

IBM DS8000 and DSClIbroker

Peter Klee



Storage



Introduction

The DSCLlbroker is a scripting framework that automates Copy Services functions. The DSCLlbroker provides the following features:

- ▶ Grouping volumes according to applications or other context
- ▶ Simplifies the execution of Copy Services commands
- ▶ Provides a scripting framework for implementing automation functions

More information: DSCLlbroker is available as an STG-Lab Service. Contact your IBM® representative to order this service. If you are outside of IBM, talk to your IBM Representative about ordering the service.

This IBM Redpaper™ publication describes IBM DS8000® and the DSCLlbroker scripting framework and its features.

IBM DS8000 and the DSCLlbroker scripting framework

In addition to the storage capacity for application data, the storage infrastructure of a modern data center must also replicate the data to other storage devices. With the Copy Services and interfaces, such as DSCLI, applications can use this replication by controlling Copy Services for their own purposes. These functions are commonly used for disaster prevention or data backup purposes.

Copy Services are frequently used for data migrations, but migrations are not a daily operation. Depending on your data center environment, applications, and type of migration, the migration tasks can get complicated. As a result, you might want to automate certain tasks, especially when the required actions cannot be accomplished using the standard storage management software.

The *DSCLIBroker* is a scripting framework that allows you to create user customized automation scripts. For example, consider a multitiered layered stack consisting of the DS8000 hardware as the lowest level and DSCLI above. You might want to write automation scripts where DSCLI commands are run against the storage. The DSCLIBroker can be positioned as an extra layer between the DSCLI and the applications (Figure 1).

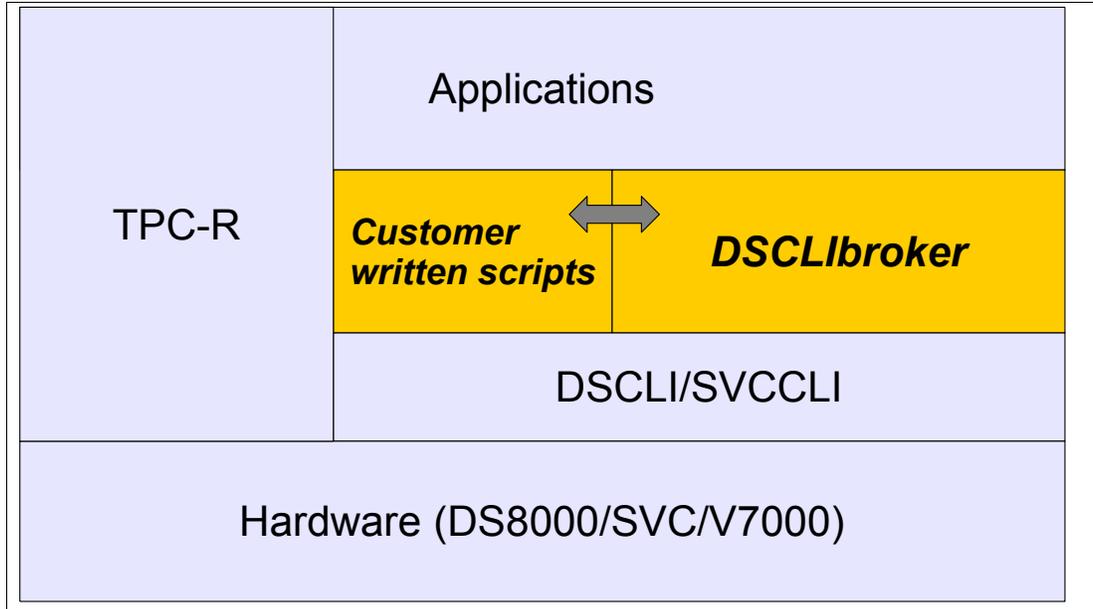


Figure 1 Positioning of DSCLIBroker

The framework is written in Perl scripting language and consists of a series of perl library modules that support all DS8000 Copy Services functions. You can write your own scripts using the library. The framework also provides a perl script for each Copy Services DSCLI command that can be used without additional programming.

In the subject field, enter Migration using DSCLIBroker and continue to process the form.

DSCLIBroker concepts

The complexity of Copy Service configuration increases as the number of applications involved and the size of those applications increases. The more applications must be migrated at the same time, the more care must be taken to map the correct volumes to the applications. In addition, larger application can have large lists of volumes that have copy relations to other volumes. Migrating data to the correct volumes is vital to avoid data inconsistency.

Using DSCLI Copy Services commands, with every command a list of copy relations must be specified. In a migration scenario with the steps: **mkpprc**, **1spprc**, **pausepprc**, **1spprc**, **failoverpprc** and again **1spprc**, the list of copy relations must be specified six times. It requires much effort to maintain these commands either in self written scripts or on the DSCLI command line.

With DSCLIBroker, the configuration data of the Copy Services relations is separated from the scripting code in a *repository*. In this repository, multiple Copy Services relations belonging to a single application can be grouped together and tagged with a name. When a DSCLIBroker script is run, it refers to the tagged name, and the DSCLIBroker then fetches all copy relations from the repository. Maintenance of these relationships is done in the repository, and so you do not need to change the scripting code.

In addition, the DSCLlbroker provides a scripting framework that offers an easy way to write user customized scripts. This framework is implemented in modular libraries written in perl scripting language with an object-oriented approach. There is one library for each DS8000 Copy Services function available. The libraries themselves are organized so they can be extended to support other storage platforms too. The current storage platforms supported include DS8000, IBM DS6000™ and ESS Model 800. Plans are in place to support SAN Volume Controller and V7000 storage platforms and the TotalStorage Productivity Center for Replication command-line interface.

DSCLlbroker libraries

The libraries are the core of the scripting framework. If you write perl scripts using the DSCLlbroker, you must include the libraries.

DSCLlbroker provides the following libraries:

DSPPRC.pm	This library is an object class where all remote copy functions are implemented. The functions include managing the paths and the pair relations.
DSFlashCopy.pm	This library is an object class that holds all functions that maintain IBM FlashCopy® relations.
DSGlobalMirror.pm	This library is an object class as well. It holds all functions related to DS8000 Global Mirror.
DSCLInator.pm	This library is the meta class for the preceding classes. Commonly used functions for Copy Services are located here.
DSlib.pm	This library contains global functions with no relations to Copy Services functions such as maintaining the DSCLlbroker environment and querying and retrieving data from the repository.
DBbox.pm	This library contains all necessary functions to communicate with the storage subsystems.

Figure 2 shows an overview of the libraries of the DSCLlbroker.

