Creating Hot Snapshots and Standby Databases with IBM® DB2 Universal Database™ V7.2 and EMC TimeFinder™

Aug 17, 2001

EMC PARTNUM: JM52094
IBM TR-74.180

John Macdonald
EMC Corporation

Enzo Cialini
IBM Canada Ltd.
IBM Toronto Lab
Disclaimers and Trademarks

Copyright © 1999, 2000, 2001 EMC Corporation and IBM Corporation. All rights reserved.

Printed 8/17/2001

EMC and IBM believe the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” - NEITHER EMC CORPORATION NOR IBM CORPORATION MAKE ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND BOTH SPECIFICALLY DISCLAIM IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

The furnishing of this document does not imply giving license to any IBM or EMC patents.

References in this document to IBM products, Programs, or Services do not imply that IBM intends to make these available in all countries in which IBM operates.

Use, copying, and distribution of any EMC or IBM software described in this publication requires an applicable software license.

Trademark Information

EMC, EMC², and Symmetrix are registered trademarks and TimeFinder and where information lives are trademarks of EMC Corporation.

IBM, AIX, DB2, DB2 Universal Database, and RS/6000 are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Windows is a registered trademark of Microsoft Corporation in the United States, other countries, or both.

All other trademarks used herein are the property of their respective owners.
Table of Contents

Introduction ................................................................................................................................................. 1

TimeFinder Business Continuance Sequence .......................................................................................... 1

DB2 UDB Suspend I/O Overview .............................................................................................................. 1

DB2 UDB Command Description .............................................................................................................. 2

Test Configuration ...................................................................................................................................... 3

Configuration Setup Tasks and Management Procedures ......................................................................... 5

Initial Device Setup Tasks (Both Systems) ............................................................................................... 5

Configuring LVM on the AIX Systems ....................................................................................................... 7

IBM DB2 UDB Storage Configuration ..................................................................................................... 11

Finishing Setup on the Secondary System ............................................................................................. 13

Establishing and Splitting BCVs for an IBM DB2 UDB Database .......................................................... 15

Methods and Scripts to Manage the Consistent Instant Split .................................................................. 17

Scenarios ................................................................................................................................................... 22

Scenario 1 Making a Snapshot Copy of a DB2 UDB Database .......................................................... 22

Scenario 2 Initializing a Standby Copy of a DB2 UDB Database ........................................................ 23

Scenario 3 Offload Database Backup Processing with a BCV Copy ...................................................... 24

Scenario 4 Managing an IBM DB2 UDB Standby Database with Automated Log Forwarding .......... 25

Scenario 5 Switching between Primary and Standby Systems with EMC TimeFinder ...................... 29

Appendix A db_split Script ................................................................................................................. 32

Appendix B sync_db Script ................................................................................................................ 39

Appendix C Alternative Scripts for AIX Device Lists ........................................................................... 43
Introduction

Mission-critical database systems must operate 24x7 with the highest degree of availability possible. As databases increase in size and ad hoc queries place more demands on the continued availability of the system, the time and hardware resources required to back up and recover databases grows substantially while the maintenance windows are either drastically reduced or disappear completely. The instant split feature of EMC TimeFinder software and the Database Suspend I/O features available in IBM® DB2 Universal Database™ (UDB) combine to provide highly available, advanced database technical architectures to meet these demands. We demonstrate the use of these features to create a coherent copy of a continuously available DB2 UDB database to meet the increasing availability requirements demanded in today's business environment.

This paper takes you through a sample configuration setup and usage of IBM DB2 UDB Enterprise Edition (EE) V7.2 with EMC TimerFinder™.

TimeFinder Business Continuance Sequence

With EMC TimeFinder, a Symmetrix® standard device can be paired with a Business Continuance Volume (BCV) device. A BCV is an independently host-addressable device that is a point-in-time copy of its paired standard device. A standard device and a BCV can be in a number of states. Initially, the BCV is established to the standard device, making a BCV pair. TimeFinder copies all tracks from the standard device to the BCV. While in the establishing/established state, the BCV becomes unavailable from its original host address. Once established, the BCV copy can be split from the standard device, which again makes it available to the host, for other tasks. Incremental establish resynchronizes a previously split BCV/standard device pair. For an incremental establish, TimeFinder copies only updated tracks from the standard to the BCV device, and refreshes any BCV tracks that were changed. During this process, the BCV is not available at its original host address. Similarly, restore and incremental restore copy all tracks, and only changed tracks, respectively, from the BCV to the standard device, during which process the BCV is not available at its original host address.

TimeFinder version 4.2 provides the capability of splitting a group of BCVs instantly. Earlier software required the System Admin to wait for a split operation to complete for a group of BCVs, before subsequent operations could be carried out. Splitting the BCV device group from the production volumes can now be completed as a coherent background process. Essentially, no delay to the host is incurred for this operation.

DB2 UDB Suspend I/O Overview

DB2 UDB V7.2 has implemented a Suspend I/O command that provides the capability to use split mirroring technology while DB2 is online. This functionality is available on all DB2 UDB platforms (e.g., AIX®, Solaris®, HP/UX®, Windows®). Suspend I/O supports continuous system availability by providing a full implementation for splitting a BCV without shutting down the database. The split copy (i.e., BCV) of the database can be utilized to do such tasks as the following:

1. Provide a transactionally consistent snapshot of the database at the current point in time. This database can be used to offload user queries that don't need the most current version of the database.

2. Provide a standby database that can be accessed as a disaster recovery strategy if the primary database is not available. All logs from the primary database will be applied to the secondary database so that it will represent the most current transactionally consistent version of the primary database.
3. Provide the ability to offload the database backup process from the primary database system. A DB2 backup can be performed on the secondary system. The DB2 backup can then be restored on either the primary system or on another system. Rollforward can then be issued to bring the database to a particular point in time, or until the end of the logs are reached.

4. Provide the ability for a quick primary database restore. The BCV can be reestablished so the primary copy is restored to the initial data at the time of the split. Rollforward can then be issued on the primary database to bring the database to a particular point in time, or till the end of the logs are reached.

DB2 UDB Command Description

Suspend I/O Command
The suspend command (db2 set write suspend for database) suspends all write operations to the DB2 UDB database (i.e., to tablespaces and log files). Read operations are not suspended and are thus allowed to continue. Applications can continue to process insert, update, and delete operations utilizing the DB2 bufferpools. A database connection is required for issuing the suspend command. It is recommended that you maintain the current session for executing the subsequent resume command.

Resume I/O Command
The resume command (db2 set write resume for database) resumes all write operations to the suspended DB2 UDB database. A database connection is required for issuing the resume command.

db2inidb Command
The db2inidb command (db2inidb <db_alias> as <snapshot | standby | mirror >) is required to initialize the copy of the suspended database. You do not require a database connection prior to executing this command. It can be used in the following three cases:

1. snapshot can be applied to the secondary copy, putting it into a transactionally consistent state.
2. standby can be applied to the secondary copy, putting it into a rollforward pending state. DB2 logs from the primary database can then be applied to the secondary database.
3. mirror can be applied to the primary copy after it has been restored from the secondary copy. The primary database will be placed into a rollforward pending state and then DB2 logs can be applied to the primary database.

For further information on regarding DB2 UDB’s Suspend I/O functionality, please refer to the DB2 UDB documentation and Release Notes provided with V7.2 or V7.1 Fixpack 3.
Test Configuration

These procedures were tested using an EMC Symmetrix 8430 running Symmetrix Enginuity version 5566. Two IBM RS/6000® H70 systems were connected via fibre HBA to the Symmetrix system. One RS/6000 system was used as a primary database system. The second RS/6000 system was used as a secondary database system (either as a standby system, or as a testing/analysis system). The following figure illustrates the test configuration.

- Hot Standby
- Take Backups that can be restored on primary
- Offloads Cycles from Primary
- Snapshot for Reporting, Testing, etc.

Figure 1. DB2 UDB and EMC Test Configuration

The two RS/6000 systems were running AIX 4.3.3 and IBM DB2 UDB Enterprise Edition, V7.2.

The interaction between the RS/6000 systems and the Symmetrix was done with AIX BCV support kit 2.0, and the EMC AIX kit 4.3.3.2, both from EMC. The set of program temporary fixes (PTF) listed in the EMC Open Source Matrix as of March 2001 was applied.

The primary system storage for the database was contained on 24 – 4GB Symmetrix hyper volumes (or hypers) with Symmetrix device numbers 000 to 017. The corresponding BCV mirror of this storage was configured on another 24 Symmetrix hypers with device numbers 100 to 117. Each of these hypers was also 4GB. The LUNs representing the BCV hypers were configured for use on the secondary system. Gatekeeper devices are used by SYMCLI commands to communicate with the Symmetrix. Our configuration had 8 small gatekeeper volumes, 4 for each RS/6000 system.

