Supplemental Notes

Shadow D/R File Replication Capability

Introduction

New File Replication Capability

Shadow D/R has long been a flexible and reliable database replication tool, but until recently it has only been able to replicate TurboImage databases, not flat files or KSAM files. However, release F.01.01 and later releases now provide this new capability. This document discusses the capabilitiy and how it can be used.

What is Database / File Replication?

File replication (including database replication) is the process of dynamically maintaining one or more replicas of a set of files at one or more other, possibly remote, locations. The files that are being replicated are called the *original*, *source*, *master*, or *primary* files. The replicas are called the *replica*, *destination*, *slave*, or *secondary* files. With Shadow D/R we almost always use the terms *primary* and *secondary*.

The secondary files can be placed on the same system as the primary ones, or a different system. Shadow D/R offers great flexibility in choice of destination locations, including the ability to change a file's parent group or directory.

Contents of this Document

This document is divided into four sections beyond this introduction.

Technical Overview

The Technical Overview provides an overview of what the new file replication capabilities are, how they are implemented, and how they are integrated into Shadow D/R. They do not provide all the details of implementing file replication. That is left to the next section.

Implementation

The Implementation section discusses issues regarding how one uses file replication; how to configure it, and how to control it.

Quick Start

Accompanying the Shadow D/R software is a set of demonstration configurations that allow one to quickly do a simple demonstration of file replication within the LPS account where the software is installed. This section discusses how to run this demonstration.

Appendices

The appendices provide details not covered in the main body of the document.

Appendix		Description
A	The File Capture Configuration File	Appendix A discusses in detail the format and meaning of the file that configures file replication.
B	What Intrinsics are Intercepted	Appendix B provides a list of the intrinsics that are intercepted to achieve file replication.
C	What Commands are Intercepted	Appendix C provides a list of the commands that are intercepted to achieve file replication.
D	What File Activity is Replicated	Appendix D provides information about what kind of activity against what kind of files is intercepted and replicated.
E	Limitations of File Replication	Appendix E discusses the limitations and known problems with file replication.
F	New Shadow D/R Files for File Replication	Appendix F lists the files that have been added for the F.01.01 and F.01.02 releases.

Technical Overview

Introduction

There are three aspects to replicating files on an MPE/iX system, or on any system: capturing the changes made to the primary files, transporting the changes to the system where the secondary files are being maintained, and posting the changes to the secondary files. Each of these aspects is discussed separately below.

Capturing the File Changes

On an MPE/iX system, files are changed by *processes*. The user files that we want to replicate are changed either by Command Interpreter (CI) processes, or by user processes. File changes are captured on the primary system in two ways: by UDCs for changes made by CI processes, and by Procedure Exits handlers for user processes. These two cases are discussed separately below.

The captured changes are placed into a user log maintained by MPE's user logging facility. This is the same facility used by TurboImage logging, on which Shadow D/R TurboImage replication relies. The file change information may be placed into the same log as TurboImage information, or a different one. However each user log requires its own separate Shadow D/R transport activity or mechanisms (there may be more than one Shadow D/R activity per log file).

Command Interpreter Processes

The command interpreter changes files in response to user commands, such as BUILD, PURGE, etc. Shadow D/R monitors such changes by means of User Defined Commands (UDCs) that have the same name as these commands. When, for instance, a user enters a BUILD command, the BUILD

UDC is invoked and does two things: executes the desired BUILD command, and logs it for replication use.

The implementer may choose at what level to activate these UDCs, though typically the system level is best, and last at that level. The reason is that other UDCs may issue commands such as BUILD; putting the Shadow D/R UDCs last assures that these commands will be seen.

User Processes

User processes change files by calling file system *intrinsics*, and by mapped file access. The file system intrinsics are often called indirectly, through the built-in input/output facilities of the language.

Shadow D/R monitors calls to file system intrinsics by means of *procedure exits*. This is a capability that MPE provides that allows software to monitor intrinsic calls. Shadow D/R has procedure exits *handlers*, which are functions that MPE invokes for execution both before and after the execution of an intrinsic. The handlers have access to the parameters of the intrinsic. This enables them to see what the intrinsics are doing, and to record relevant file change information to a log for transmission to the secondary systems.

