
Using SYMCLI to Perform Storage Resource Management

Abstract

This white paper introduces the Solutions Enabler Storage Resource Management (SRM) component, which provides mapping and control commands that allow your application to examine and manage objects in the storage environment.

Published 1/25/2005

Copyright © 2005 EMC Corporation. All rights reserved.

EMC believes 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”. EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

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

EMC², EMC, Symmetrix, Celerra, CLARiiON, CLARAlert, Documentum, HighRoad, Legato, Navisphere, PowerPath, ResourcePak, SnapView/IP, SRDF, TimeFinder, VisualSAN, and where information lives are registered trademarks and EMC Automated Networked Storage, EMC ControlCenter, EMC Developers Program, EMC OnCourse, EMC Proven, EMC Snap, Access Logix, AutoAdvice, Automated Resource Manager, AutoSwap, AVALONidm, C-Clip, Celerra Replicator, Centera, CentraStar, CLARevent, Connectrix, CopyCross, CopyPoint, DatabaseXtender, Direct Matrix, Direct Matrix Architecture, EDM, E-Lab, Enginuity, FarPoint, FLARE, GeoSpan, InfoMover, MirrorView, NetWin, OnAlert, OpenScale, Powerlink, PowerVolume, RepliCare, SafeLine, SAN Architect, SAN Copy, SAN Manager, SDMS, SnapSure, SnapView, StorageScope, SupportMate, SymmAPI, SymmEnabler, Symmetrix DMX, Universal Data Tone, and VisualSRM are trademarks of EMC Corporation. All other trademarks used herein are the property of their respective owners.

Part Number 300-000-507 REV F

Table of Contents

Introduction	5
Purpose and Scope	5
Related Documentation	5
Practical Uses	5
Mapping Concepts.....	7
Host System Mapping to a Relational Database	8
Connecting to a Database	9
Examining a Database.....	10
Using Daemons for Database Mapping.....	12
File System Mapping and File Mapping	13
Logical Volume Mapping.....	14
Mapping Logical Objects to Symmetrix Physical Devices	16
Displaying Partition Information about a Physical Device	18
Performing Database I/O Control Operations.....	19
Using Daemons for Database I/O Control Operations.....	19
Suspending Writes to the Database	19
Freezing an Oracle Database	19
Freezing an Informix Database	19
Freezing a Sybase Database	20
Freezing an IBM UDB Database.....	20
Freezing an SQL 2000 Database.....	20
Starting Up and Shutting Down a Database Server Manager	20
Performing Control Operations on a Volume Group	21
Deporting and Importing a Volume Group	21
Creating a Volume Group	21
Performing Control Operations on Logical Volumes	22
Collapsing a File's Logical or Physical Extents	22
Extent Transformation in the Mapping Layers	24

Examples: Symmetrix Storage Devices	25
Example 1: Displaying Relational Database Objects	25
Examining Oracle Database Objects	25
Examining IBM DB2/UDB Database Objects	28
Examining Informix Database Objects	29
Examining Microsoft Exchange Database Objects	31
Example 2: Mapping Files and Other Disk Storage Objects	32
Example 3: Displaying Volume Groups and Logical Volumes	45
Example 4: Deporting and Importing a Volume Group	50
 Examples: CLARiiON Storage Devices	 52
Example 5: Mapping Files	52
Example 6: Displaying a Logical Volume	55

Introduction

The EMC® Solutions Enabler Storage Resource Management (SRM) component provides mapping and control commands that allow your application to examine and manage objects located:

- On your host
- Within your relational database, file systems, and logical volume managers
- In a storage environment that stores its data on Symmetrix® or CLARiiON® systems

SRM mapping commands permit host applications to identify and obtain information about the total system storage environment, from database composition down to the physical storage on disks.

SRM control commands allow applications to manage this storage environment via common volume group and logical volume control operations across all supported logical volume managers (LVMs). You can also perform various control operations on supported database systems. SRM's capabilities are particularly useful in mixed storage environments because they provide a consistent interface across all supported operating system platforms, file systems, LVMs, and databases.

The SRM component currently supports the following storage resources and environments:

- Relational databases
- File systems and files within file systems
- Logical volumes and volume groups

For currently supported versions of operating systems, volume managers, file systems, and database types, refer to the table "Supported Platforms and Environments" in the *EMC Solutions Enabler Symmetrix SRM CLI Product Guide*.

Purpose and Scope

This paper provides an introduction to the SRM functionality included in EMC Solutions Enabler up through version 6.0. This white paper focuses primarily on the use of SRM commands to examine mapping information and perform control operations.

Related Documentation

The following EMC manual provides information related to this paper:

- *EMC Solutions Enabler Symmetrix SRM CLI Product Guide*

Practical Uses

As stated in the introduction, SRM's management capabilities are particularly useful in mixed operating systems and/or storage environments because they provide a consistent interface across all supported operating system platforms, file systems, LVMs, and databases.