Primary storage devices are also referred to as Standard (STD) devices on the Symmetrix. The 24 STD hypers were configured into 2 volume groups on the RS/6000 systems. The first volume group, drvg, consisted of 20 hypers and was further divided into two logical volumes. The first logical volume, dlv, consisted of 4 hypers, which were used as a journaled file system containing the base data for our database. The next 16 hypers were configured into a logical volume, rlv, which was used as a raw logical volume for the majority of the data in the database. The final 4 hypers were configured into a second volume group containing 1 logical volume, llv, used as a journaled file system to contain the logs for the database.

The following table shows the Symmetrix device group name, AIX volume group name, AIX logical volume name, file system mount points (device name in the case of raw logical volumes), the Symmetrix device numbers used on each system, and the AIX hdisk names assigned the Symmetrix hyper volumes on both systems.
The following were demonstrated using this configuration:
2. Initializing a hot standby database from a live production database.
3. Using a BCV copy to offload the processing requirements for taking a database backup of a live production database.
4. Creating a standby copy of a live database with automatic log forwarding for disaster recovery (which can also be configured to delay log application to protect against data corruption).
5. Maintaining a synchronized BCV copy to be used as a standby database system in the event of system failure.
Configuration Setup Tasks and Management Procedures

A fair amount of planning and configuration goes into setting up the storage for a large database implementation on an EMC Symmetrix. Before you can use Symmetrix Standard or TimeFinder volumes, the physical drives in the Symmetrix must be logically subdivided into hypervolumes, which can appear to your system as physical disks. Symmetrix hyper volumes are further configured as STD, BCV, and other Symmetrix device types. Once this is complete, the devices must be made known to your operating system. This step is operating system (OS) specific; in our case, the OS is AIX 4.3.3. The same procedures could be done on other OSes, such as HP/UX or Solaris, but the commands to manage the devices would need to be changed into the appropriate OS-specific equivalents.

The EMC Solutions Enabler SYMCLI Base Component software version 4.2 is required for the SYMCLI functionality described in this document. SYMCLI commands must be installed and available to the root user doing the initial device configuration and subsequent operational steps. Finally, the SYMCLI database is initialized to include the new devices so that later SYMCLI commands can operate efficiently on those devices.

Document Command Conventions

Commands that must be run by root will be shown with a shell prompt of `root>`. Commands that must be run by the database user (the instance owner or database admin) will be shown with a shell prompt of `dba>`. Commands in bold face are shown as the actual commands issued on one (or both) of the systems.

Initial Device Setup Tasks (Both Systems)

**AIX and Symmetrix Volume Discovery**

The path to the SYMCLI software should be put into root’s default profile:

```
root> export PATH=$PATH:/usr/symcli/bin
```

Additional commands are required when initializing devices:

```
root> export PATH=$PATH:/usr/lpp/Symmetrix/bin
```

Detect new Symmetrix devices on AIX:

```
root> emc_cfgmgr
```

Initialize communication with the Symmetrix, and discover device paths and other device information

```
root> symcfg discover
```

List the available devices so that you can determine how AIX device names are mapped to the Symmetrix volumes available to this host:

```
root> symdev list
```
Set Up Symmetrix Device Groups for TimeFinder Operations

This section describes how to define the members of a group containing a standard volume set and a BCV volume set. You must have the same number of primary and BCV volumes, and they must be added in the order you want them associated together. The volumes to be associated must be the same size. Use the same procedure on both systems so that both will use the same group names. SYMCLI commands takes these group names as parameters.

General Procedure

Values required to run procedure:

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Name for the Symmetrix Device Group consisting of STD Symmetrix volumes to be mirrored with BCV volumes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSTART</td>
<td>First standard volume (3-digit hex Symmetrix device number)</td>
</tr>
<tr>
<td>SEND</td>
<td>Last standard volume (3-digit hex Symmetrix device number)</td>
</tr>
<tr>
<td>BSTART</td>
<td>First BCV volume (3-digit hex Symmetrix device number)</td>
</tr>
<tr>
<td>BEND</td>
<td>Last BCV Volume (3-digit hex Symmetrix device number)</td>
</tr>
</tbody>
</table>

Create the group for the production volumes:

    root> symdg -type REGULAR create GROUP

Add a range of primary devices to the group:

    root> symld -g GROUP -RANGE SSTART:SEND addall dev

Repeat this step if you want to combine nonconsecutive blocks of volumes into a single STD group.

Associate a range of BCV devices with the group:

    root> symbcv -g GROUP -RANGE BSTART:BEND associateall dev

Repeat this step if you want to combine nonconsecutive blocks of volumes into a single BCV group.

Performed on Both Systems

    root> symdg -type REGULAR create data
    root> symdg -type REGULAR create logs
    root> symld -g data -RANGE 000:013 addall dev
    root> symld -g logs -RANGE 014:017 addall dev
    root> symbcv -g data -RANGE 100:113 associateall dev
    root> symbcv -g logs -RANGE 114:117 associateall dev
How you group Symmetrix devices is a function of the operations you plan to perform. We created a separate group for each of the data and the logs because we were demonstrating both snapshot and standby options. If we had only been using one option, we would have created a single group: for the snapshot option only, it would have contained both data and logs; for the standby option only, it would have contained only the data (logs would not be copied at all).

## Configuring LVM on the AIX Systems

The AIX operating system provides a Logical Volume Manager (LVM) to manage large groups of devices as volume groups. Volume groups are further subdivided into logical volumes to be used for file systems or raw logical volumes. If you are using a different Volume Manager, you may use different commands to accomplish the same purposes.

To perform the LVM configuration, we used the AIX devices we discovered earlier (with `symdev list`) to create volume groups, logical volumes within the volume groups, and file systems on logical volumes as required. For our demonstration, we configured 2 volume groups, 2 file systems, and 1 raw logical volume. This configuration combines filesystems and raw devices to demonstrate the use of both.

### Establish the BCV Mirror

To simplify configuration on the secondary system, the rest of the LVM and database setup steps on the primary system were done with BCV volumes established. This makes the setup of the secondary system much simpler – as we will see later in the section Import the Disk Configuration on the Secondary System.

#### General Procedure

Value required to run procedure:

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Name for the Symmetrix Device Group consisting of STD Symmetrix volumes to be mirrored with BCV volumes</th>
</tr>
</thead>
</table>

Initial establish of a group of STD - BCV device pairs

```bash
root> symmir -g GROUP -noprompt -full -exact establish
```

Repeat this as needed for each group.

(Optional) Wait until the process completes.

```bash
root> symmir -g GROUP -i 30 verify
```

### Performed on the Primary System

```bash
root> symmir -g data -noprompt -full -exact establish
root> symmir -g logs -noprompt -full -exact establish
```
We didn't wait for the establish to complete, but continued with other setup tasks. We wanted the results of those tasks to be included before we split the BCV copies.

**Configure AIX Device Variables**

To be used, a number of commands need to be given a list of physical devices. If you are only using a small number of physical devices for each purpose, that is no problem. If you are using groups of devices, as in our demonstration case, it is convenient to set variables to contain the lists of devices.

Performed on the Primary System

```bash
root> datadk='hdisk7 hdisk8 hdisk9 hdisk10'
root> rawdk='hdisk11 hdisk12 hdisk13 hdisk14'
root> rawdk='$rawdk hdisk15 hdisk16 hdisk17 hdisk18'
root> rawdk='$rawdk hdisk19 hdisk20 hdisk21 hdisk22'
root> rawdk='$rawdk hdisk23 hdisk24 hdisk25 hdisk26'
root> logsdk='hdisk27 hdisk28 hdisk29 hdisk30'
```

Alternative scripts for performing this task, suitable for long lists of devices, that might not have consecutive IDs, are shown in Appendix C – Alternative Scripts for AIX Device Lists.

**Create Volume Groups Using AIX Device Variables**

**General Procedure**

Values required to run procedure:

<table>
<thead>
<tr>
<th><strong>VGNAME</strong></th>
<th>Name for the volume group</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PPSIZE</strong></td>
<td>Size of partitions to be used</td>
</tr>
<tr>
<td><strong>DKLIST</strong></td>
<td>List of physical volumes to be included in the group</td>
</tr>
</tbody>
</table>

The partition size you choose depends on the size of the hyper volumes. There is a limit of 1016 partitions per physical volume, so your partition size must be as least 0.1% of the total size you may wish to use on the volume. The partition is the unit of allocation for space on the volume, so a large size will waste more space because a request is rounded up to the partition size. Our hyper volumes were 4GB, the default partition size of 4MB was just barely too small. However, we were allocating entire volumes to dedicated purposes, so a larger size (64MB) was convenient to work with.

```bash
root> mkvg -f -y VGNAME -s PPSIZE DKLIST
```

Performed on the Primary System

```bash
root> mkvg -f -y drvg -s 64 $datadk $rawdk
root> mkvg -f -y lvg -s 64 $logsdk
```

**Create Logical Volumes to Contain Journalled File Systems**

**General Procedure**

Values required to run procedure:
LVNAME  Name for the logical volume  
MAXPP  Maximum number of partitions to be used  
VGNAME  Name of the volume group to contain the logical volume  
DKLIST  List of physical volumes to be included in the logical volume

Specify the number of partitions to be available to the logical volume (which we set to almost all of the available space). We set the space initially to 1, so that when a journalled file system is built, the logical volume to be used for the journal is placed in the middle of the available space. This is not especially critical for volumes that are on a Symmetrix, but it doesn't hurt and can have some beneficial effect.