Transporting the File Changes

The new Shadow D/R file replication capability uses the same transport mechanism that is used by the Shadow D/R TurboImage database replication capability. A SHADOWCP process within a job on the primary monitors the user log and transfers changes as they appear to a companion SHADOWCP process on the secondary.

A single Shadow D/R activity (all activity under control of a particular Shadow D/R configuration file) monitors only a single user log. However a single log can contain change data from as many as nine databases and any number of files.

Posting the File Changes

The transport process SHADOWCP on the secondary receives the file and database change information from the primary, and passes it on to a posting process SHADOWRP on the secondary. It is this process which posts the changes to the files and databases. The same processes are used for both file and database information. The only difference is that the database posting code resides in the SHADOWRP program file as it always has, whereas the file posting code resides in an XL file that resides in HANDLERS.LPS group.

Implementing Replication

This section will discuss some implementation considerations of the new file replication capabilities of Shadow D/R. It does not discuss database issues, which are covered in the Shadow D/R manual.

We always recommend that you purchase some consulting when you purchase the Shadow D/R product. It is a complex product, and typically it is being asked to do things of great importance, so the money will almost certainly be well spent.

Deciding What You Want To Do

As with most things, when dealing with replication you should start by deciding the goal. Of course replication can serve a number of possible goals, such as the following.

Load Balancing

By replicating databases and files to a different system, or perhaps just a different volume set, one can use the secondary files for read-only access such as on-line query or batch reporting. By removing such activity from the primary, the performance at the primary can be improved.

Online Backup

Replicating data in real-time to a different system or even a different volume set can provide protection against physical destruction of the data at the primary.

Disaster Recovery

If data is replicated to a backup system, that data can be available for use on the backup system in case anything goes wrong on the primary.

Once you have decided the goal, it should become clear what data needs to be replicated, and to where.

Synchronization Issues

MPE/iX does not provide a transaction mechanism that encompasses both database and file activity, nor does Shadow D/R. Thus, in the event of a crash on the primary, there is a problem. Even though Shadow D/R provides a mechanism for backing out incomplete database transactions, there is no mechanism for backing out corresponding file data. All file data that has made it to the secondary will remain posted. However it is likely that the situation is no different than what one already faces on the primary because of the lack of synchronization of database and file activity.

You have a choice of logging your file activity to the same user log as the database activity, or a different one. If you log to a different one, synchronization problems may be even more severe. Separate logs also means separate replication activities to start and stop. One activity might be started and the other stopped. Though Shadow D/R typically does a very good job of keeping posting up-to-date, even if both are started they might not both be at the same stage of posting. For these reasons we recommend that all activity related to a particular application activity be logged to the same user log.

For unrelated application activities, different logs can safely be used. Just keep in mind that different logs *require* separate shadow transport and posting activities.

The File Replication Configuration

You must decide what data will be logged, and to what user log. As discussed earlier in "Synchronization Issues," file data and database data that are part of the same application would typically be logged together.

Keep in mind that, just as a database is only logged to a single user log, so a file is only logged to one, the first one identified in the logging configuration.

Decide whether there is to be any file renaming to be done. Renaming means changing the secondary file name to be different from the corresponding primary file name. Shadow D/R provides facilities in its configurations to rename TurboImage databases and files.

Once all of these issues are decided, you are ready to create the file replication configuration file CAPTCONF.SHADOW.LPS. The format of this file is discussed in “Appendix A: The File Capture Configuration File” on page 8.

Monitoring Commands

The files that you want to replicate are almost certainly operated on by various commands in sessions and job streams. If so, you must capture file modification commands for the appropriate users. The safest way to do this is the command:

```
SETCATALOG SHADUDC.SHADOW.LPS;SYSTEM;APPEND
```

This makes the commands in SHADUDC the last to be searched, assuring that file modification commands issued by any user and from any other command file or UDC will be detected and honored. It may be possible for you to succeed with a SETCATALOG command that is only seen by particular accounts or users, but you must be very careful about this.

