resolver must be started before TCP/IP can be started. TCP/IP will not initialize until the resolver address space is started.

SUBFILESYSTYPE NAME(transport_name) TYPE(type_name) ENTRYPOINT(entry_name) PARM('parameter') DEFAULT
Specifies an AF_INET or AF_INET6 physical file system that is to run underneath the CINET socket file system. The TYPE() value is usually CINET and matches the TYPE operand on a previous FILESYSTYPE and NETWORK statement. In the case of TCP/IP, the NAME() value is the procname. The system attaches the EZBPFINI load module during initialization, and this file system should be used as the default INET physical file system.

The SUBFILESYSTYPE statement is associated with its corresponding FILESYSTYPE and NETWORK statements by matching the value specified in the TYPE operand.

The value specified on all of the TYPE operands must match, but can be any 1- to 8-character value. The value specified on the NAME parameter on the SUBFILESYSTYPE statement is the name to be used by the physical file system when it is initialized. The first character of the NAME parameter must be non-numeric.

For SecureWay® Communications Server, the SUBFILESYSTYPE statement must match the TCPIPJOBNAME of that stack. See "Customizing the File System Statements on the BPXPRMxx Member" in *z/OS UNIX System Services Planning* for more details.

New SUBFILESYSTYPE statements can be added dynamically. However, you cannot dynamically change (or delete) a value. For more information, see "Dynamically Adding FILESYSTYPE Statements in BPXPRMxx" in *z/OS UNIX System Services Planning*.

The parameters are:

NAME(transport_name)
Specifies the name that identifies this file system to the CINET physical file system.

NAME is a required parameter. The name is 1 to 8 characters with the first character non-numeric; the system converts the name to uppercase. The value specified by the NAME parameter on the SUBFILESYSTYPE statement is the name that the physical file system uses to identify itself when it is initialized. For example, for TCP/IP, this is the starting procedure name.

TYPE(type_name)
Specifies the name of the CINET file system type identified in a FILESYSTYPE statement. The TYPE(type_name) parameter must be the same name that was used for the TYPE(type_name) parameter on the FILESYSTYPE statement for the CINET physical file system.

TYPE is a required parameter. The name is 1 to 8 characters; the system converts the name to uppercase.

ENTRYPOINT(entry_name)
Specifies the name of the load module containing the entry point into the file system type.

ENTRYPOINT is a required parameter. The name is 1 to 8 characters; the system converts the name to uppercase.

PARM('parameter')
Provides a parameter to be passed to the transport driver. The parameter format and content are specified by the file system receiving the data.

PARM is an optional parameter. The parameter is up to 500 characters long; the characters can be in uppercase, lowercase, or both. If the characters are not all in uppercase, the parameter must be enclosed in single quotes.

If the physical file system specified does not expect a PARM operand, it ignores all PARM operands. Refer to the documentation for the specific physical file system for valid entry point names.

DEFAULT
Identifies this file system as the default CINET file system.

DEFAULT is an optional parameter. If it is not specified, the file system specified in the first SUBFILESYSTYPE statement found in the parmlib member is designated as the default. See "Setting Up for CINET AF_INET Sockets" in *z/OS UNIX System Services Planning* for more information about the use of the DEFAULT parameter.

For additional information, see **SUBFILESYSTYPE** in *z/OS UNIX System Services Planning*.

STARTUP_PROC
This statement specifies a 1-to-8-character name of a started JCL procedure

that initializes the kernel. The name specified in this statement must exist on the system before IPL or errors will occur.

Using a started procedure other than OMVS is **strongly discouraged**. If you want to change the value of STARTUP_PROC, you will have to edit the BPXPRMxx member and then re-IPL. You cannot use the SET OMVS or SETOMVS command to change the value.

If you decide to use a started procedure other than OMVS:

- The replacement started procedure must also be a single jobstep procedure that invokes the BPXINIT program (EXEC PGM=BPXINIT). If it invokes any other program, the OMVS initialization will fail.
- Change the procedure name in the RACF started procedures table or the definitions in the STARTED Class. See “Preparing the RACF Security Program“ in *z/OS UNIX System Services Planning*.

Note: Renaming OMVS to some other value may affect the setup of other products such as TCP/IP.

Default: STARTUP_PROC(OMVS).

STARTUP_EXEC

STARTUP_EXEC names a REXX exec that does application environment initialization for z/OS UNIX. This statement is optional; if it is specified, the BPXOINIT process will not run **/etc/init**. The startup exec is typically used by an installation that does not have an HFS, but is using a TFS for a file system. It can be used to populate the TFS with any directories and files that are needed. It is specified as:

```
STARTUP_EXEC('Dsname(Memname)',SysoutClass)
```

where:

- Dsname is a 1-to-44-character valid data set name.
- Memname is a 1-to-8-character valid REXX exec member.
- SysoutClass is 1 character and is alphanumeric and specifies the sysout class that the REXX exec will run under. Specifying SysoutClass is optional.

If you want to change the value of STARTUP_EXEC, you will have to edit the BPXPRMxx member and then reIPL. You cannot use the SET OMVS or SETOMVS command to change the value.

Default: There is no default value for STARTUP_EXEC.

RUNOPTS(’string’)

Specifies the _CEE_RUNOPTS environment variable used when z/OS UNIX initialization invokes **/etc/init** or **/usr/sbin/init**. z/OS UNIX passes the _CEE_RUNOPTS value and all programs invoked from **/etc/rc** to the shell.