Create a Logical Volume with the File System Volumes:

```
root> mklv –y LVNAME –tjfs –x MAXPP VGNAME 1 DKLIST
```

Performed on the Primary System

```
root> mklv –ydlv –tjfs –x250 drvg 1 $datadk
root> mklv –yllv –tjfs –x250 lvg 1 $logsdk
```

Create Logical Volumes to Contain Raw Disks

General Procedure
Values required to run procedure:

<table>
<thead>
<tr>
<th>LVNAME</th>
<th>Name for the logical volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXPP</td>
<td>Maximum number of partitions to be used</td>
</tr>
<tr>
<td>VGNAME</td>
<td>Name of the volume group to contain the logical volume</td>
</tr>
<tr>
<td>ACTPP</td>
<td>Initial number of partitions to be used</td>
</tr>
<tr>
<td>DKLIST</td>
<td>List of physical volumes to be included in the logical volume</td>
</tr>
</tbody>
</table>

Specify the number of partitions available to the logical volume (which we set to almost all of the available space). We allocated the space immediately here. No other logical volumes are involved for a raw volume, so no special arrangements to locate them within the volume group are required.

```
root> mklv –y LVNAME –x MAXPP VGNAME ACTPP DKLIST
```

Performed on the Primary System

```
root> mklv –yrlv –x1000 drvg 1000 $rawdk
```
Create Journalled File Systems

General Procedure
Values required to run procedure:

<table>
<thead>
<tr>
<th>LVNAME</th>
<th>Name of the logical volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNTPT</td>
<td>Mount point where the file system will be used</td>
</tr>
<tr>
<td>AUTO</td>
<td>Specifies whether the file system is to be mounted on reboot, either yes or no; usually yes for the primary system and no for the secondary system.</td>
</tr>
<tr>
<td>ISIZE</td>
<td>Size for inode blocks</td>
</tr>
<tr>
<td>FSSIZE</td>
<td>Final size for the file system (in 512 byte blocks)</td>
</tr>
</tbody>
</table>

Logical volumes to contain file systems now get those file systems created on them. Each file system needs a mount point where it will appear in the computer’s directory structure. The size of an inode block controls the number of files (and directories and devices and so on) that can be created on that file system. The default of 4096 is a bit too small for the size of file systems we were creating. We made the final size of each of our file systems 32,768,000 blocks (of 512 bytes), which is all of the space (slightly under 16GB) on the logical volume. An alternate approach is to use smit fs to create file systems.

```
root> crfs -vjfs -dLVNAME -mMNTPT -AAUTO -prw -tno -a nbpi=ISIZE
root> chfs -a size=FSSIZE MNTPNT
root> mount MNTPNT
```

Performed on the Primary System:

```
root> crfs -vjfs -ddl -m/data -Ayes -prw -tno -a nbpi=8192
root> crfs -vjfs -dll -m/logs -Ayes -prw -tno -a nbpi=8192
root> chfs -a size=32768000 /data
root> chfs -a size=32768000 /logs
root> mount /data
root> mount /logs
```

Make File Systems and Devices Available to DB2 Instance

To permit the file systems and devices to be used by the database instance, the permissions must be set up properly. We show how to provide access to the mount point of the file system, but this is not a requirement. You could create subdirectories from the mount point and have the database created within these subdirectories. However, any other files or directories placed under the mount point will be copied to the BCV volumes along with the database.

Performed on the Primary System:

```
root> chown dba /data /logs /dev/rrlv
```
IBM DB2 UDB Storage Configuration

IBM DB2 UDB needs storage for creating database configuration and control information, and for tablespaces, which will in turn hold tables and other database objects. DB2 UDB provides two types of storage for databases: SMS (System Managed Storage) and DMS (Database Managed Storage), to be used for database. For more information on DB2 UDB storage, refer to the IBM DB2 UDB manuals.

Creating a DB2 Database in the /data File System

General Procedure

Values required to run procedure:

| DBNAME | Name of the database to be created |
| PATH   | Location of the database           |

Start IBM DB2 UDB (if it is not already running):

   dba> db2start

Create the database on the file system:

   dba> db2 create db DBNAME on PATH

Performed on the Primary System

   dba> db2 create db testdb on /data

Configure a Raw Device as DMS in a Database

A raw device can be used to hold data within a database.

General Procedure

Values required to run procedure:

| DBNAME   | Name of the database          |
| TSNAME   | Name to give the tablespace that will contain the raw device |
| TBNAME   | Name of the table in the tablespace |
| TBDEF    | DB2 table definition         |
| DVPATH   | Pathname of the raw device container |
| DVSIZE   | Amount of space available on the device |

Establish a connection to the database:

   dba> db2 connect to DBNAME
Use " " on the next command so that the quotes and parentheses won't need to be escaped to protect them from interpretation by the shell.

    dba> db2 "create tablespace TSNAME managed by database using \n     ( device 'DEVPATH' DEVSIZE )"

Create table(s) in the tablespace:

    dba> db2 "create table TBNAME ( TBDEF ) on TSNAME"

Performed on the Primary System

    dba> db2 connect to testdb
    dba> db2 "create tablespace testraw managed by database \n       using ( device '/dev/rrlv' 60G )"

Any tables created in the tablespace, testraw, will be stored in the DMS tablespace.

Set the Location of Logs

The default location for the DB2 log files is in the SQLOGDIR directory which is a relative path under the database path used during the create database command. You would omit setting the location of logs if you were not using a separate BCV group for the logs and the logpath was contained under the database path.

General Procedure

Values required to run procedure:

<table>
<thead>
<tr>
<th>DBNAME</th>
<th>Name of the database</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPATH</td>
<td>Location for the logs</td>
</tr>
</tbody>
</table>

Modify the default location of the DB2 log files:

    dba> db2 update db cfg for DBNAME using NEWLOGPATH LPATH

Performed on the Primary System

    dba> db2 update db cfg for testdb using NEWLOGPATH /logs

Enable Log Retain and USEREXIT

Additional configuration parameters may need to be set.

To use a standby copy of the database, you need to enable LOGRETAIN.

To automate the archiving and retrieval of log records, you need to enable USEREXIT.

Performed on the Primary System

Enable LOGRETAIN:

    dba> db2 update db cfg for testdb using LOGRETAIN on

Enable USEREXIT:
Finishing Setup on the Secondary System

The configuration of devices, file systems, and the database must now be made known to the OS on the secondary system. This is largely a matter of telling that system to use the configuration already copied onto the BCV volumes from the standard volumes. The first step makes the BCV volumes available to the secondary system by splitting them from the standard volumes. We demonstrate this by stopping the database on the primary system, using the assumption that both systems are being created at the same time. But if you are adding the ability to use a secondary system to an existing database application that is already in production, the steps used later for splitting a live database could be used instead.

Split the BCV Volumes from the Standard Volumes

General Procedure
Value required to run procedure:

| GROUP | Name of a device group to split |

Stop any active applications. This may require actions specific to your particular applications.

Stop the IBM DB2:

```
dba> db2stop
```

(If you have not terminated all applications, but they do not need special termination actions, you can use `db2stop force` to forcibly terminate all running applications.)

Make sure that the `establish` operation started earlier has completed successfully:

```
root> symmir -g GROUP -i 30 verify
```

Split the BCV volume groups:

```
root> symmir -g GROUP -noprompt -instant split
```

Performed on the Primary System

```
dba> db2stop
root> symmir -g data -i 30 verify
root> symmir -g data -noprompt -instant split
root> symmir -g logs -i 30 verify
root> symmir -g logs -noprompt -instant split
```

Import the Disk Configuration on the Secondary System

If you haven't done the initial device setup tasks on the secondary system as described above in the section on Configuration Setup Tasks and Management Procedures, do them now.
The volume manager setup, file system setup, and database setup have already been done on the STD volumes on the primary system, and copied to the BCV volumes; the secondary system merely has to learn about it. Using the AIX volume manager, this step is simple.