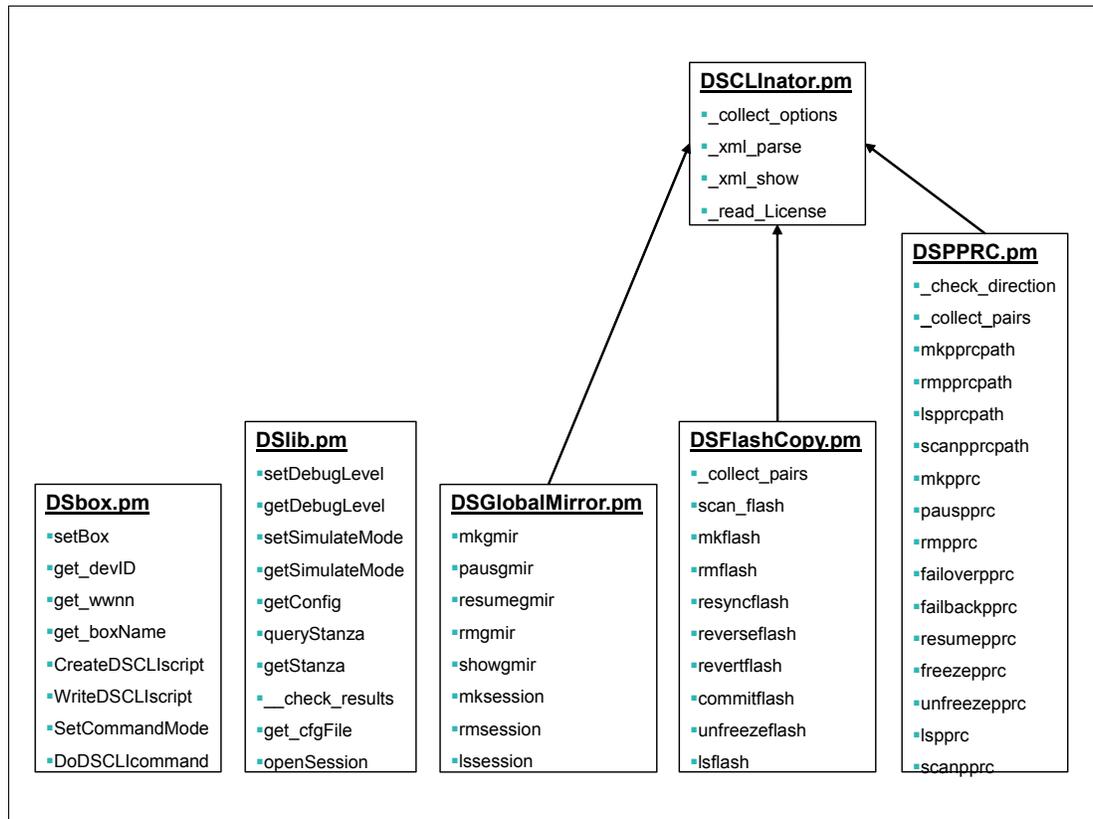


Figure 2 DSCLlbroker libraries

The data repository

All configuration data is stored in the data repository and is used by the DSCLlbroker libraries when composing commands for the DSCLI. In this sense, *configuration data* includes the following information:

- ▶ Copy relations defined by source volume to target volume.
- ▶ Copy paths information for remote copy, which includes the relation source LSS to target LSS used by source port to target port.
- ▶ Definitions for DS8000 Global Mirror.
- ▶ Information about the storage systems.

Because some DSCLI commands use the same information, this information can be collected in the same repository entity. For example, when creating a Metro Mirror or a Global Copy, the same DSCLI command is used. Metro Mirror is denoted by the option **-type mmir** and Global Copy uses the option **-type gcp**. The remaining parameters are the source and target device, and the source and target volumes. A FlashCopy needs the same set of information except that the target device is not required. All these copy pair relations can be described with the same set of data, which is in the form of a database table or stanza file.

The complete set of required information is shown in Figure 3.

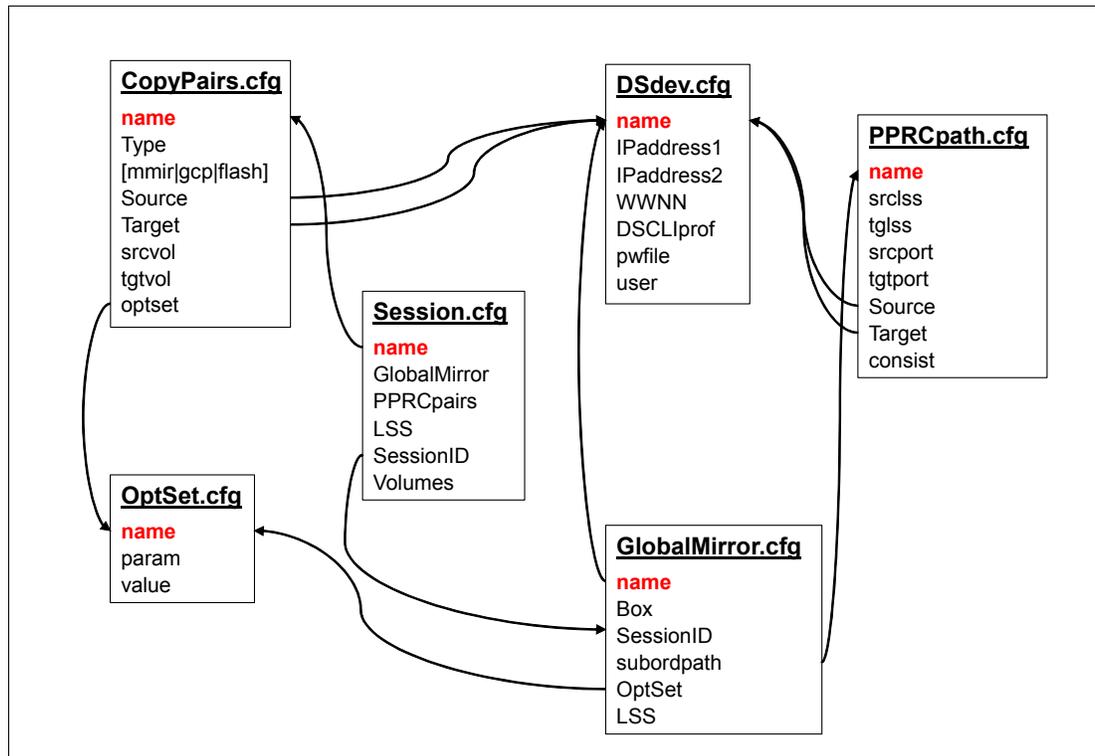


Figure 3 DSCLlbroker data model

This set of tables is a normalized data model, which in theory can be used by a relational database system. For DSCLlbroker however, the tables are implemented as flat stanza files because they can handle hundreds of thousands of entities without any problems. When the data migrations are done, the data in the repository is obsolete and can be discarded. In other engagements, the data in the repository must be maintained for a longer time. In these engagements, the data management capabilities of a database system might be more useful or even required.

As shown in Figure 3, each table has a name that is the key to a set of configuration data. A table might have an entry that refers to a key name of another table. For example, in the table CopyPairs.cfg, the entries for **Source** and **Target** reference a storage device defined in the table DSdev.cfg.

The following section includes an example for a complete repository definition where an application named *SAPHRI* manages a copy relation using the DSCLlbroker. The whole setup includes the following steps:

1. Define the copy relations.
2. Define the storage devices.
3. Define the paths.

In the example, the application SAPHR1 is using the volumes 820A-820C and 8410-841F, which are two different contiguous volume ranges. The complete copy relation for this application is denoted in the stanza file `CopyPairs.cfg`. One stanza is created for the first volume range, and another for the second volume range. Both stanzas have the same name, `SAPHR1_ab` (Example 1).

Example 1 Defining a complex copy relation

```
CopyPair {
    name      = SAPHR1_ab
    type      = gcp
    srcss     = 82
    tgtss     = 8E
    srcvol    = 820A-820C
    tgtvol    = 8E0A-8E0C
    source    = ATS_3
    target    = ATS_1
    seqnun    =
    optset    = gcp_cascade
}

CopyPair {
    name      = SAPHR1_ab
    type      = gcp
    srcss     = 84
    tgtss     = 8E
    srcvol    = 8410-841F
    tgtvol    = 8E10-8E1F
    source    = ATS_3
    target    = ATS_1
    seqnun    =
    optset    = gcp_cascade
}
```

In some cases, one or more command-line options are required multiple times. For example, a Global Copy relation that is cascaded to an existing remote copy relation requires the option `-cascade`. This option can be placed in the repository using the stanza file `OptSet.cfg` as shown in Example 2. The defined option set `gcp_cascade` is referenced in the `CopyPair` stanza with the `optset` tag, as shown in Example 1.

Example 2 Define parameter in OptSet.cfg

```
OptSet {
    name      = gcp_cascade
    param     = cascade
    value     = yes
}
```

When this entry is referenced in a `CopyPairs.cfg` stanza, the option `-cascade` is the default every time DSCLbroker generates a DSCLI command for this relation.