You may issue the SETCATALOG command before you are ready for replication to begin. The UDCs *will* intercept the commands, but those commands will be executed in the usual way, and no commands will be logged until capture activity is started with the CAPTON.SHADOW.LPS command.

Monitoring File Intrinsic Activity

Of course the files you want to replicate are almost certainly operated on by various programs. To capture this activity, you must activate the procedure exits handlers. We recommend that you activate them when then system is brought up, and leave them activated. Even though they are activated, they will not actually capture and log file activity until capture activity is started by the CAPTON.SHADOW.LPS command.

The safest way to activate the procedure exits is for all processes on the system. This assures that important activity will be captured. This is done with the ARMALL.SHADOW.LPS command.

If you can definitely determine which users and jobs will be modifying files, there is an alternative that monitors activity for only those users and jobs. Use a logon UDC to execute the ARM.SHADOW.LPS command for the appropriate users and jobs (or insert the ARM.SHADOW.LPS command directly into the jobs). This commands activates the handlers against the command interpreter process, and its children, for the user or job only. The advantage to doing this is that you are not adding any overhead to processes that do not modify the files you are replicating. This disadvantage is the danger that relevant activity will be missed.

The Shadow D/R XL file has a name of the form SXLvvvvv.HANDLERS.LPS, where vvvvv stands for the Shadow D/R version id. For this first release of file replication the versions is F.01.02, so the XL file would be named SXLF0102.HANDLERS.LPS.

Quick Start

Included with your properly installed version of Shadow D/R is a set of command files that may be used to provide a quick demonstration of file replication capabilities. It will replicate files from the group SHADPRI.LPS to the group SHADSEC.LPS. Please follow these steps. All of the files mentioned are in the group SHADDEMO.LPS.

1 Give LG capability to Mgr.LPS (unless that user already has OP or LG capability). This command issued by a user with SM capability will do the job:

```
ALTUSER MGR.LPS; CAP = +LG
```

2 Log on as Mgr.LPS, Shadow.

There should be a password. If not, maybe you should add one!

3 Allow Mgr.LPS the LOG command.

The START.SHADDEMO and STOP.SHADDEMO command files discussed below issue the LOG command. Normally this command is restricted to the console. However it can be temporarily given to Mgr.LPS (for the life of the session) by issuing this command:

```
ALLOW MGR.LPS; COMMANDS = LOG
```

4 Execute the Setup.ShadDemo command file.

The SETUP.SHADDEMO command file contains commands to do the following:

a Create a logid named SHADLOG.

b Create these groups:

- SHADPRI.LPS - to hold the primary files (the files to be replicated)
- SHADSEC.LPS- to hold the secondary files (the replicas of the files in ShadPri)
- SHADLOGP.LPS - to hold the primary log files
- SHADLOGS.LPS - to hold the secondary log files

c Copy the file replication configuration CAPTCONF.SHADDEMO to CAPTCONF.SHADOW. The Shadow D/R capture code looks for the capture configuration to be named CAPTCONF.SHADOW. This particular configuration just specifies that files in the group SHADPRI.LPS should be replicated to SHADSEC.LPS.

5 Modify the DemoConf.ShadDemo file.

The DEMOCONF.SHADDEMO file contains the Shadow D/R configuration to be used. However, the host name must be modified to be the name of the host on which Shadow D/R resides (which is serving both as the primary and the secondary host in this demonstration). Look for the dummy host name "<host>" in the configuration and change it to the correct name for your host (without the angular braces).

There are three logons in this configuration file (two :JOB commands and one :HELLO command). These must be modified to contain appropriate logon passwords.

6 Execute the Start.ShadDemo script.

The START.SHADDEMO command file contains a script that does the steps needed to initiate replication. These are:

a Stop replication in case it is already started.

b Start logging for the logid SHADLOG (includes purging old log files, building new ones, and issuing the Log command).

c Start up the Shadow D/R processes. It does this by running the program SHADOWMP.SHADOW, which is the control program for Shadow D/R. Normally a user runs this program and enters commands, but in this case input is redirected to the file START0.SHADDEMO.