**General Procedure**

Values required to run procedure:

<table>
<thead>
<tr>
<th><strong>VGNAME</strong></th>
<th>Name of the volume group being imported</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DISK</strong></td>
<td>Device name of the first disk in the group</td>
</tr>
<tr>
<td><strong>RAWDEV</strong></td>
<td>Name of a raw device (if any) in the volume</td>
</tr>
</tbody>
</table>

Import volume group definition from the physical volumes:

```
root> importvg -yVGNAME DISK
```

If a raw device exits, it needs to have permissions set correctly (the contents of the file systems will have the permissions already set from the mirroring of STD devices on the primary system).

```
root> chown DBOWNER RAWDEV
```

**Performed on the Secondary System**

```
root> importvg -ydrvg hdisk7
root> importvg -ylvg hdisk27
root> chown dba /dev/rllv
```

**Catalog the Database on the Secondary System**

The database is already present on the imported disk volumes, but IBM DB2 UDB on the secondary system does not know about the imported volumes yet. If your database application requires any auxiliary data or files that are stored outside of the database (such as control scripts), they should be set up at this time if this hasn’t been done yet.

**General Procedure**

Values required to run procedure:

<table>
<thead>
<tr>
<th><strong>DBNAME</strong></th>
<th>Name of the database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PATH</strong></td>
<td>Location of the database</td>
</tr>
</tbody>
</table>

Make the database known to the DB2 Instance:

```
db2> db2 catalog database DBNAME on PATH
```

Install database control scripts, auxiliary files, etc. as required to use the database properly on the secondary system.

**Performed on the Secondary System**

```
dba> db2 catalog database testdb on /data
```

Creating Hot Snapshots and Standby Databases
Establishing and Splitting BCVs for an IBM DB2 UDB Database

We now come to the main focus of this document: making a coherent copy of the content of a database while the database is being used. Until the recent changes to IBM DB2 UDB and to TimeFinder, it was recommended that you stop the database, split the BCV, and then restart the database; a procedure which took minutes, or even a significant portion of an hour. For many high-availability database applications, this amount of downtime is unacceptable at any time of the day; yet, there are many valid reasons for wanting to make these copies at regular intervals even at the busiest times.

We now describe three methods of making a BCV copy that reduces the interruption considerably. They differ slightly in the ease with which you can script and maintain them, trading some scripting inconvenience for a reduction in database interruption.

The split command can be used on either a single Symmetrix group, or on a list of Symmetrix devices listed in a file. Depending on how you plan to use the copy, you may have to split the logs at the same time as the data. In other words, you must use two split commands to specify the logs using the group names, or use a device list file to split the logs with a single command. There is a good reason to split the logs with a single split command. When a split is executed, the first stage of the command is to determine the list of device pairs and to confirm that each of those device pairs is in an synchronized state. This confirmation process takes much longer than the actual split operation itself. In our configuration, we found that a split took 10 to 12 seconds for the validation phase, and 1 to 2 seconds for the split. This time was independent of the number of volumes that were being split and of the amount of I/O activity that was happening at the time. The entire split command took 12 to 14 seconds. Depending on the level of activity of your database, suspending writes for a few tens of seconds may be perfectly acceptable. If you really need to limit the pause to the 1 or 2 seconds, you'll need to use an alternate method for invoking the split.

Splitting the BCV Volumes

Verifying That the Establish Has Completed

Before you can split the BCV volumes, they must be fully established. As long as the previous steps were done before you want to split the BCVs, establishing the BCV volumes should already be completed. This check can be used to ensure that it is completed.

General Procedure

Value required to run procedure:

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Name of a device group that will be split</th>
</tr>
</thead>
</table>

Verify the state of the BCV copy:

```
root> symmir -g GROUP -i 30 verify
```

Repeat this for each group.

Performed on the Primary System

```
symmir -g data -i 30 verify
symmir -g logs -i 30 verify
```
Preparing for Reestablish on the Secondary System

If the BCV volumes you want to establish are being used on the secondary system, they must be freed up. You have to stop any processes that are using the volumes, including AIX’s use of file systems mounted from any of the volumes. This whole step is specific to the way you are using the volumes on your system; we do not provide a general procedure. You do not need to take special action to stop these processes cleanly. Because the volumes are going to be established as BCV copies of the primary system, the data on them from the current processes on the secondary system is totally overwritten. So, you may want to kill the processes; an action that would be unthinkable on the primary system with a live database.

General Procedure

Value required to run procedure:

| FILESYS | Name of a file system to unmount |

Find any running DB2 applications

   dba> db2 list applications

Stop them. This may require application-specific procedures. If stopping them does not require application-specific procedures, see the force option below.

To stop the database if all applications have already been stopped:

   dba> db2stop

To stop the database with connected applications:

   dba> db2stop force

Unmount the file systems:

   root> umount FILESYS

Repeat this step for each file system.

Performed on the Secondary System

   dba> db2stop force
   root> umount /data
   root> umount /logs

Reestablish on the Primary System

General Procedure

Value required to run procedure:

| GROUP | Name of a device group that will be established |
Establish each group of devices

```bash
root> symmir -g GROUP -noprompt establish
```

Performed on the Primary System

```bash
root> symmir -g data -noprompt establish
root> symmir -g logs -noprompt establish
```

## Methods and Scripts to Manage the Consistent Instant Split

We present three methods for splitting the volumes. The appropriate method depends on how short an interruption to database writes you must have, and whether you have multiple groups of volumes to be split at the same time.

The first is simple enough that it can be run by hand. The second and third methods are tricky enough that it is recommended to run them using a shell script. The script `db_split`, listed in Appendix A [db_split Script](ftp://ftp.emc.com/pub/symm3000/DB2/db_split_latest) can be used to apply these methods, and provides error checking and recovery for them. It is available from [ftp://ftp.emc.com/pub/symm3000/DB2/db_split_latest](ftp://ftp.emc.com/pub/symm3000/DB2/db_split_latest) for anonymous ftp.

### Force Archive Log (Log Switch)

Regardless of the method you use, you may want to force the active log files to be archived (i.e., switch the logs) just before the split. Such a switch may help you determine the log position that was in effect at the time of the split. This is useful for an active standby system, when you do a periodic BCV copy to get it fully synchronized. However, the log switch and the write suspend are separate actions, so you do not get a guarantee that no log records will be written between those actions.

#### General Procedure

Values required to run procedure:

<table>
<thead>
<tr>
<th><code>DBNAME</code></th>
<th>Name of the database whose logs will be switched</th>
</tr>
</thead>
</table>

```bash
dba> db2 terminate
dba> db2 archive log for database DBNAME
```

### Simplest Split

The simplest split method suspends write activity to the database for a few tens of seconds. In our test configuration, it took 12 to 14 seconds per group split – about 25 seconds in total when two groups were split, or 13 seconds when just one group was split.

#### General Procedure

Values required to run procedure:

<table>
<thead>
<tr>
<th><code>GROUP</code></th>
<th>Name of a device group that will be established</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DBNAME</code></td>
<td>Name of the database to be split</td>
</tr>
</tbody>
</table>

Connect to the database (if not already done)
dba> db2 connect to DBNAME

Suspend database writes:

dba> db2 set write suspend for database

Split a BCV group:

root> symmir -g GROUP -noprompt -instant split

Repeat for each BCV group.

Resume database writes:

dba> db2 set write resume for database

Performed on the Primary System

dba> db2 connect to testdb
dba> db2 set write suspend for database

root> symmir -g data -noprompt -instant split
root> symmir -g logs -noprompt -instant split

dba> db2 set write resume for database

Split with Callout Actions

Using callout actions from the split command lets you avoid suspending database writes for the time required to test the status of the devices involved in the first split. The callout script takes into account that the split command must be run as root, while the database commands must be run as the database instance user. The db_split script creates these callout scripts automatically, complete with error checking. This script is shown in Appendix A db_split Script. It is available for anonymous ftp from ftp://ftp.emc.com/pub/symm3000/DB2/db_split_latest.

This method suspends write activity to the database for a few seconds if only a single group is being split, or for another dozen seconds each if additional groups are being split. In our test database, write activity was suspended for about 2 seconds when one group was split, or 15 seconds when two groups were split.

General Procedure

Values required to run procedure:

<table>
<thead>
<tr>
<th>DBID</th>
<th>dba instance user name - instance name or admin id</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP</td>
<td>Name of a device group that will be established</td>
</tr>
<tr>
<td>DBNAME</td>
<td>Name of the database to be split</td>
</tr>
</tbody>
</table>
Creating Hot Snapshots and Standby Databases

The argument \texttt{-l pre} or the argument \texttt{-l post} (or both) will force a log switch before and/or after the split, respectively. The argument \texttt{-g} specifies that the list of devices to be split are group names (rather than files containing device lists, as shown below). As long as you don’t have files with the same name as your groups, you can omit this argument, but it is safer to specify it explicitly.