In the CopyPair stanzas, the entries for **source** and **target** refer to the source and target storage devices. Both entries need to be defined in the DSdev.cfg file. In Example 3, the source device is named ATS_3 and the target device is named ATS_1.

Example 3 Definition of the storage devices

```
DSdev {
  name      = ATS_1
  IPaddress1 = 9.155.70.26
  IPaddress2 = 9.155.70.55
  WWNN      = 5005076303FFC08F
  devID     = IBM.2107-7503461
  DSCLIProf = script_03461.profile
  PWfile    = script_03461.pwfile
  user      = script
}

DSdev {
  name      = ATS_3
  IPaddress1 = 9.155.62.97
  IPaddress2 = 9.155.62.97
  WWNN      = 5005076303FFC1A5
  devID     = IBM.2107-7520781
  DSCLIProf = script_20781.profile
  PWfile    = script_20781.pwfile
  user      = script
}
```

Remember: As shown in the stanza, password files are used for the authentication to DSCLI. When writing scripts that runs command against the DSCLI, an automated authentication to the DSCLI is useful. Otherwise you must type in the user name and password each time the script is run. DSCLI offers a secure method to log in to the DSCLI automatically.

After identifying the storage devices, the paths must be specified. To establish the paths for copy relations, a physical link must be established first. Verify that the links are available and the required information for the stanza entries can be obtained using the script **lsavailpprcport.pl**, as shown in Example 4.

Example 4 Obtaining the port information for the paths

```
$ lsavailpprcport.pl -source ATS_1 -target ATS_3 -src1ss 84 -tgt1ss 8e
Local Port          Attached Port      Type
=====
IBM.2107-7503461/I0141 IBM.2107-7520781/I0402 FCP
IBM.2107-7503461/I0142 IBM.2107-7520781/I0400 FCP
$_
```

In this example, two ports have physical connections to the target storage device. Use this information to create the stanza file for the paths. For each repository stanza file, a script is available that generates the stanza information automatically. Use these scripts for the path definitions. The path definitions are the most vital in the repository because the configuration of the path can get complex. Only when the path definitions are correct will the subsequent copy commands work properly.

Use the `gen_pprcpaths.pl` command to generating path stanzas as shown in Example 5. The option `-name SAPHR1` defines the base name of the stanza. The option `-d 'f:ab'` specifies that the paths data are created for the forward direction only. This option adds `_ab` to the stanza base name, which results in the real stanza name `SAPHR1_ab`. The option `-l` is used to specify the LSS relation as given. The option `-p` is used for the port pairs as shown by the `1savai1pprc.pl` command in Example 4 on page 7.

Example 5 Generating path stanzas

```
$ gen_pprcpaths.pl -name SAPHR1 -d 'f:ab' -l '82:8E 84:8E' -p 'I0141:I0402
I0142:I0400' -s ATS_1 -t ATS_3
#####
#
# PPRC paths for SAPHR1_ab
#
#####
#####
# LSS 82 -> 8E
# Box ATS_1 -> ATS_3
#####
PPRCpath {
    name          = SAPHR1_ab
    srcLSS        = 82
    tgtLSS        = 8E
    srcport       = I0141
    tgtport       = I0402
    Source        = ATS_1
    Target        = ATS_3
    consist       = no
}

PPRCpath {
    name          = SAPHR1_ab
    srcLSS        = 82
    tgtLSS        = 8E
    srcport       = I0142
    tgtport       = I0400
    Source        = ATS_1
    Target        = ATS_3
    consist       = no
}

#####
# LSS 84 -> 8E
# Box ATS_1 -> ATS_3
#####
PPRCpath {
    name          = SAPHR1_ab
    srcLSS        = 84
    tgtLSS        = 8E
    srcport       = I0141
    tgtport       = I0402
    Source        = ATS_1
    Target        = ATS_3
    consist       = no
}
```

```

PPRCpath {
  name      = SAPHR1_ab
  src1ss    = 84
  tgt1ss    = 8E
  srcport   = I0142
  tgtport   = I0400
  Source    = ATS_1
  Target    = ATS_3
  consist   = no
}

```

DSCLlbroker scripts

The DSCLlbroker comes with scripts that can be used as soon as the repository has the required configuration data. For FlashCopy, Global Copy, and Metro Mirror, the stanzas for the Storage device must contain the paths and the copy pairs. For more information, see “DSCLlbroker concepts” on page 2. For Global Mirror, the stanzas **GlobaMirror.cfg** and **Session.cfg** must also be filled with data.

For each Copy Services-related DSCLl command, a corresponding DSCLlbroker script is available. Each script provides all command options that you can use in DSCLl. Some scripts have additional options that provide additional functionality. For example, using the **-p** option with the command **failoverpprc.pl** pauses the relation at the primary site before the failover. All scripts require the option **-name**, where the corresponding tagged name of the stanzas is requested. If no options are provided, you see a help text where the complete syntax is shown. Example 6 shows the help text for the command **mkpprc.pl**.

Example 6 Help output example

```

$ mkpprc.pl
Usage:
./mkpprc.pl
  [-h|help]# This output
  [-d|debug 1-4]# Set the debug level. Recommended level: 2
  [-s|simulate]# Run script in simulate mode. Requires -debug 2
  [-b|banner]# Prints out a banner
  [-direction forward|reverse]
           # Specifies in which direction of the pairs
           # should operate. The default is -d forward.
  [-t|type mmir|gcp]
           # Overrides the type specified in the stanzas
  [-m|mode full|nocp]
           # Overrides the copy mode specified in the stanzas
  [-cascade]# Enable relation to be cascaded
  [-nocascade]# Overwrite the cascade option if set in OptSet
  [-incresync enable|enableoinit|disable|recover|override]
           # apply incremental resync. Overrides the stanzas
  [-to|tgtonline]# enables target to get online (z/OS only)
  [-tr|tgtread]# Allows read access from target volumes
  [-suspend]# suspend relations after task has finished
  [-crit|critmode]# enables critmode (z/OS only)
  [-disableautoresync]# disable the auto resync functionality of Global Copy
  [-rr|resetreserve]# reset the reservation on the target volumes
  [-tgtse]# specifies the target volumes as space efficient

```

```

[-w|wait]# wait until initial copy has completed
# WATCH OUT: May take a while. Make sure that IPC_Timeout and NET_Timeout in
# DSCLIBroker.cfg matches. Otherwise the session may break down!!!
[-nooppt]# Will not use a defined option set
-n|name <CopyPairName>
# Specifies the name of the copy pairs
$ _

```

The following examples are based on the configuration data that was created in the previous chapter. You can now establish the Global Copy relation using the script `mkpprc.pl` as shown in Example 7. The only required parameter you must specify is the name of the stanza where all copy relations are defined. `SAPHR1_ab` is the corresponding stanza as shown in Example 1 on page 6.

Example 7 Establish the Global Copy with mkpprc.pl

```

$ mkpprc.pl -n SAPHR1_ab
CMUC00153I mkpprc: Remote Mirror and Copy volume pair relationship 820A:8E0A successfully created.
CMUC00153I mkpprc: Remote Mirror and Copy volume pair relationship 820B:8E0B successfully created.
... snippet ...
CMUC00153I mkpprc: Remote Mirror and Copy volume pair relationship 841F:8E1F successfully created.
$ _

```

All other scripts work in the same way. The only required parameter is the name of the copy relation located in the stanzas. Additional parameters can be supplied, depending on what you want to do. For an overview of the available options, use the `-help` option or run the command without any parameters. Example 8 shows how to pause the Global Copy.