You need to allow jobs to log on. Two jobs will be created, one on the primary, and one on the secondary (the same machine in this case). The first has the job name SHADPRI and runs on the primary. It runs a program named SHADOWCP.SHADOW which acts as a master process on the primary and creates a slave process on the secondary. Together these two processes will transmit the data from primary to

secondary (the same machine in this case). The second job is the posting job. The slave process hands data to the poster process in this job, which posts the changes.

This command file also does three things to enable file change capture:

- It issues the command:

  SETCATALOG SHADUDC.SHADOW; APPEND

  This will add the Shadow UDCs to whatever UDCs already exist for the MGR.LPS user. This enables Shadow D/R to see command activity done by the MGR.LPS user.
- It arms the Shadow D/R handlers for the current session.
- It runs the program CAPTON.SHADOW to tell the Shadow D/R handlers and UDCs to start capture activities.

7 Create file activity in group ShadPri.

At this point you can create, modify, and purge files in group SHADPRI.SHADOW, and this activity will be replicated to group SHADSEC.SHADOW.

The file change activity is logged to the user log files for logid SHADLOG in group SHADLOGP.LPS. You can view the status of logid SHADLOG with the command "SHOWLOGSTATUS SHADLOG".

The command file BULDKSAM.SHADDEMO will create a KSAM file that has three keys. The command file LOADKSAM.SHADDEMO will load it with data from CATALOG.PUB.SYS.

You can view the contents of the user log files by running the command file VIEWLOG.SHADDEMO. This formats the contents of the log files using the program DUMPLOG.SHADOW, and directs the formatted output to a file named LOGDUMP.SHADDEMO. It then runs Quad to view the file.

8 Stop replication.

The command STOP.SHADDEMO will stop all replication and capture activity, disarm the handlers, and unset the UDCs from user MGR.LPS.

Appendix A: The File Capture Configuration File

Configuration Overview

The file capture configuration is read from the file CAPTCONF.SHADOW.LPS. The following paragraphs describe the format and meaning of this file. Text in bold represents keywords. These should appear in the configuration as written, though case (upper case or lower) does not matter. Text in italics represents a generic element that will be further defined. Whatever is enclosed in braces is optional; e.g., {this is optional}.

It might be easiest to start out showing a sample configuration file.

<< Begin Sample Configuration File (this message not part of file) >>

```
!*************************************************************************!
!                                                                         !
!               Shadow D/R File Replication Configuration File            !
!                                                                         !
!*************************************************************************!


Begin
Fileset fileset1;
Logid dblog;
Files
testfil1.shadow
[shadow -> shadow2],
testfil1.shadow2
[shadow2 -> shadow3];
End
```

<< End Sample Configuration File (this message not part of file) >>

This configuration specifies that files from the groups testfil1.shadow and testfil1.shadow2 are to be monitored, and changes are to be logged to logid dblog. However at the destination, files found in testfil1.shadow are to be placed in testfil1.shadow2. Similarly, files found in testfil1.shadow2 are to be placed in testfil1.shadow3.

A file replication configuration takes the following form:

Begin
fileset;
...
fileset;
End

Thus there is a **Begin** keyword, an **End** keyword, and multiple fileset specifications, each one followed by a semicolon.

Filesets

Each fileset specification has this form:

Fileset *fileset_name*

Logid *logid_name*

Files

filespec,

filespec,

filespec;

The fileset_name is any name beginning with a letter and containing up to 16 letters or digits. This name is currently not used, but may be used in the future. The logid_name is a logid. It specifies the logid to which this fileset will be logged. Shadow D/R will currently accommodate as many as 10 different logid's. Different filesets may use the same logid.

The idea of a fileset is to provide a grouping mechanism for files that is similar to what a database provides for datasets (database files). Shadow D/R allows one to select particular databases to be replicated from a logid. In the future, Shadow D/R may allow one to specify particular filesets from the logid.

Filespecs and File Patterns

The filespec has the following form:

*include_file_pattern {(***MPEONLY***)}{[renaming_spec]}*

*exclude_file_pattern {(***MPEONLY***)}*

.

*exclude_file_pattern {(***MPEONLY***)}*

The include_file_pattern and exclude_file_pattern, as their name suggests, are patterns which are compared to file names. If a file name matches the include_file_pattern but not any of the exclude_file_pattern's, it is included in the filespec and in the fileset which contains the filespec. Therefore it is a file whose activity will be captured, and changes to the file will be recorded in the user log whose logid is specified for the fileset.