**Performed on the Primary System to Split Only Data Group**

\begin{verbatim}
root> db_split -g testdb dba data
\end{verbatim}

**Performed on the Primary System to Split Only Data Group with a Log Switch**

\begin{verbatim}
root> db_split -g -l pre testdb dba data
\end{verbatim}

**Performed on the Primary System to Split Both Groups: Data and Logs**

\begin{verbatim}
root> db_split -g testdb dba data logs
\end{verbatim}

**Split with Callout Actions Using a Device List File**

Using a device list file lets you avoid suspending database writes for the time required to test the status of the devices involved in the split of the second and subsequent groups. The callout script takes into account that the split command must be run as root, while the database commands must be run as the database instance user. The \texttt{db_split} script creates these callout scripts automatically, complete with error checking. This script is shown in Appendix A \texttt{db_split Script}. It is available for anonymous ftp from \texttt{ftp://ftp.emc.com/pub/symm3000/DB2/db_split_latest}.

This method suspends write activity to the database for a few seconds regardless of the number of device volumes are being split. In our test database, write activity was suspended for about 2 seconds when 4, 20, or 24 volumes were being split. To make this minimal suspend time occur even when more than one device list file is provided, a single temporary file is created that contains the contents of all of the multiple files. (If only one file is provided, it is used without a copy.)

**General Procedure**

Values required to run procedure:

<table>
<thead>
<tr>
<th><strong>DEVFILE</strong></th>
<th>a file (or files) containing a list of STD-BCV device pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBID</strong></td>
<td>dba instance user name - instance name or admin id</td>
</tr>
<tr>
<td><strong>SID</strong></td>
<td>Symmetrix ID where the devices reside, the last 3 digits is enough</td>
</tr>
<tr>
<td><strong>DBNAME</strong></td>
<td>Name of the database to be split</td>
</tr>
</tbody>
</table>

\begin{verbatim}
root> db_split [-f] [-l [pre|post]] -s SID DBNAME DBID DEVFILE
\end{verbatim}

The argument \texttt{-l pre} or the argument \texttt{-l post} (or both) will force a log switch before and/or after the split, respectively. The \texttt{-s SID} argument is required when using a device list file. The \texttt{-f} argument specifies that the trailing arguments are files containing a list of device pairs, rather than group names. As long as the files actually exist, \texttt{db_split} gets it right even if the \texttt{-f} argument is omitted. But if there is
any chance you might accidentally name an existing Symmetrix group by misspelling a filename, you should use the explicit argument.

A device list file contains lines, each with two Symmetrix device IDs of 3 hex digits. The first device ID is the standard volume, and the second is its associated BCV. The list can contain devices from more than one group, or you can have more than one device list file. (You might find it useful to keep a device list file for each of your groups and to provide the file for each of the groups to be split.) Multiple files will be merged into a single temporary file so that the split operation is done only once.

**Device List Files on the Primary System**

We had a number of device list files, including device list files for each individual group, as well as a concatenated file for the combination of both groups. We only used the combined file as an argument to the db_split script. The files for our system looked like this:

<table>
<thead>
<tr>
<th>Filename</th>
<th>/tmp/data</th>
<th>/tmp/logs</th>
<th>/tmp/devlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 100</td>
<td>014 114</td>
<td>000 100</td>
<td></td>
</tr>
<tr>
<td>001 101</td>
<td>015 115</td>
<td>001 101</td>
<td></td>
</tr>
<tr>
<td>002 102</td>
<td>015 116</td>
<td>002 102</td>
<td></td>
</tr>
<tr>
<td>003 103</td>
<td>017 117</td>
<td>003 103</td>
<td></td>
</tr>
<tr>
<td>004 104</td>
<td>004 104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>005 105</td>
<td>005 105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006 106</td>
<td>006 106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>007 107</td>
<td>007 107</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008 108</td>
<td>008 108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>009 109</td>
<td>009 109</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00A 10A</td>
<td>00A 10A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00B 10B</td>
<td>00B 10B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00C 10C</td>
<td>00C 10C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00D 10D</td>
<td>00D 10D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00E 10E</td>
<td>00E 10E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00F 10F</td>
<td>00F 10F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010 110</td>
<td>010 110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011 111</td>
<td>011 111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>012 112</td>
<td>012 112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>013 113</td>
<td>013 113</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>014 114</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>015 115</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>016 116</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>017 117</td>
</tr>
</tbody>
</table>
Performed on the Primary System

    root> db_split -f -s 343 testdb dba /tmp/devlist
Scenarios

The following scenarios illustrate various uses of the documented functionality utilizing the secondary system as a snapshot and/or standby database system as well as a hot standby system (ie. a system available to takeover).

Scenario 1  Making a Snapshot Copy of a DB2 UDB Database

Taking a snapshot of a database lets you have a transactionally consistent copy of the database from a single point in time. The database application can continue being used without affecting the copy. The copy can be used on a secondary system. This will not cause extra load on the primary system and it will not affect the database application.

Performed on the Primary System
Suspend, Split, Resume:

root> db_split -f -s 343 testdb dba /tmp/devlist

Performed on the Secondary System
Mount the file systems:

root> mount /data
root> mount /logs

Start the DB2 instance (if not already started):

dba> db2start

Initialize the database as a snapshot:

dba> db2inidb testdb as snapshot

Complete other steps as required by your application. The database is now available for use, for example, for testing or reporting.
Scenario 2 Initializing a Standby Copy of a DB2 UDB Database

Initializing a standby database is almost identical to making a snapshot of the database. The main difference, aside from the db2inidb options, is that the log filesystem does not have to split along with the data filesystem/devices.

Performed on the Primary System

```
dba> db2 connect to testdb
dba> db2 set write suspend for database
root> symmir -g data -noprompt -instant split
dba> db2 set write resume for database
```

Performed on the Secondary System

Mount the file systems:

```
root> mount /data
```

Start the DB2 instance (if not already started):

```
dba> db2start
```

Initialize the database for standby mode:

```
dba> db2inidb testdb as standby
```

The testdb database is now in rollforward pending mode and available to accept archived log files from the primary database system.

When you are ready to take the database out of rollforward pending state issue:

```
dba> db2 rollforward database testdb complete
```
Scenario 3  Offload Database Backup Processing with a BCV Copy

Making backup copies of your database is an important task. However, it puts a significant load on both the CPU and I/O of the system. Offloading this task to the secondary system keeps the task from impacting the performance of the production database system.

Making a BCV Copy for Offloaded Backup

Making a BCV copy for offloaded backup is almost identical to making a standby database copy. It backs the copy up without rolling the logs forward.

Performed on the Primary System

Make a BCV copy

```
root> db_split -f -s 343 testdb dba /tmp/devlist
```

Performed on the Secondary System

Mount the file systems:

```
root> mount /data
root> mount /logs
```

Start the DB2 instance (if not already started):

```
dba> db2start
```

Initialize the database:

```
dba> db2inidb testdb as standby
```

Backup the database:

```
dba> db2 backup database testdb to /dev/tape
```

Note: Although the syntax of the backup command is similar to an offline backup, the backup is in fact treated as an online database backup and can be used to roll forward through the logs.
Scenario 4  Managing an IBM DB2 UDB Standby Database with Automated Log Forwarding

An active standby system maintains its own copy of the database by receiving completed logs forwarded from the primary system and applying them (rolling them forward) on its copy of the database. An active standby system can be set up to provide a delay so that it applies logs only after a suitable time has elapsed.

This system can be used to repair failures within the database itself – for example, if the application program writes invalid data. When such a failure occurs, the standby system can be used to repair the problem and then the repaired copy of the database can be moved back to the primary system.

An active standby system uses far more resources than a passive one does – the secondary system must duplicate the processing of all of the completed logs that the primary system has already processed whereas a passive standby system lets the Symmetrix do the duplication by retaining an established BCV. If the secondary system does not have the same processing capabilities as the primary system, it is possible that the secondary system may be unable to keep up with the stream of logs.

TimeFinder functions are a less integral part of an active standby system than a passive one. A BCV copy can be used to initialize the standby system in the first place. The same BCV can be occasionally reestablished to allow the two systems to be resynchronized (this might be done regularly if the secondary system is too slow to keep up with the actual load). After an error has been determined and repaired on the secondary system, a BCV restore might be used to propagate the repair back to the primary system; however, recovery procedures are very dependent on the circumstances and are not dealt with in detail here.

Initial Data Setup

An active standby system needs to have an initial copy of the data from the primary database. The same setup procedure used in Scenario 2 Initializing a Standby Copy of a DB2 UDB Database is used. The BCV is not established again unless the standby database is refreshed with a new copy from the primary database.

Log Sharing Area

The active standby system also needs to be provided with logs from the primary system as they are completed. We used a directory on the primary system that was exported using Network File System (NFS) to the secondary system to provide joint access to the archived logs. The subdirectories within this shared file system had a structure that provided for future expansion, allowing for the same shared area to be used by multiple databases, even by multiple IBM DB2 UDB Enterprise-Extended Edition (EEE) nodes. The dba account on both systems must have the same account number.