Example 8 Pause Global Copy with pausepprc.pl

```

$ pausepprc.pl -n teamblack_ab
CMUC00157I pausepprc: Remote Mirror and Copy volume pair relationship 820A:8E0A successfully paused.
CMUC00157I pausepprc: Remote Mirror and Copy volume pair relationship 820B:8E0B successfully paused.
... snippet ...
CMUC00157I pausepprc: Remote Mirror and Copy volume pair relationship 841F:8E1F successfully paused.
$ _

```

In the previous examples, no information is provided about how the script is generating the DSCLI commands. To make the scripts more verbose, the `-d` (debug) option can be used. There are four levels of debug information available. The first debug level shows the generated DSCLI command.

Another helpful option is the `-simulate` option, which displays the DSCLI command but does not run it. This option can be used to verify whether the generated DSCLI command is the one you are expecting before it takes effect. The `-d 1` option also shows the generated command, but the command runs.

Example 9 shows the output when using the debug and simulate options.

Example 9 Simulate mode and verbose command execution for mkpprc.pl

```

$ mkpprc.pl -n SAPHR1_ab -simulate
mkpprc -dev IBM.2107-75ABTV1 -remotedev IBM.2107-7503461 -mode full -type gcp 820A-820C:8E0A-8E0C
8410-841F:8E10-8E1F
$
$ mkpprc.pl -n SAPHR1_ab -d 1
Looking for name: SAPHR1_ab

```

```

mkpprc -dev IBM.2107-75ABTV1 -remotedev IBM.2107-7503461 -mode full -type gcp 820A-820C:8E0A-8E0C
8410-841F:8E10-8E1F
CMUC00153I mkpprc: Remote Mirror and Copy volume pair relationship 820A:8E0A successfully created.
CMUC00153I mkpprc: Remote Mirror and Copy volume pair relationship 820B:8E0B successfully created.
... snippet ...
CMUC00153I mkpprc: Remote Mirror and Copy volume pair relationship 841F:8E1F successfully created.
$ _

```

When a PPRC relation has to be failed over, the DSCLI command **failoverpprc** must be issued at the target storage device. Provide the pair relations in the reverse order. The **failoverpprc.pl** script does both automatically. In Example 10 the generated **failoverpprc** command is displayed. Comparing it to the Example 9 on page 10, the device IDs of the storage devices and the pair relations are reversed.

Example 10 Fail over Global Copy

```

failoverpprc.pl -n SAPHRI_ab -d 1
Looking for name: SAPHRI_ab
failoverpprc -dev IBM.2107-7503461 -remotedev IBM.2107-75ABTV1 -type mmir 8E0A-8E0C:820A-820C
8E10-8E1F:8410-841F
CMUC00196I failoverpprc: Remote Mirror and Copy pair 8E0A:820A successfully reversed.
CMUC00196I failoverpprc: Remote Mirror and Copy pair 8E0B:820B successfully reversed.
... snippet ...
CMUC00196I failoverpprc: Remote Mirror and Copy pair 8E1F:841F successfully reversed.
$ _

```

User customized scripts

You can write your own scripts if you have basic perl scripting language skills. The perl scripts must include the DSCLIBroker perl libraries as described in “DSCLIBroker libraries” on page 3. The DSCLIBroker libraries are designed as a modular set of building blocks to make creating scripts easier.

The following example demonstrates a simplified migration to target DS8000 storage devices using a Global Copy replication. The following steps are processed by the scripts:

- ▶ Create the paths to the target storage device.
- ▶ Establish a Global Copy and wait until all tracks have been copied.
- ▶ Pause the Global Copy.
- ▶ Fail over Global Copy to the target site.

Using this script, you can perform a series of migrations without any changes to the code. For each application that needs to be migrated, you must change the data in the corresponding stanza files.

Example 11 shows the complete script. This script uses the libraries `DSlib.pm`, where the debug levels and the simulation mode are included, and `DSPPRC.pm`, where all remote copy functions are defined. The libraries are referenced in lines 6 and 7 of the script.

Example 11 Example migration script using DSCLIBroker

```

1 #!/usr/bin/perl -w
2
3 #
4 # Include required library modules
5 #
6 use DSlib;

```

```

7 use DSPPRC;
8
9 #
10 # Declare command line options
11 #
12 use Getopt::Long;
13 my ($opt_debug,
14     $opt_simulate,
15     $opt_name,
16 );
17
18 GetOptions(
19     "d|debug:i"    => \$opt_debug,    # --debug
20     "s|simulate"  => \$opt_simulate,  # --simulate
21     "n|name=s"    => \$opt_name,      # --name <stanza_name>
22 );
23
24 if (!defined $opt_name) {
25     print "option -name is required\n";
26     exit(3);
27 }
28
29 #
30 # set debug level and simulator mode
31 #
32 my $dbg_level=setDebugLevel($opt_debug);
33 my $dbg_simulate=setSimulateMode($opt_simulate);
34
35 #
36 # Define source and target device objects
37 #
38 my $source=DSbox->new();
39 my $target=DSbox->new();
40
41 #
42 # Create PPRC thingy
43 #
44 my $PPRC=DSPPRC->new($source,$target);
45
46 #
47 # Apply command for the primary site
48 #
49 $PPRC->mkpprcpath($opt_name);
50 $PPRC->mkpprc($opt_name,'forward','-wait');
51 $PPRC->pausepprc($opt_name);
52
53 #
54 # Execute primary DSCLI commands
55 #
56 my $ret=$source->DoDSCLIcommand('server');
57
58 #
59 # Apply failoverpprc at secondary site
60 #
61 $PPRC->failoverpprc($opt_name);

```

```
62 $ret=$target->DoDSCLIcommand('server');
63
64
65 exit($ret);
```

In the lines 9 - 27 of the script, the required command options are declared. The **-name** option is required and therefore the lines 24 - 27 are checking whether this option is supplied. If the option is not given, an error message is reported and the script exits.

Verify the DSCLI commands that are generated by this script before they take effect. Therefore, the simulation mode is enabled when the command-line option **-simulate** is specified. The simulation mode is engaged in line 33.

In lines 35 - 44, the necessary objects are defined. There is one object for each storage device required, and an object that provides the remote copy function to the script.

In lines 49 - 51, the DSCLI commands for the paths and the copy pairs are generated. In this case, a Metro Mirror is established with the option **-wait**. The Metro Mirror allows all tracks to be copied to the secondary site before the operation continues. When the copying is complete, the copy relation is paused using the **pausepprc** command. This command sequence must be applied to the primary storage device.

In line 56, all DSCLI commands generated in the previous lines are sent to the primary storage device for execution.

In line 59, the failover to the secondary device is applied in the same manner. This command is sent to the auxiliary storage device. This script is now completed.

Example 12 shows the output of the simulation mode of the script, which is the generated sequence of DSCLI commands. The commands, up to the **pausepprc** command, are sent to the primary storage device. The **failoverpprc** command is sent to the target storage device.

Example 12 Simulation mode output

```
$ ITS0examp.pl -name app1_ab -s
mkpprcpath -dev IBM.2107-75ABTV1 -remotedev IBM.2107-7503461 -remotewwnn
5005076303FFC08F -src1ss 42 -tgt1ss 64 -consistgrp I0011:I0142 I0012:I0141

mkpprcpath -dev IBM.2107-75ABTV1 -remotedev IBM.2107-7503461 -remotewwnn
5005076303FFC08F -src1ss 53 -tgt1ss 57 -consistgrp I0011:I0142 I0012:I0141

mkpprc -dev IBM.2107-75ABTV1 -remotedev IBM.2107-7503461 -mode full -type -wait
4200-4203:6400-6403 5300-5303:5700-5703

pausepprc -dev IBM.2107-75ABTV1 -remotedev IBM.2107-7503461 4200-4203:6400-6403
5300-5303:5700-5703

failoverpprc -dev IBM.2107-7503461 -remotedev IBM.2107-75ABTV1 -type mmir
6400-6403:4200-4203 5700-5703:5300-5303

$_
```

This script can be used for a series of migrations. In this example, the migrations are done application by application. In other words, this script will migrate one application after the other. For each migration, the same script is run, with the name of the application is supplied

with the `-name` option. The only thing that must be changed are the copy relations for each application, which must be defined correctly in the `CopyPairs.cfg` stanza file.