The **MPEONLY** keyword with its parentheses is optional. Its meaning is presented below.

The *renaming_spec* with its brackets is optional. If it is included, the file will be renamed during replication; i.e., it will have a different name on the secondary. This is covered below.

File Patterns

A file pattern (include_file_pattern or exclude_file_pattern) may be any of the following:

- An MPE account pattern
- An MPE group pattern
- An MPE file pattern
- A POSIX directory pattern
- A POSIX file pattern

The first three of these, the MPE patterns, translate directly to POSIX patterns, so the POSIX patterns will be discussed first.

POSIX File Patterns

A POSIX file pattern has the form:

/posix_pattern/.../posix_pattern

and a POSIX directory pattern has the form:

/posix_pattern/.../posix_pattern/

The only difference is the closing slash '/' on the directory pattern. This will be explained below.

These patterns are similar in form to a POSIX absolute pathname, which is a series of directory names and a final file name, all separated by slashes. Here is a sample POSIX absolute pathname:

/user/bin/shadowdr/control

A POSIX file pattern or directory pattern is compared element by element to a POSIX pathname. The pattern matches the pathname if each *posix_pattern* in the pattern matches the corresponding directory or file name in the POSIX pathname. The pathname may have extra unmatched directory or file names that are unmatched because the pathname is longer than the pattern. This doesn't matter.

Each *posix_pattern* is a string of characters that are allowed in POSIX names, and of the wild-card characters '?', '#', and '@'. These have the usual MPE meanings. Following are some examples.

The POSIX pattern:

/a/bcd

matches these pathnames:

/a/bcd

/a/bcd/efgh

/a/bcd/efgh/ij

It does *not* match:

/a

/a/bcde

Here is an example with wild cards.

The POSIX pattern:

/a/b#c@/d

matches:

/a/b2c/d

/a/b3c4a/d/efg

If the pattern is a directory pattern with a slash on the end, it is the equivalent of the file pattern obtained by adding an '@'. Thus the directory pattern:

/a/b/

is the same as the file pattern:

/a/b/@

The main point is that neither of these match a file named /a/b. Thus, the directory pattern matches files descending from the directory /a/b, but does not match a file of that name.

MPE File Patterns

Now that we have defined the POSIX directory and file patterns, it is easy to define the MPE patterns as a subset. However one should first be aware of how MPE-syntax filenames translate into POSIX filenames.

An MPE-syntax filename has the familiar form:

mpe_name.mpe_name.mpe_name

where each of the *mpe_name* elements has one to eight letters or digits and begins with a letter. This name translates to the POSIX name:

/MPE_NAME/ MPE_NAME / MPE_NAME

where we reverse the name order and upshift the characters. Thus the MPE-syntax name:

a.b.c

has the POSIX name:

/C/B/A

Something worth noting is that a file with an MPE-syntax name *may or may not be an MPE file*. For instance, in the example above, /C may be a POSIX directory, not an account; or even if /C is an account, /C/B may be a POSIX directory, not a group. In these cases the file may still be named either a.b.c or /C/B/A, but it is *not* an MPE file.

A file is an "MPE file" if and only if it resides in a group, and its name follows the MPE naming conventions of 1-8 (upper-case) letters or digits beginning with a letter.

Now we can talk about the MPE account, group and file patterns.

An MPE account pattern takes the form:

mpe_pattern

and is the equivalent of the POSIX file pattern:

/MPE_PATTERN/@

The *mpe_pattern* contains only letters, digits and the wild card characters '?', '#', and '@'. *MPE_PATTERN* is the upshifted version of it. Note that this pattern matches anything contained within the account, including both MPE and non-MPE files. If one wants only MPE files, one should use the **MPEONLY** keyword. More on this below.