Note: This example illustrates the functionality of log sharing but you can use your existing log archival location.

Performed on the Primary System

```
root> mkdir /big/dba /big/dba/userexit
root> cd /big/dba/userexit
root> for i in archive retrieve applied
root> do
root>     mkdir $i $i/TESTDB $i/TESTDB/NODE0000
root> done
root> chown -R dba .
```
We used SMIT to export /big/dba/userexit to the secondary system. The command smit ran was:

```
root> /usr/sbin/mknfsexp -d /big/dba/userexit -t rw \
    -c secondary -r secondary -B
```

Performed on the Secondary System

```
root> mkdir /big/dba /big/dba/userexit
```

We used SMIT to import /big/dba/userexit from the primary system. The command smit ran was:

```
root> /usr/sbin/mknfsmnt -f /big/dba/userexit \
    -d /big/dba/userexit -h primary -n -B -A \
    -t rw -w bg -Y -Z -X -H -j -q -g
```

### Automating Log Forwarding

We configured the sample DB2 userexit C program that comes with IBM DB2 UDB, found in sqllib/samples/c/db2uext2.cdisk, to do our forwarding. The same program is used on each system. On the primary system, it is invoked when each log is switched out to forward that log to the secondary system. On the secondary system, it is invoked to retrieve logs that have been forwarded so that they can be applied. On the secondary system, we also used a perl script to control when forwarded logs will be applied. In our example, we apply the forwarded logs as soon as they are received, but this script could be modified to delay processing the logs. Logs that are being forwarded from the primary system are stored under the archive directory using a subdirectory named for the database name and node number — these are forced to upper-case for safety on other operating systems. The perl script, which can be modified to cater to any local control needs (such as delaying when logs are processed), moves logs that should be applied to the retrieve directory and then causes IBM DB2 UDB to apply the log using the db2uext2 program. The log is finally moved under the applied directory.

Building the C program is described in the comments at the top of the file.

Note: This example illustrates the functionality of how the log forwarding works. The userexit program logic, and the archival, and retrieval locations used need not be the same, and you can leverage your existing logic/locations.

### Performed on the Primary System

```
dba> cd sqllib/samples/c
dba> cp db2uext2.cdisk db2uext2.c
dba> vi db2uext2.c
```

We set the directory layout being used in the configuration area of the program and left everything else alone:

```
#define ARCHIVE_PATH="/big/dba/userexit/archive/
#define RETRIEVE_PATH="/big/dba/userexit/retrieve/
#define AUDIT_ERROR_PATH="/big/dba/userexit/
```

Then we saved the db2uext2.c file back, compiled and installed it, and informed IBM DB2 UDB that it should be used:

```
dba> cc -o db2uext2 db2uext2.c
```
We copied the program over to the same location on the secondary system. We show it here using the `rcp` program, but you can use the shared file system as a staging area or `FTP` instead.

```sql
dba> rcp db2uext2 secondary:sqlib/adm
```

### Receiving Logs

The perl script `sync_db` is listed in [Appendix B sync_db Script](#) and available by anonymous ftp from `ftp://ftp.emc.com/pub/symm3000/DB2/sync_db_latest`. It can be used to control the processing of logs that have been archived by the primary system. It normally runs continuously, processing logs whenever they are passed on from the primary system. You can start it with this:

```sql
dba> export USEREXIT_DIRECTORY=/big/dba/userexit
dba> sync_db testdb &
```

If you edit the script to set the default directory settings, you can skip setting the environment variable `USEREXIT_DIRECTORY`. If you use a different directory layout within the directory area, you can set environment variables `USEREXIT_ARCHIVE`, `USEREXIT_RETRIEVE`, and `USEREXIT_APPLIED` to specify your desired usage.

### Using a Standby System

There are a number of reasons for running an active standby system, and your particular needs have an important effect on how you use it. So this document only gives some general ideas about using an active standby system.

If there has been a data-entry error on your primary system and if your control over the log processing has ensured that the secondary has not yet processed the transaction that caused the error, you can use the copy on the secondary system to examine the problem. Most likely, your first step will be to stop further log processing:

```sql
dba> touch \\
    /big/dba/userexit/archive/TESTDB/NODE0000/stop_sync
```

This command causes the `sync_db` demon to terminate after it finishes the current round of log processing (or immediately if there isn’t any log being processed). After that, you might manually roll forward through the logs to get to the point of the error and then take the database out of rollforward pending state. Alternatively, your application might maintain its own high-level logs that are easier to process. To take the database out of rollforward pending state:

```sql
dba> db2 rollforward database testdb complete
```

After finding the problem, you would try to repair it, and then test to ensure that the repair was successful both in fixing the original problem without creating a new problem. Next, you would repair the primary database. Finally, you would have to ensure that the secondary system was again synchronized with the primary one – and that changes made on the secondary system were properly undone if they were not
duplicated on the primary one. Establishing the data volumes to update the BCV copy again might be one way to do this.

**Performed on the Secondary System**

```
# Stop the database application, and then stop db2
# …
dba> db2stop

root> umount /data
root> symmir -g data -noprompt establish
root> symmir -g data -I 30 verify
root> symmir -g data -noprompt -instant split
root> mount /data

dba> db2start
# Continue with normal operations, perhaps restarting
# the sync_db demon.
dba> sync_db testdb &
```

Now your secondary system is back in an automated rollforward state.
Scenario 5  Switching between Primary and Standby Systems with EMC TimeFinder

An even more critical issue may arise from the desire to have a database application running at all times. To deal with times when the computer system is unavailable, a second system ready to take over running the application is useful. There are situations in which the primary system can be unavailable. An unexpected event, such as a power failure or a crash of the hardware or the software on the system, can cause the system to stop operating without any notice. On other occasions, you may have some warning that the system will need to be shut down for a while for such things as hardware upgrades, maintenance, repair, system reconfiguration, etc.

To handle unexpected outages, the best solution permanently establishes a BCV copy of the entire database, ready to be switched over to the standby system. This copy can be used for scheduled switchovers, too. A permanently established BCV means that you always have the latest changes to the database immediately available.

Switching to a standby system is similar to making a snapshot of the database. The main difference is that the copy of the database being used on the secondary system becomes the official production instance of the database, and the original becomes considered a copy. The original will have to be brought up to date with the changes made in the standby copy before it can be reinstated as the official production instance. The standby copy must retain the ability to roll logs forward and backward, and it must be copied back to the primary system when the application is restarted there. For some applications, additional changes are required during the switch to inform users of the application about the switch. This might involve having the secondary system take over the network name of the primary system, for example. The other difference is that when the copy is taken, the application is not left running on the primary system – the application is either taken down deliberately for a planned switchover or taken down by a disaster.

Planned Switchover to the Standby Secondary System

When the switchover to the standby system is a planned one, you first cleanly shut down the database on the primary system.

Performed on the Primary System

Stop database applications (site dependent) and shutdown the database:

```
dba> db2stop force
```

Split each BCV group (no database suspend is required because the database is shutdown):

```
root> symmir -g data -noprompt -instant split
root> symmir -g logs -noprompt -instant split
```

Unmount the file systems:

```
root> umount /data
root> umount /logs
```

Performed on the Secondary System

Mount the file systems:

```
root> mount /data
root> mount /logs
```

Start the database:
dba> db2start (if required)

dba> db2 connect to testdb

Other Tasks Required for Your Application
Other actions may need to be performed to make the database application available to its users. These actions might involve updating network addresses so that the users will be redirected to the secondary system instead of the primary system.

Disaster Switchover to the Standby Secondary System
If the primary system has crashed, you cannot depend on being able to do any tasks with it. In particular, you cannot shut down the database. This procedure compensates for the fact that the database was not shut down cleanly. Fortunately, you can carry out the entire procedure from the secondary system.

Performed on the Secondary System
Split the BCV volumes:

root> symmir -g data -noprompt -instant split

root> symmir -g logs -noprompt -instant split

Mount the file systems:

    mount /data
    mount /logs

Start DB2 and connect to the database:

    dba> db2start (if required)

If database not enabled for AUTORESTART:

    dba> db2 restart database testdb

Connect to database to ensure available:

    dba> db2 connect to testdb

Switching Back to the Primary System
Once the primary system is again capable of running the database application, you may want to switch back to using it. This process is similar to a planned switchover done in reverse.