Additional useful scripts

The DSCLIBroker framework includes additional scripts that help maintain the data repository. There are also other useful scripts that enable enhanced functions for the DSCLI commands. The following are the additional scripts and their functions:

- ▶ `gen_dsdev.pl`, `gen_gmir.pl`, `gen_pprcpairs.pl`, `gen_pprcpaths.pl`, `gen_session.pl`

These scripts generate the stanza files that correspond to their name. They format the stanza files and place the values you supply with the command file options in the correct places. The scripts `gen_pprcpaths.pl` and `gen_pprcpairs.pl` are especially useful because they allow you to list multiple relations in the command-line options. For more information about creating stanzas dynamically, see , “Automation techniques” on page 21.

- ▶ `ValidatePairs.pl`, `ValidatePaths.pl`

These scripts validate the pair and path configuration. This means that the content of the data in the stanza is compared against the real status on the storage devices.

- ▶ `scanpprc.pl`, `scanflash.pl`, `scanpprcpath.pl`, `scansession.pl`

The scan scripts obtain information from the storage devices and the output from the DSCLI is sent back to the DSCLIBroker as an XML data format. This information allows the scan commands to select specific parameters and display their values in the window.

- ▶ `QUERY.pl`

This script allows you to retrieve selected data from the repository. `QUERY.pl` searches after a pattern in a stanza file and prints out the results as a stanza or separated by commas.

- ▶ `lsprrc.pl -sum`, `lsflash.pl -sum`

The `-sum` option with the `lsprrc.pl` and `lsflash.pl` scripts prints the total number of Out-of-Sync tracks. This list is useful when many copy pairs are involved. Otherwise, to obtain the total amount of Out-of-Sync tracks for each single output line, the Out-of-Sync tracks must be summarized manually.

- ▶ `lsprrc.pl -waitnull`, `lsflash.pl -waitnull`

The `-waitnull` option is similar to the `-sum` option, but the command will not return until all tracks have been copied to the remote target volumes.

System environment

The system environment of the DSCLIBroker is divided in a server part and a client part. In this section, the architecture of the DSCLIBroker, and how the communication and the maintenance are organized are described.

Architecture

The DSCLIBroker itself is organized as a client server application. The server part includes the broker itself. The broker is a daemon that waits for a request from a client to open a connection to a storage device. This connection is called a session. The session is opened by forking a child process where the DS8000 command-line interface (DSCLI) is started. The session waits at the input prompt. The broker manages the communication to the client.

The server where the DSCLI is running must have the role of a trusted storage management server. The server hardware must have access to the DS8000 storage systems, but they must also be accessible by storage administrators. This server can additionally be a gateway to the storage environment. Therefore it must be equipped with two different network interfaces. One interface is used for the administrator access and the second one is used to communicate with the storage environment. Figure 4 shows the system architecture.

The server also hosts the repository with all its stanza files. In this way, the configuration data is in one centralized spot and does not need to be distributed to the administrators.

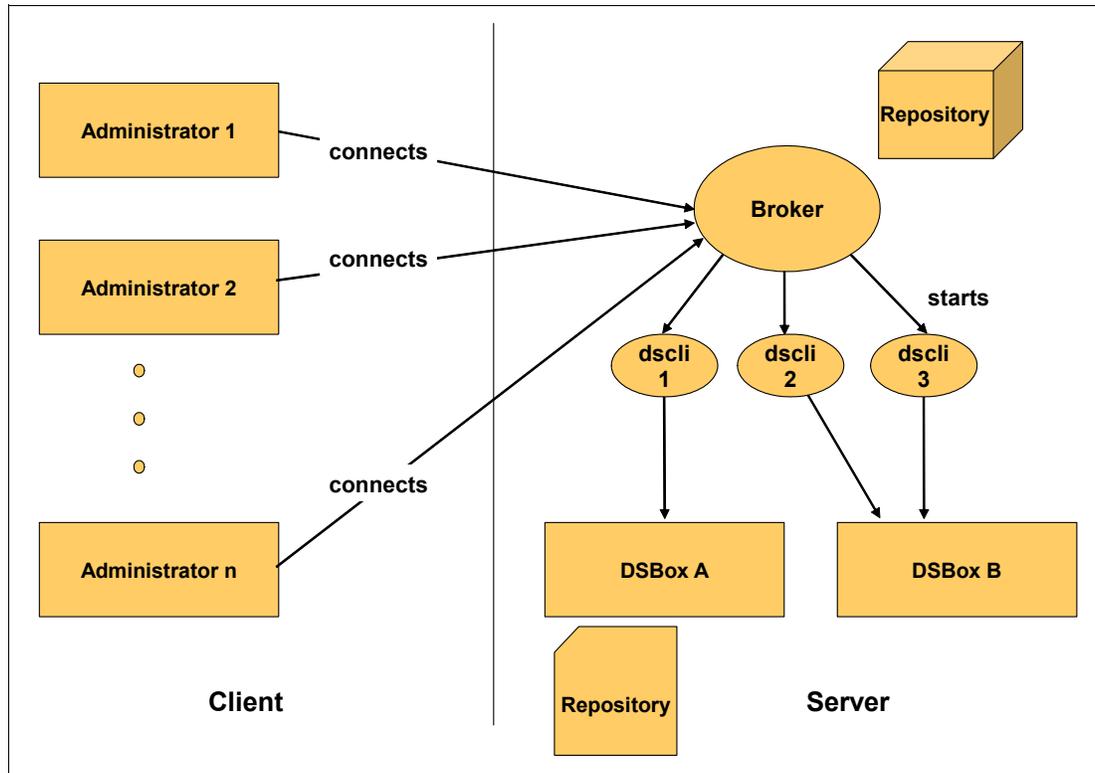


Figure 4 DSCLIBroker architecture

The client part of the DSCLIBroker is dedicated to the storage administrators, who work with one or more storage devices. Every DSCLIBroker script that runs at the client site communicates with the server to perform the following tasks:

- ▶ Retrieve data from the repository
- ▶ Compose the DSCLI commands
- ▶ Send the commands back to the server

The server forwards the commands to the corresponding DSCLI daemon waiting in the background. When the command is run by the DSCLI, the results are sent back to the client.

Communication

The communication between the DSCLIBroker client and the server is organized in sessions. The storage administrator, working as a client, must establish a *session* to a storage device. Use the script `startSession.pl` to send a request with a name to the corresponding entry in the `DSdev.cfg` stanza file to the broker. When the broker finds the stanza entry, it uses this information to fork a process call the *worker* and starts it in the DSCLI.

The broker sends the client a unique session ID and the port where the worker is listening. This information is located in a dedicated session file at the client. The broker then sends a **1ssi** command to the DSCLI in the background. The output of this command is sent back to the client. Use this output to verify whether the session was opened against the correct DS8000 storage system.

A session can also be closed. The command **stopSession.pl** sends an appropriate request to the corresponding worker process that quits the DSCLI and terminates the process.

Users

Multiple users can be defined in DSCLI for each storage system. Users can also be defined using the DSCLIBroker. To do so, define an additional stanza in the DSdev.cfg stanza file for the same storage system. The user name must be declared (Example 3 on page 7). The session must be started with that user name supplied using the option **-user** as shown in Example 13.

Example 13 Start session for a dedicated user

```

$ startSession.pl -box ATS_3 -user admin03
-----
Name          ID                Storage Unit      Model WNN          State  ESSNet
-----
ATS_20780 IBM.2107-7520781 IBM.2107-7520780 922   5005076303FFC1A5 Online Enabled
dscli>
-----

Session on port 55559 for ATS_3 with id 1311037799 was created successfully!
$ _

```

Although with DSCLI a user can log in multiple times to the same storage device, with DSCLIBroker only one session per storage device is allowed.