An MPE group pattern takes the form:

mpe_pattern.mpe_pattern

and is the equivalent of the POSIX file pattern:

/MPE_PATTERN/MPE_PATTERN/@

The *MPE_PATTERN*'s are the upshifted *mpe_pattern*'s in reverse order. Thus the pattern:

s@.develop

is the equivalent of:

/DEVELOP/S@/@

An MPE file pattern takes the form:

mpe_pattern.mpe_pattern.mpe_pattern

and is the equivalent of the POSIX file pattern:

/MPE_PATTERN/MPE_PATTERN/MPE_PATTERN

where the *MPE_PATTERN*'s are the upshifted *mpe_pattern*'s in reverse order.

The MPEONLY Keyword

Every i*nclude_file_pattern* and every *exclude_file_pattern* may be followed in a *filespec* by the optional keyword **MPEONLY** enclosed in parentheses. This restricts the pattern to matching *only* MPE files. As mentioned above, MPE files are files that reside in MPE groups, and whose names are MPE names; i.e., they contain one to eight upper-case letters or digits and begin with a digit. The letters *must* be upper-case when considered from the POSIX name point of view. However when specifying them in MPE-syntax form they need not be written as upper-case. The meaning of the MPE syntax is that the name is implicitly upshifted.

The renaming_spec

Every *include_file_pattern* may optionally be followed by a *renaming_spec* enclosed in square brackets (the *renaming_spec* should be placed after the **MPEONLY** keyword and parentheses if they are there). The renaming spec specifies that the file is to be known under a new name on the secondary.

The renaming_spec has the form:

primary_name -> secondary_name

The *primary_name* and *secondary_name* are examples of file patterns as discussed above, except that wild cards are not allowed.

It might be best to give an example first before proceeding. Suppose the *filespec* is:

sys [sys -> sys2]

This means that the SYS account on the primary is to be replicated to the SYS2 account on the secondary. Here we have used the MPE account name form for the file patterns. The POSIX file name equivalent is:

/SYS/@ [/SYS/ -> /SYS2/]

In the process of renaming, the *primary_name* is matched against the pathname of the file being replicated. If there is a match, the matching part of the pathname is replaced by the *secondary_name*.

The *primary_name* and *secondary_name* need not be the same length. They can even be empty.

For example, consider this *filespec*:

sys [-> /LPS/SHADOW]

The empty *primary_name* is considered to match an empty prefix of the replicated file's pathname. Therefore the *secondary_name* is inserted in front of the file's pathname. For example, a file named:

command.pub.sys

would become:

/LPS/SHADOW/SYS/PUB/COMMAND

on the secondary.

Appendix B: What Intrinsics are Intercepted

The procedure exits handlers currently intercept these file system intrinsics:

HPFOPEN
FOPEN
FCLOSE
FWRITE
FWRITEDIR
FWRITELABEL
FRENAME
FUPDATE
COMMAND
FREMOVE

Since the HPCICOMMAND intrinsic will execute UDCs and command files, its activity is replicated through the UDCs, and need not be intercepted by the procedure exits handlers.

Appendix C: What Commands are Intercepted

The Shadow D/R UDCs intercept these MPE commands:

- Build
- Copy
- Newlink
- Purge
- Purgelink
- Rename
- Save

Appendix D: What file Activity is Replicated

Here are the rules about what file activity is replicated:

- The file must be in the permanent domain. New or temporary files are not replicated. If a new or temporary file is saved at close time into the permanent domain, it will be replicated in its entirety at that time. One should note that this could cause a large delay in the closing program at that time.
- Files that are permanent but whose pending disposition domain is PURGE are not replicated. As soon as a process closes a permanent file with this disposition domain, and it becomes the pending disposition domain, a delete message is sent to the secondary. At a later time another process might override this domain with a close that causes the file to be saved. If so, the file will be retransmitted to the secondary (we don't expect many events such as these).
- The file's pathname must be included in one of the filesets in the file capture configuration. Please refer back to "Appendix A: The File Capture Configuration File," for details about this configuration.
- The file must be an "ordinary" (flat) MPE file (type 0), KSAM XL file (type 3) or a KSAM64 file (type 7). Flat files include the various record types of fixed, variable, undefined and byte stream. Other types are *not* replicated. These include KSAM/3000 (compatibility mode KSAM, type 1), RIO (type 2), message (type 6), directory (type 9), pipe (type 12), FIFO (type 13).
- Files with negative filecodes are not replicated. Of course, TurboImage activity *is* replicated by Shadow D/R, but not as part of the new file replication capability. It may be that there are some other significant files with negative file codes. If so, future versions of Shadow D/R may replicate them.
- "System" files such as $NEWPASS and $OLDPASS are not replicated.