Performed on the Secondary System
Stop database applications (site dependent) and shutdown the database:

    dba> db2stop force

Unmount the file systems:

    root> umount /data
    root> umount /logs

Performed on the Primary System
The database should not be running and the file systems should be unmounted.
Restore the primary volumes from the BCV volumes that the secondary system has been using:

```bash
root> symmir -sid 343 -f /tmp/devlist restore -noprompt
```

Mount the file systems:

```bash
root> mount /data
root> mount /logs
```

Start DB2 and connect to database:

```bash
dba> db2start
dba> db2 connect to testdb
```
Appendix A  

db_split Script

This script can be used to split BCV copies of volumes that contain a database. You can download this script by anonymous ftp from [ftp://ftp.emc.com/pub/symm/DB2/db_split_latest](ftp://ftp.emc.com/pub/symm/DB2/db_split_latest) (New versions will be placed in the same directory with date extensions, the most recent will always have the _latest extension as well). Whenever a line in this script ends with a trailing backslash (\), means that a line break was required for printing. You can join up such lines and remove the backslashes. The copy of the script available by FTP has the original unbroken lines.

```bash
#!/bin/ksh

# db_split [-g] [-t] [-l [pre|post]] [-s SID] dbname dbid group ...
# db_split [-f] [-t] [-l [pre|post]] [-s SID] dbname dbid file ...
#
# split one or more groups or device lists that are being used by a database
#
# dbname database name to connect to
# dbid account to run db2 commands
# group Symmetrix groups to be split
# file file containing list of devices to be split
#
# -f list of files provided (instead of groups)
# -g list of groups provided (instead of files)
# (if neither -f nor -g is specified, the script will check whether the provided args are all existing files and use -f if they are, or -g if any is not)
# -t issue time stamps for each operation
# -l pre switch logs before the split
# -l post switch logs after the split
# -s Symmetrix ID (either this arg or env $SYMCL_SID is required when device list file(s) are used)

PATH=$PATH:/opt/emc/SYMCLI/V4.2/bin

```
Creating Hot Snapshots and Standby Databases

```bash
typeset timing=/bin/false
typeset list=guess
typeset sid=
typeset preswitch=
typeset postswitch=

while [[ $1 = -* ]];
do
  typeset arg=${1#-}
  shift
  while [ -n "$arg" ];
do
    case "$arg" in
      (t*) timing=/bin/true
        ;;
      (f*) list=file
        ;;
      (g*) list=group
        ;;
      (l*) if [ "$arg" = "l?" ]
        then
          switchtime=${arg#l}
          arg=l
        else
          if [ -z "$1" ]
            then
              echo "pre or post required as arg for -l" >&2
              echo $usage >&2
              exit 1
            fi
          switchtime="$1"
          shift
        fi
      if [ "$switchtime" = pre ]
        then
          preswitch=yes
        elif [ "$switchtime" = post ]
        then
          postswitch=yes
        else
          echo "pre or post required as arg for -l" >&2
          echo $usage >&2
          exit 1
        fi
      ;;
      (s*) if [ "$arg" = "s?" ]
        then
          sid=${arg#s}
          arg=s
        else
          if [ -z "$1" ]
            then
```
echo "Symm. id required for -s" >&2
echo $usage >&2
exit 1
fi
sid="$1"
shift
fi
;

(*) echo "unrecognized option $arg" >&2
echo $usage >&2
exit 1
;
esac
arg=${arg#?}
done
done

if [ -z "$1" ]
then
echo "database name required" >&2
echo $usage >&2
exit 1
fi
typeset connect=$1
shift

if [ -z "$1" ]
then
echo "database id required" >&2
echo $usage >&2
exit 1
fi
typeset dbid=$1
shift

if [ $# = 0 ]
then
echo "one or more argument required: device list file(s) " \
"or Symmetrix group(s)" >&2
echo $usage >&2
exit 1
fi

if [ $list = guess ]
then
list=file
for i in *
do
  [ -f "$i" ] || list=group
done
fi
if [ $list = file ]
 then
  if [ -z "$sid" ]
  then
    if [ -z "$SYMCL_SID" ]
    then
      echo "-s SID or env SYMCL_SID required for device " \ 
      "list file" >&2
      echo $usage >&2
      exit 1
    fi
    else
      sid="-sid $sid"
    fi
  else
    listfile=/tmp/dbs_${_dev}
    if cat $* > $listfile
    then
      :
    else
      echo "error copying file argument(s) aborting"
      exit 1
    fi
  else
    listfile=$1
  fi
fi

pre=/tmp/dbs_${_pre}
post=/tmp/dbs_${_post}
grave='`'
rm -f $pre $post

(^
cat <<EOF
!/bin/ksh
exec >/tmp/pre.out 2>/tmp/pre.out
if [ -n "$preswitch" ]
 then
   su - $dbid -c "db2 archive log for database " \ 
   "$connect" >/tmp/switch.out
   stat=\$?
   if [[ \$stat -ne 0 ]]
   then
      echo "db2 log switch failed: status: \$stat"
      $diag /tmp/switch.out
      exit 1
   fi
   rm /tmp/switch.out
 fi

 su - $dbid -c "db2 connect to $connect" > /tmp/su.db.out
 stat=\$?
 if [[ \$stat -ne 0 ]]
 then

EOF)
#!/bin/ksh
exec >/tmp/post.out 2>/tmp/post.out
su - $dbid -c "db2 connect to $connect" > /tmp/su.db.out
stat=$?
if [[ $stat -ne 0 ]]
then
    echo "db2 connect to $connect failed: status: $stat"
    $diag /tmp/su.db.out
    exit 1
fi

$timing && echo ${grave}date${grave} resuming database \ writes >>/tmp/timing.out
su - $dbid -c "db2 set write resume for database" \ > /tmp/su.db.out
stat=$?
if [[ $stat -ne 0 ]]
then
    echo "db2 write resume failed: status: $stat"
    $diag /tmp/su.db.out
    exit 1
fi

su - $dbid -c "db2 terminate" > /tmp/su.db.out
stat=$?
if [[ $stat -ne 0 ]]
then
    echo "db2 terminate failed: status: $stat"
    $diag /tmp/su.db.out
    exit 1
fi

if [ -n "$postswitch" ]
then
    echo "db2 connect to $connect failed: status: $stat"
    $diag /tmp/su.db.out
    exit 1
fi

if [[ $stat -ne 0 ]]
then
    echo "db2 write suspend failed: status: $stat"
    $diag /tmp/su.db.out
    exit 1
fi
exit 0

EOF
) >$pre
chmod +x $pre

( cat <<<EOF

EOF

# EMC Corporation and IBM Corporation
DATE: 08/17/01

Creating Hot Snapshots and Standby Databases  36
su - $dbid -c "db2 archive log for database $connect" \ 
   >/tmp/switch.out
stat=$? 
if [[ \$stat -ne 0 ]]
then 
echo "db2 log switch failed: status: \$stat"
   $diag /tmp/switch.out
   exit 1
fi 
   rm /tmp/switch.out
fi 
exit 0
EOF
) >$post
chmod +x $post

rm -f /tmp/symmir.out /tmp/pre.out /tmp/post.out

if [ $list = group ]
then
    next_pre="-preaction $pre"
    next_post=
    while $#
        do
            group=$1
            shift
            [ $# -eq 0 ] && next_post="-postaction $post"
            $timing && echo 'date' splitting group $group
            symmir -g $group -noprompt split -instant $next_pre $next_post \ 
                > /tmp/symmir.out 2>&1
            stat=$?
            if [ -s /tmp/timing.out ]
                then
                    cat /tmp/timing.out
                    rm /tmp/timing.out
                fi 
            if [[ \$stat -ne 0 ]]
                then
                    echo "split of group $group failed: status: \$stat"
                    $diag /tmp/symmir.out /tmp/pre.out /tmp/post.out
                    exit 1
                fi
            pre=
            rm -f /tmp/symmir.out /tmp/pre.out /tmp/post.out
        done
else
    $timing && echo 'date' splitting listed devices
    symmir $sid -f $listfile -noprompt split -instant \ 
        -preaction $pre -postaction $post > /tmp/symmir.out 2>&1
    stat=$?
    if [ -s /tmp/timing.out ]
        then
            cat /tmp/timing.out
            rm /tmp/timing.out
        fi
    