The authentication to the DSCLI is done using a password file, which must be generated manually in DSCLI. The password files must be located in the directory `./pwfiles` of the home directory of the DSCLIBroker scripting framework. This password file is an encrypted file, and access should be restricted to read access for the administrator only. The password file is generated with the DSCLI command **managepwfile**. See the DSCLI online help or the DSCLI documentation for details.

DSCLIBroker maintenance

The DSCLIBroker server is an autonomic process. However, there are a few maintenance tasks you need to perform.

The most important task is to start and stop the broker daemon using the script **Broker.pl start**. After the broker daemon is started, the DSCLIBroker server is ready to receive requests for starting sessions from any client. To clean up the DSCLIBroker server, run the same script with the parameter **stop**. This parameter stops all worker processes running in the background, and then stops the broker daemon itself.

To monitor the activities of the broker and the worker processes, a logging mechanism has been implemented. The log files are located in the directory `~/log` in the home directory of the DSCLIBroker installation. A numbered index is appended to the log file for each day. The log files are kept for 10 days before they are deleted.

Licenses

The DSCLlbroker is licensed per function and number of storage devices that are managed by the DSCLlbroker. For this reason, the serial numbers of the storage devices must be registered. There is no capacity-based license. For migrations, the DSCLlbroker can be ordered as a tailored STG Lab Service, in which the DSCLlbroker is available for no additional fee.

Programming techniques

The DSCLlbroker scripting framework provides all functions of the DS8000 Copy Services, and some additional functions. However, the true function of DSCLlbroker is to allow you to write your own code that takes advantage of the framework. Writing code is especially of interest for migrations, because certain operations cannot be covered by standard tools and software components provided with the storage devices.

Using DSCLlbroker scripts

The DSCLlbroker scripts are perl programs that can be used in shell scripts. These shell scripts are the easiest way to write scripts. With this method, you can implement interactions between applications or middle ware components and the storage devices. The following are a few examples of what you can do with scripts:

- ▶ Stopping an application before taking a FlashCopy and then restarting the application afterward
- ▶ Alter a database into backup mode before failing over to the remote storage system
- ▶ Implement an interactive script that guides an administrator through a migration scenario

Example 14 shows a simple interactive migration script.

Example 14 Simple interactive migration script

```
1 #!/bin/ksh
2 #
3
4 echo "Validating the paths"
5
6 ValidatePaths.pl -n appl_ab
7
8 echo -n "Are all paths in correct state? [y/n]"
9 read yn
10
11 if [[ $yn = 'y' ]]
12 then
13     echo "Establish Global copy to replicate"
14
15     mkpprc.pl -n appl_ab -type gcp
16     lspprc.pl -n appl_ab -waitnull
17
18     echo "All tracks has been copied"
19     echo -n "Ready to fail over to remote [y/n]"
20     read yn
21     if [[ $yn = y ]]
22     then
23         failoverpprc.pl -n appl_ab -p
```

```

24     else
25         echo "Migration aborted"
26         exit 3
27     fi
28 else
29     echo "Please fix paths issues first"
30     exit 2
31 fi

```

In line 6 the paths are checked for the status and if all paths defined in the repository are available, and the results are displayed. In line 9, you are asked by the script to confirm the paths status before the script continues. If not, the script quits with a message. Otherwise the Global Copy is established in line 15 using the DSCLlbroker script `mkpprc.pl`. The synchronization of the Global Copy is monitored in line 16 using `!spprc.pl -waitnull`. With this option, the script waits until the synchronization completes. In line 19, you are asked if everything is ready to fail over. If not the script stops with a message. If you enter yes, the script `failoverpprc.pl` is applied.

You can use DSCLlbroker scripts in other script languages like python. In this case, system calls must be used that pass the command execution to the shell.

Using DSCLlbroker libraries

Using DSCLlbroker libraries is the most effective way to write user customized scripts. The libraries are written in the perl language, which means the program must also be written in perl. No special perl programming skills are required. However, the knowledge of object and complex data structures, and nested hashes can be helpful.

DSCLlbroker libraries provide additional functions that can be helpful to implement automated migrations tasks. DSCLlbroker libraries can be helpful in the following situations:

- ▶ To provide the correct information so the scripts make the correct decisions and implement a valid logic for the migration. The repository of the DSCLlbroker can contain this information, and the libraries provide functions to retrieve it from the repository.
- ▶ To obtain current information from the storage devices, for example to retrieve the status of paths or copy pairs.
- ▶ To run storage commands using the broker that are not related to copy services functions. For example, you can change the host connection to a target host after a successful replication of the data.

Retrieving information from the repository

In this example, an operation in a migration scenario requires details of the Copy Service configuration. This information can be retrieved from the repository of the DSCLlbroker using the library function `getConfig` from the module `DSlib.pm` (Example 15).

Example 15 Retrieve information from the repository

```

use DSlib;

...

my @RESULTS = getConfig ('CopyPairs', 'name', 'MycopyPairs');
for my $pairs (@RESULTS) {
    $source = $pairs->{Source};
    $target = $pairs->{Target};
}

```

```
...
...
}
```

In this example, the source and the target volumes of a copy relation are retrieved. Include the `DSLlib` library with the `use` statement somewhere at the top of your program. With `getConfig`, the `CopyPairs.cfg` stanza is queried for a certain relation as shown in the example. The result is an array of hashes, where each key of a hash element corresponds to a tag entry of the stanza. The value in each hash entry is the data you are looking for. To unpack this data, a simple `for` loop is used.

Getting information from the storage devices

Using the DSCLI, the usual way to obtain information about the storage configuration or status information is to use an appropriate list command such as `lspprc`. The default output format is a formatted table. However, when the script needs only specific values, this output format spends extra time parsing through the data. To make processing easier, the additional output formats stanza, delimiter, and XML are provided by the DSCLI.

The DSCLIBroker uses the XML output format from the DSCLI to parse for a required parameter. In the following example, the Out-of-Sync values for a copy relation are collected from the output of the `lspprc.pl -sum` command (Example 16).

Example 16 Parsing XML output from DSCLI

```
use DSPPRC;

#
# Create two new boxes because it is a remote copy
my $source=DSbox->new();
my $target=DSbox->new();

#
# Create PPRC thingy
my $PPRC=DSPPRC->new($source, $target);

#
# send 'lspprc -fmt xml' command using the pairs of 'myCopyRelation'
my @SCAN=$PPRC->scanpprc('myCopyRelation','', '-1');

#
# Collect the OOS from each pair
my $totalOOS=0;
for my $pair (@SCAN) {
    totalOOS = $totalOOS + $pair->{outsynctrks};
}

print "Total number of OOS: $totalOOS\n";
```

The `scanpprc` function generates a `lspprc` command with the option `-1` for DSCLI, where the output format XML is requested. The XML format is transformed into an array of hashes where each hash represents the output of one pair relation from the `lspprc` output. To dissolve this structure and collect the Out-of-Sync tracks, a `for` loop searches through all hashes and lists the Out-of-Sync tracks as shown.

Running storage commands

The main focus of the DSCLIBroker functionality is the Copy Services functions for FlashCopy, Remote Copy, and Global Mirror, and combinations thereof. This means that the corresponding library modules are generating the DSCLI commands that are then sent to the broker. The broker then forwards the commands to the worker daemon with the DSCLI in the background.

However, the underlying libraries where the commands are sent to the broker can be used to issue any command. You can use them to allocate volumes, create volume groups, and so on.

Example 17 shows how to set a host connection for a target host system.