If you want to change the value of RUNOPTS, you will have to edit the BPXPRMxx member and then re-IPL. You cannot use the SET OMVS or SETOMVS command to change the value. After the value is specified in BPXPRMxx, you can use one of the following methods to change this string:

- The system is re-IPLed with a new BPXPRMxx RUNOPTS string.
- The user or installation sets _CEE_RUNOPTS in **/etc/rc** or **/etc/init.config**.
- A program or shell script sets _CEE_RUNOPTS.

If you do not specify a value for RUNOPTS, the RUNOPTS string or _CEE_RUNOPTS environment variable is not provided.

The TSO/E OMVS command uses the specified options as the Language Environment® run-time options, by default.

The setting of RUNOPTS has no effect on BPXBATCH jobs.

Use the RUNOPTS parameter only when using RTLS. Before using RTLS, you must set up FACILITY profiles as documented in the **CSVRTLxx** description.

Specifying the RUNOPTS parameter causes the kernel to set the **_CEE_RUNOPTS** environment variable when starting **/etc/init**, or when the TSO/E OMVS command is entered. This environment variable is normally propagated to subsequent processes (such as **/etc/init** to **/bin/sh** to **/etc/rc** to **/bin/inetd** to **/bin/rlogind** to **/bin/sh** for shell users).

To do this, you must make sure that any other steps in the flow (such as export statements in **/etc/rc**) do not overwrite the value of **_CEE_RUNOPTS**. If additional run-time options are needed, they should be concatenated to the old value of **_CEE_RUNOPTS**.

Value Range: From 1 to 250 characters.

Default: No RUNOPTS string or _CEE_RUNOPTS environment variable is provided.

Restrictions:

- The string must be enclosed in parentheses and quotes (”).
- An empty string (’ ’) is not valid.
- Although all characters are allowed, nulls, slashes (/), unbalanced SO/SI, and unbalanced parentheses and quotes cause unpredictable problems in areas such as the TSO/E OMVS command.

For more information on specifying RUNOPTS strings, see “Customizing the BPXPRMxx Parmlib Member“ in *z/OS UNIX System Services Planning*.

SYSCALL_COUNTS(YES/NO)

Specifies that syscall counts are to be accumulated in internal kernel data areas so that the RMF data gatherer can record the information.

If you specify YES, the path length for the most frequently used z/OS UNIX system calls is increased by more than 150 instructions. This setting will also cause the reporting of CPU time for z/OS UNIX to be more accurate. This will be reflected in the output from the BPX1TIM, BPX1GPS, BPX1GTH, and BPX1RMG services and from BPXESMF.

Default: NO

Use the SETOMVS or SET OMVS command to dynamically change the value of SYSCALL_COUNT. To make a permanent change, edit the BPXPRMxx member used for IPLs.

MAXQUEUEDSIGS(nnnnnn)

Specifies the maximum number of signals that z/OS UNIX allows to be concurrently queued within a single process.

Value Range: *nnnnnn* is a decimal value from 1 to 100000.

Default: 1000

You can change the value of MAXQUEUEDSIGS dynamically using the SETOMVS or SET OMVS command. To make a permanent change, edit the BPXPRMxx member that will be used for future IPLs.

LIMMSG(NONE|SYSTEM|ALL)
Specifies how console messages that indicate when parmlib limits are reaching critical levels are to be displayed:

NONE No console messages are to be displayed when any of the parmlib limits have been reached.

SYSTEM
Console messages are to be displayed for all processes that reach system limits. In addition, messages are to be displayed for each process limit of a process if:
- The process limit or limits are defined in the OMVS segment of the owning User ID
- The process limit or limits have been changed with a SETOMVS PID=*pid*,*process_limit*

ALL Console messages are to be displayed for the system limits and for the process limits, regardless of which process reaches a process limit.

Default: NONE

AUTHPGMLIST('/etc/authfile')|NONE
Specifies the pathname of a hierarchical file system (HFS) file that contains the lists of APF-authorized pathnames and program names. If you do not specify a value for AUTHPGMLIST, or if you specify NONE, invocations of APF-authorized and program controlled programs will not be checked against a list of authorized programs or authorized pathnames. If you specify a pathname for AUTHPGMLIST parameter, the system checks this list during hfsload, exec and spawn processing. If the target program of an exec or spawn has an authorization code of 1 (AC=1), then that program name must appear in the authorized program list.

Use the SETOMVS or SET OMVS command to dynamically change the value of AUTHPGMLIST. To make a permanent change, edit the BPXPRMxx member that will be used for IPLs.

For additional information, see *z/OS UNIX System Services Planning*.

SWA(ABOVE/BELOW)
Specifies whether SWA control blocks should be allocated above or below the 16 megabyte line.

ABOVE
All SWA control blocks are to be allocated above the 16 megabyte line.

BELOW
All SWA control blocks are to be allocated below the 16 megabyte line.

Default: BELOW

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

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

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the products and/or the programs 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

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This book is intended to help the customer plan for, customize, operate, manage, and maintain a z/OS system with z/OS UNIX System Services (z/OS UNIX).

This book primarily documents intended Programming Interfaces that allow the customer to write programs that use z/OS UNIX.

This book also documents information that is NOT intended to be used as Programming Interfaces of z/OS UNIX. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

NOT Programming Interface information

End of NOT Programming Interface information

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AnyNet
DFS
Domino
IBM
Language Environment
MVS
Notes
OS/390
RACF
RMF
SecureWay
z/OS
zSeries

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.

Program Number: 5694-A01, 5655-G52

Printed in USA