```
if [[ $stat -ne 0 ]]
then
  echo "split of devices failed: status: $stat"
  $diag /tmp/symmir.out /tmp/pre.out /tmp/post.out
  exit 1
fi
rm -f /tmp/symmir.out /tmp/pre.out /tmp/post.out /tmp/dbs_$_dev
fi
rm -f $pre $post
rm /tmp/su.db.out
$timing && echo `date` db_split complete
Appendix B  sync_db Script

This script loops until you create a stop file. It watches for logs posted by the primary system in order to roll them forward on the secondary system. You can modify this script to suit your needs, such as delaying the time until a log is processed. You may download this script by anonymous ftp from ftp://ftp.emc.com/pub/symm/DB2/sync_db_latest (New versions will be placed in the same directory with date extensions, the most recent will always have the _latest extension as well).

#!/bin/perl
#
# 1) Name of file : sync_db
#
# 2) Purpose : This script will check for existence of log files which need to be applied to the backup database and apply them.
#
# 3) History :
#    02/16/96 Enzo Cialini - Creation to show functional specs
#    03/29/01 Yong Kim -
#    04/16/01 Yong Kim - Convert OS/2 Rexx script to perl
#    05/09/01 John Macdonald - test dirs, node argument
#      - daemon/one-time option, quick sleep
#      no file was pending and no change to dir
#      - env vars to override default dirs
#
# 4) Example Scenarios
#
# Given that we are "dba", are working with a database name "SAMPLE" on primary System
#
# $USEREXIT_DIRECTORY="/db1/dba/userexit/";
# $USEREXIT_ARCHIVE="/db1/dba/userexit/archive/";
# $USEREXIT_RETRIEVE="/db1/dba/userexit/retrieve/";
# $USEREXIT_APPLIED="/db1/dba/userexit/applied/";
#
# Userexit on primary machine archives logs to $USEREXIT_ARCHIVE directory. Sync_db queries the $USEREXIT_ARCHIVE directory and if any log exists, move them to $USEREXIT_RETRIEVE directory for processing.
#
# Once the logs has been processed (i.e. Rollforward completed). They are moved to the $USEREXIT_APPLIED directory.
#
# 5) Usage:
#    sync_db [options] <db_name>
#
# where db_name = Database name to roll forward
# Options:
#   -1 only process one set of logs instead of loop forever
#   -n node (for EEE) use a node other than NODE0000
#
# Environment variables can override default directories:
# $USEREXIT_DIRECTORY = $ENV{USEREXIT_DIRECTORY} | "/db1/dba/userexit";
# $USEREXIT_ARCHIVE = $ENV{USEREXIT_ARCHIVE} | ""
Creating Hot Snapshots and Standby Databases

```bash
"$USEREXIT_DIRECTORY/archive";
$USEREXIT_RETRIEVE = $ENV{USEREXIT_RETRIEVE} ||
"$USEREXIT_DIRECTORY/retrieve";
$USEREXIT_APPLIED = $ENV{USEREXIT_APPLIED} ||
"$USEREXIT_DIRECTORY/applied";

$usage= <<EOF;
USAGE : sync_db [-n node] [-1] <db_name>
   where db_name is database name to roll forward
   -n node specifies node (default NODE0000)
   -1 runs once (default is to loop)
EOF

$node = 'NODE0000';
$demon = 1;

while( @ARGV && $ARGV[0] =~ s/^-([n1r])// ) {
  if( $1 eq '1' ) {
    $demon = '';
  } else {
    # all others take a value, allow it to be either 
    # part of the same word: -xFoo
    unless ($value = $ARGV[0]) {
      # or as a separate word: -x FOO
      $value = splice( @ARGV, 1, 1 );
    }
    $node = $value if $1 eq 'n';
  }
  $ARGV[0] eq '' ? shift : ($ARGV[0] = "-$ARGV[0]");
}

if( $ARGV[0] =~ /^-/ ) {
  print ("\n");
  print ("Error : Unknown option: $ARGV[0].\n\n");
  die ($usage);
}

unless (@ARGV == 1) {
  print ("\n");
  print ("Error : A Database Name must be provided.\n\n");
  die ($usage);
}

$database=$ARGV[0];
$database =~ tr/a-z/A-Z/;
$arch = "$USEREXIT_ARCHIVE/$database/$node";
$retr = "$USEREXIT_RETRIEVE/$database/$node";
$appd = "$USEREXIT_APPLIED/$database/$node";

#
# check directories (archive, retrieve, applied)
foreach $dir ( $arch, $retr, $appd ) {
  die "Directory $dir does not exist"
  unless -d $dir;
  die "Directory $dir not writeable"
  unless -w _;
```
# remove leftover stop file
unlink "$arch/stop_sync" if -f "$arch/stop_sync";

# open AUDIT LOG
#

open (AUDIT, ">>$USEREXIT_DIRECTORY/$database.LOG");
print AUDIT ("n******************************************************************

printf AUDIT ("Initial Execution : %s
", &get_time);
print AUDIT ("Database : $database

print AUDIT ("archive directory : $USEREXIT_ARCHIVE

print AUDIT ("retrieve directory : $USEREXIT_RETRIEVE

print AUDIT ("applied directory : $USEREXIT_APPLIED

#
# check to see if the log file exists. If not, then no logs has been archived
#

print ("Looking for log files : ");
print AUDIT ("Looking for log files : ");

$timestamp = 0;
$activelog = 0;

while( 1 )
{
    # quick sleep if there wasn't a log file still changing
    # last round, and the arch directory hasn't been touched
    # (thus no new file has been added)
    unless( $activelog ) {
        while( 1 ) {
            $mtime = (stat($arch))[9];
            last if $mtime != $timestamp;
            sleep 10;
        }
        $timestamp = $mtime;
    }
    opendir (DIR, $arch) ||
        die ("n Error : Cannot open directory ($arch) \n\n");
    $now = time;
    @logfiles = ( );

    foreach $logfile (readdir (DIR))
    {
        if( $logfile eq 'stop_sync' ) {
            print ("Stopping by request.\n");
            print AUDIT ("Stopping by request.\n");
            exit 0;
        }
        next unless ($logfile =~ /S\d\d\d\d\d\d.LOG/);
        next unless ($now - (stat($logfile))[9]) > 10;
push (@logfiles, $logfile);
}

$num_logs=@logfiles;

unless (@logfiles) {
    if ( $demon ) {
        sleep 10;
        redo;
    }
    print ("\n\n");
    print AUDIT ("Not found\n");
    close (AUDIT);
    die ("Error : Log files not found. \n\n");
}
else {
    print("log Found!!\n");
    print AUDIT ("log Found\n");

    print ("\n$num_logs file(s) to rollforward [\$logfiles[0] -",  
        " \$logfiles[\$num_logs-1]]\n\n");
    print AUDIT ("\n$num_logs file(s) to rollforward [\$logfiles[0] -",  
        " \$logfiles[\$num_logs-1]]\n\n");

    foreach $logfile (@logfiles) {
        $rc = system("mv $USEREXIT_ARCHIVE/$database/$node/$logfile" .  
            " $USEREXIT_RETRIEVE/$database/$node");
    }

    print ("Rolling forward ...\n");
    print AUDIT ("Rolling forward ...\n\n");
    printf AUDIT ("START : %s ",&get_time);

    $rc = system("db2 rollforward db $database to end of logs");
    printf AUDIT ("STOP : %s RC: %d",&get_time, $rc);

    $rc = system("cp -r $USEREXIT_RETRIEVE/$database/$node/S*.LOG" .  
        " $USEREXIT_APPLIED/$database/$node");

    if ($rc == 0) {
        $rc = system("rm -rf $USEREXIT_RETRIEVE/$database/$node/S*.LOG");
    }
    print ("\n");
    last unless $demon;
}

sub get_time {
    ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) = localtime(time);
    $cyear=$year+1900;
    ++$mon;

    $retval= "$cyear/$mon/$mday $hour:$min:$sec";
}

Creating Hot Snapshots and Standby Databases 42
Appendix C  Alternative Scripts for AIX Device Lists

The following code creates the same lists that we set up in the section **Configure AIX Device Variables**, but this way of doing so is faster to type (especially using a shell that permits you to edit the previous command to change the limit and the two places that use the list name being created).

```bash
curid=7
while [ $curid -le 10 ]
do
datadk="$datadk hdisk$curid"
  (( curid = $curid + 1 ))
done
while [ $curid -le 26 ]
do
  rawdk="$rawdk hdisk$curid"
  (( curid = $curid + 1 ))
done
while [ $curid -le 30 ]
do
  logsdk="$logsdk hdisk$curid"
  (( curid = $curid + 1 ))
done

And here is third approach for people who do this frequently. It can be easily used for large numbers of lists, or for non-contiguous sequences of numbers:

```bash
curid=7
while read var limit
do
typeset list=''
  while [ $curid -le $limit ]
do
    list="$list hdisk$curid"
    (( curid = $curid + 1 ))
done
  eval "$var=""$var $list"
  ""
done <<var_lastdisk_list_end
datadk 10
rawdk 26
logsdk 30
var_lastdisk_list_end
```

If the devices are not contiguous, this last approach works well. Just use a list name of "junk" to skip over the device numbers that are not used, and repeat a list if it has non-contiguous devices.

For example, if the data disk set used hdisk7-hdisk10 as above, but also included hdisk40-hdisk43, you would add the following two lines immediately before the `var_lastdisk_list_end`:

```bash
junk 39
datadk 43
```
and the list variable $datadk would end up with the value:

```
hdisk7 hdisk8 hdisk9 hdisk10 hdisk40 hdisk41 hdisk42 hdisk43
```
while the other arrays would have the same values as before.