Example 17 Making a host connection

```
use DSbox;

#
# Create a box object
#
my $box=DSbox->new('ATS_1');

#
# Create DSCLI script
#
$box->WriteDSCLIScript('managehostconnect -volgrp v12 42');
$box->WriteDSCLIScript('lsconnect -volgrp v12 42');

#
# Execute DSCLI script
#
my $ret=$box->DoDSCLIcommand ('server');
exit($ret);
```

In this script, an object **\$box** of the storage device 'ATS_1' is allocated. All DSCLI commands are generated and run using this object. Any valid DSCLI command can be defined with the **\$box->WriteDSCLIScript** function.

Attention: All DSCLI commands that are sent to the broker must not be interactive. When using removing commands, such as **rmpprc** and **rmpprcpath**, make sure the **-quiet** option is used.

In this example, two DSCLI commands are collected into a script that is stored in the object. This DSCLI script is sent and executed to that storage device using the **\$box->DoDSCLIcommand** function.

Automation techniques

The DSCLlbroker provide a wide range of ways to automate migration tasks. Depending on the type of migration and the platforms and applications involved, you can reduce an entire migration down to a single operational task. This section covers the following automation techniques using DSCLlbroker:

- ▶ Subjects for writing automation scripts
- ▶ Generating DSCLI scripts
- ▶ Using control files

Subjects for writing automation scripts

The DSCLlbroker itself offers a certain level of automation, due to the organization of the data in the repository. The scripts included with the DSCLlbroker take advantage of this organization, but they mostly mimic DSCLI basic commands. Using the framework of the DSCLlbroker, automation can be more efficient, allowing a convenient and save migration operation.

Typically a migration consists of several steps of storage operations and operations that must be done on the servers. To determine where automation can be implemented, the technical workflow and the responsibilities must be defined first. Typically, you have a classification of organizational responsibilities in place, which is reflected in a dedicated storage management, server management and application management. Typically automation does not go across the management boundaries. A meaningful automation includes all steps that can be handled by a single operator. End the script at the hand over to the next system management instance in the workflow.

DSCLlbroker is typically assigned to the storage management organization. However, you can integrate the DSCLlbroker into other storage management automation tasks. For the storage management tasks, automation can be implemented for the following situations:

- ▶ Automating the storage migration
- ▶ Generating repository data

Automating the storage migration

The storage migration tasks are related to all tasks that must be applied against the storage subsystems. Tasks that can be run in a row can be implemented in a script as is. If you need to hand control over to another organization, you can have the automation script exit. Alternately, you can have the script wait at a user prompt until control comes back to the storage system management. The storage operator then confirms the return of the control to the script and it can continue with the next steps.

Generally, implement simulation mode for the migration scenario so you can review the steps of the script before they take effect. During this process, print out detailed logging information and the DSCLI commands. In the simulation mode, no DSCLI commands are run. However, any commands that are display information can be issued and the output checked for accuracy before the real execution.

Generating repository data

It is mission critical for migration using the DSCLlbroker that the data in the repository is correct. Otherwise the correct volumes for a single application or a group of applications might not be copied to the correct target volumes. In large migration projects that take weeks or months, the production environment will probably change due to data growth or other reasons.

Typically, migration projects at that size are organized in waves, where groups of servers or applications are migrated in one go. Before starting a migration of such a wave, create the whole set of repository data for that wave. This guarantees that changes to the production environment are considered every time.

You can maintain the repository data by implementing a script that generates the data using the scripts mentioned in , “Additional useful scripts” on page 14. These scripts require a list of volumes that are mapped to the applications and the migration wave, which is applied next as input.

Important: You need to know which volumes are subject of the migration and which applications are affected before generating repository data.

Generating DSCLI scripts

In certain cases DSCLIBroker cannot be installed in your environment because of internal restrictions such as installation policies. In this case, you can use DSCLIBroker to support the migration by generating the DSCLI commands for each step using simulation mode (Example 12 on page 13). The output of the simulation mode of each DSCLIBroker script can then be used as a series of steps to complete the migration.

The DSCLIBroker provides a script that can generate these DSCLI scripts automatically. Therefore the repository data for the copy pairs, the storage devices, and optionally the paths must be available. When all the scenarios are defined, all DSCLI commands are extracted and collected in a control file. This control file is used to produce all DSCLI commands with the required parameters and volumes. Example 18 shows that a control file containing all commands is required for a Global Mirror migration.

Example 18 Control file example that created all commands for Global Mirror

```
#
# cmd          option          relation  outputfile
mkpprc.pl      ab                LBG_mkpprc
mkflash.pl     bc                LBG_mkGMflash
mksession.pl   ab                LBG_mksession
pausepprc.pl   ab                LBG_pausepprc
failoverpprc.pl ab                LBG_failoverpprc
reverseflash.pl -fast -tgtpprc    bc                LBG_reverseflash
lspprc.pl      ab                LBG_lspprc
lspprc.pl      -r                ab                LBG_lspprc_remote
lspprc.pl      -l                ab                LBG_lspprc_long
lsflash.pl     bc                LBG_lsflash
lsflash.pl     -l                bc                LBG_lsflash_long
rmpprc.pl      ab                LBG_rmpprc
rmpprc.pl      -direction reverse ab                LBG_rmpprc_remote
rmsession.pl   ab                LBG_rmsession
```

The DSCLI commands are written in text files into a certain directory. Each text file holds the commands for a single step of the scenario. Finally the DSCLI scripts themselves must be placed in the directory where the command files are located.

Using control files

Using control files with the DSCLlbroker allows more automation. This automation can be used when you are performing multiple migrations and many different scenarios apply. Instead of writing a script or a program, control files are defined for each scenario.

In Example 18 on page 22, a control script was used to generate the DSCLI script. The format of the file used in this example is generally the same as for any other control file. The required parameters are the command itself, a list of parameters, and the stanza entry to which the operation applies. The other fields in the control file are optional, for example a name for a log file or a comment.

A single script can be used to read the content of a selected control file and run, step by step, the listed commands with their parameters. The control file can be selected using a command-line option. For each execution for a scenario, write logging information into a dedicated log file. Every DSCLlbroker script has a **-banner** option, which allows you to supply comments that are printed at the top of each execution.

The advantage of this approach is that only one program must be developed in the beginning of the migration project. All migration scenarios are defined and maintained using the control files. No programming is required when a scenario must be changed.

Considerations about configuration data

This addresses considerations about the data that must be provided to the DSCLlbroker. This data is configuration data of the copy relations of the volumes to be migrated. Use the following steps to provide the data to the DSCLlbroker:

1. Define the data input format
2. Write a script to read in the configuration data
3. Generate the repository data using the appropriate DSCLlbroker scripts

Data sources

For a migration using Copy Services, application data must be transmitted from a set of source volumes to corresponding target volumes. Typically, the target volumes are on a remote storage device. You need the volume information and the definition of the connectivity, including the communication ports, zoning, and path relations, to extract data for the DSCLlbroker repository.

Usually you maintain this information using a data management system such as IBM Tivoli® TotalStorage Productivity Center for Replication, or even just a spreadsheet. In any case, a data export function that provides comma-separated value data is needed to import the data into the DSCLlbroker repository.

The success of every migration depends on the quality of the data in the repositories. The information in the repository must represent the exact production environment. Use DSCLlbroker as an interface for each migration. It is a vital connectivity between the production environment and the migration environment. Re-import the production data using this interface into the repository of the DSCLlbroker before each migration

Importing data to the repository

If the migration comprises many applications and servers, the whole migration project can take several weeks or even months. Normal production cannot be paused for that long, even

on the applications that will be migrated. Therefore, the repository data might change due to capacity upgrades or any other configuration changes. The data in the repository of the DSCLlbroker must be adopted accordingly before the migration can start.

Assuming you keep your configuration database at the most current state, import this data to the DSCLlbroker repository just before the next migration is started. A script can read a csv export of your configuration data and generate a whole new set of the DSCLlbroker repository automatically. Use the repository generation scripts as described in “Generating repository data” on page 21.