Appendix E: Limitations of File Replication

Save Disposition Specified at Open Time

If a close disposition of SAVE PERMANENT is specified at open time, Shadow D/R will transfer the file at close time. If the process dies without closing, this does not get done.

Non-standard Fill Characters

MPE uses fill characters when some part is caused to exist by a file intrinsic, but the contents of that part is not specified by the intrinsic. This can happen when records are written that are shorter than the record size for the file, or when records are written by FWRITEDIR beyond the EOF leaving unspecified records.

The standard fill characters used by MPE are *blank* for ASCII files and *null* for binary files. MPE allows, in the HPFOPEN intrinsic, the specification of non-standard fill characters. Any characters are allowed. Unfortunately MPE does not provide a supported way for Shadow D/R to determine what non-standard fill characters have been specified for a file. Shadow D/R will note the fill characters at HPFOPEN time if possible, but this is not always possible. Fortunately, the use of non-standard fill characters is probably very rare.

The PUTACD Intrinsic is not Being Replicated

When a file is created on the primary, whatever ACDs it was created with are replicated. However if a later operation does a PUTACD or other operation that modifies ACDs, the change is not replicated. An enhancement to do this is high-priority, and will appear on a later Shadow D/R release.

Mapped Files Transferred at Close Time

Mapped file activity cannot be replicated as it occurs. The interception techniques used by Shadow D/R cannot detect mapped file access. Therefore files that are opened mapped are transmitted in their entirety at close time.

Appendix F: New Shadow D/R Files for File Replication

These files are new in version F.01.01 of Shadow D/R:

File Name	Description
DISARMAL.SHADOW.LPS	Command file to disarm all processes in the system
SHADUDC.SHADOW.LPS	UDC file to capture Command Interpreter file change activity
ARMPE.SHADOW.LPS	Program that arms processes with procedure exits handlers, used by ARMALL
CAPTCMD.SHADOW.LPS	Program invoked within SHADUDC to log commands to the user log
CAPTOFF.SHADOW.LPS	Program to turn off all file capture activity
CAPTON.SHADOW.LPS	Program to turn on file capture activity
CAPTSTAT.SHADOW.LPS	Program that displays the status (on or off) of file capture activity
CHKCONF.SHADOW.LPS	Program that checks the syntactical validity of a configuration file.
DISARMPE.SHADOW.LPS	Program that disarms process from procedure exits handlers, used by DSIARMALL
DUMPLOG.SHADOW.LPS	Program to dump the file change contents of a user log
FILEGEN.SHADOW.LPS	Program that generates file access activity, used for testing replication
SXL?####.HANDLERS.LPS	XL files that contain the procedures exits handlers that capture user process file change activity

The following files are all part of the Shadow D/R demonstration, and are not needed by Shadow D/R:

ARM.SHADDEMO.LPS
BUILDDB.SHADDEMO.LPS
BUILDDB0.SHADDEMO.LPS
BUILDDB1.SHADDEMO.LPS
BULDKSAM.SHADDEMO.LPS
CAPTCONF.SHADDEMO.LPS
DISARM.SHADDEMO.LPS
GETCIPIN.SHADDEMO.LPS
INVENSCH.SHADDEMO.LPS
LISTSHAD.SHADDEMO.LPS
LOADKSAM.SHADDEMO.LPS
LOGDUMP.SHADDEMO.LPS
PRGEDEMO.SHADDEMO.LPS
READKSAM.SHADDEMO.LPS
SAMPCONF.SHADDEMO.LPS
SETUP.SHADDEMO.LPS
START.SHADDEMO.LPS
START0.SHADDEMO.LPS
STATUS.SHADDEMO.LPS
STATUS0.SHADDEMO.LPS
STOP.SHADDEMO.LPS
STOP0.SHADDEMO.LPS
VIEWLOG.SHADDEMO.LPS