Example migration using DSCLlbroker

This section shows an example of a real migration that was successfully accomplished for an IBM client. The DSCLlbroker is engaged during the analysis after the requirements for the needed automation are discovered.

Overview

The client was running several data centers in a metropolitan area. In a consolidation project, data center sites must be migrated to a new, larger site. In this context, two DS8000 storage devices and the connected servers must be moved to the new data center location. Both storage devices were the primary storage of a Global Mirror configuration. The new location was intended to be the new primary site, while the target site for the Global Mirror remains.

The whole migration comprised several thousand primary volumes and hundreds of applications. The server platforms were mostly IBM AIX® -based, but included VMware ESX server, OpenVMS, and zSeries server. The Global Mirror was managed using IBM TotalStorage Productivity Center for Replication. After the migration, a new TotalStorage Productivity Center for Replication server was established. The migration was organized in groups of applications that must be moved one by one to the new location.

Figure 5 shows the basic structure of the migration.

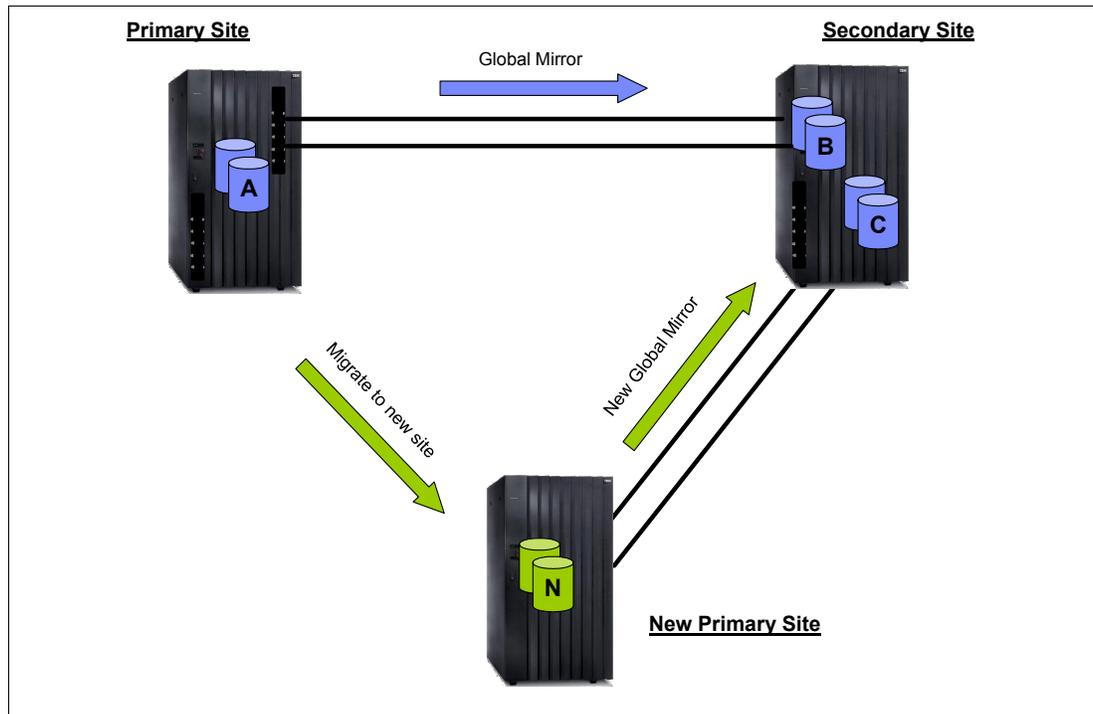


Figure 5 Migration overview

Because of the huge number of applications, the projected duration of the whole project was several months. The main reason for this long time was the data center logistics like change management, hardware ordering, and installation management. Also, the client wanted the current production to be impacted as little as possible.

The general approach was to establish another Global Copy at the new data center by cascading the target volumes of the current Global Copy. During the initial copy phase, the production running at the original volumes was not affected. The volumes that had to be removed from the Global Mirror session and the direction of the new Global Copy reversed. The Target volumes and the FlashCopy relation at the secondary site were reused for the new Global Mirror. The new Global Mirror was started at the new data center site.

Analysis

In the analysis phase, the planned scenario had to be proofed for feasibility. The production environment was analyzed and dependencies discovered. In this case, the applications had a strong dependency on the server hardware because most applications where hosted in logical partitions LPARS that shared hardware. Therefore, the applications had to be migrated in groups according to their server hardware.

The original idea for the migration was to use TotalStorage Productivity Center for Replication to establish the cascaded Global Copy relation and to transfer that relationship to the new Global Mirror session. But it is not possible to establish the cascaded Global Copy relation to the new data center site without removing the current Global Mirror session. Therefore all migration steps were planned using DSCLI commands.

Given these major topics, the analysis was that automation had to be implemented for the following major reasons:

- ▶ To move the correct set of volumes for each application
- ▶ To avoid human error and guarantee consistency during command execution

DSCLbroker provides a solution for both these concerns. Mapping applications to volumes is covered by supplying the repository with the correct data. In addition, the scripting framework was used to write scripts that fulfill the automation requirements.

Test and develop

The migration environment was rebuilt in an IBM lab. This test environment was used to develop and test the entire migration scenario. In addition to the scenarios for migration, scenarios to roll back to normal production if problems prevent migration from completing were.

Scripts to generate the repository data and the migration steps were developed and tested. For the repository data, the customer provided the TotalStorage Productivity Center for Replication session export. They also provided a spreadsheet with the applications, the volumes, and the group they were to be migrated with. Using both data sources, the repository data was generated for all entities.

The test environment was also used to educate the client about using the automation scripts and provide hands-on training.

Planning

With the tools and the scenarios developed in the testing phase, the final planning was finalized. All information was put in place and used to generate the instructions for every required production change management task.

A detailed time line plan was finalized after the test and development phase. All roles were assigned so the complete migration could be conducted by the customer.

Running

The migration execution was divided in to two general steps.

A trial run a couple of days before the go-live migration was scheduled. In this trial run, the migration was run until the data was failed over from the storage perspective to the new data center. However, production was continued at the primary site. At the new data center, the applications were started using the migrated data and a health check was performed. After all tests were passed, the application was ready for the go-live migration.

In the second step, the go-live migration was run. This full migration ended when the production was started in the new data center.

Validating

Most validation was done during the trial run. This validation made sure that the data was replicated as expected, which was the final approval for the go-live migration.

Even with the go-live migration, the health checks were been issued as well to double check the success of the migration. During these validation steps, the data at the original production was preserved in case the client wanted to back out.

Authors

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Peter Klee is an IBM Professional Certified IT specialist in IBM Germany. He has many years of experience in Open Systems platforms, SAN networks, and high end storage systems in huge data centers. He formerly worked in a large bank in Germany where he was responsible for the architecture and the implementation of the disk storage environment. This environment has included the installation from various storage vendors, including EMC, HDS, and IBM. He joined IBM in 2003, where he worked for Strategic Outsourcing. Since July 2004, he has worked for ATS IBM System Storage® Europe in Mainz. His main focus is Copy Services, Disaster Recovery, and storage architectures for IBM DS8000 in open systems environments.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks publications

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new IBM Redbooks® publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>

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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

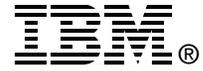
COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright International Business Machines Corporation 2015. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This document REDP-5155-00 was created or updated on October 15, 2015.



Send us your comments in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400 U.S.A.



Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	IBM®	System Storage®
DS6000™	Redbooks®	Tivoli®
DS8000®	Redpaper™	
FlashCopy®	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

Other company, product, or service names may be trademarks or service marks of others.



REDP-5155-00

ISBN 0738454605

Printed in U.S.A.

Get connected