SRM allows you to locate specific data files within a file system and restore a file should the need arise. Being able to locate specific data files precludes having to back up an entire file system. If the file system resides on a standard device that is paired with a BCV, then freezing I/O to the file system for the fraction of a second required to split the BCV means that the file system I/O is resumed again almost immediately. A backup server can then import the volume group and mount the file system to use the BCV for backup or restore.

SRM can enable backups between different operating systems. Figure 1 shows an AIX backup application that needs to back up a Sun file system disk but is unable to decipher the Sun file system structure. An agent on the Sun host uses SRM file system mapping to retrieve file extent meta data, which the Sun host sends to the AIX backup application so that it can access the Sun disk and perform the high-speed backup.

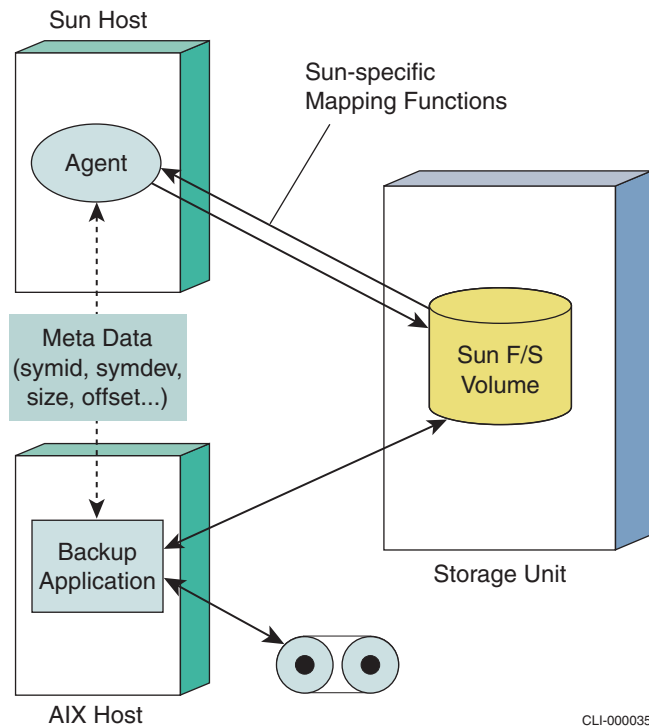


Figure 1. SRM File System Mapping

SRM also allows you to determine how storage devices are used on a given host and can help you identify devices that are not used by any file system or database.

Some performance tools use SRM logical-to-physical mapping. For example, one performance tool maps “hot” pages in an Oracle data file to a Symmetrix physical disk. SRM can also be used for physical-to-logical mapping, for example, to map a SCSI CDB trace back to files used by Oracle tables.

An EMC application like Double Checksum for Oracle uses SRM to locate extents for an Oracle data file that requires track protection (protecting data blocks on tracks containing a file’s extents). A DBMS query optimizer uses mapping of data files to storage disks to gain better performance.

SRM also allows you to suspend I/O to a database momentarily to perform some short operation such as initiating an instant split or activating a snap. With the `symioctl` command, SRM can place a database in backup mode (for Oracle) or momentarily freeze a database (such as Informix, SQL Server, etc.), suspending any database updates until the operation completes and another `symioctl` command resumes access to the database.

Beginning with Solutions Enabler version 5.3, SRM allows applications to control volume groups and logical volumes, provides LVM statistics, and allows you to use a variety of options when starting up or shutting down a database server manager.

Mapping Concepts

A storage system such as a Symmetrix or CLARiiON unit applies a high degree of virtualization between what the host sees and the actual physical disk drives. A logical device with a name that the host can address is mapped to a physical disk. From the perspective of the host, these logical devices appear as physical devices connected to one or more I/O controllers.

The data management software (database, logical volume, file system) allocates space for the logical device. This space is allocated as one or more extents, an *extent* representing a contiguous string of data blocks, and a *block* representing the smallest unit of data that can be read or written.

A common source of confusion is the use of the terms *logical extents* and *physical extents*. In many cases, each mapping layer refers to the extents at its level as logical, and the extents at the next lower layer as physical. This viewpoint makes sense when there are only two layers. However, when there are 13 layers, there is no correct answer to the question, “Where are the physical extents for my file?” This paper uses *physical extents* to mean an object’s physical-device (pdev) extents.

A *physical device* is a name by which the host system recognizes a storage device. For example, a physical device name (such as `/dev/rdisk/c5t4d3s2`) can refer to a Symmetrix device that the Symmetrix storage system has configured either to an entire physical disk, to part of a disk that has been divided into partitions, or to multiple disks (for a Symmetrix mirrored device or meta device).

To maintain multiple copies of the data, a logical device usually maps to at least two mirrors on the storage unit (a.k.a., the storage array). However, the mapping commands do not map *directly* to the physical disks. Generally, the mapping commands that return extent information go down to the physical device level (and to the Symmetrix or CLARiiON device level for those storage units). Figure 2 shows a logical device mapped to a physical device that has been configured at the storage level to be a 2-way mirror. This mirror protection means that the device extents exist on different disks in the storage unit, but data management software on the host has no knowledge of storage system mirrors (M1 and M2).

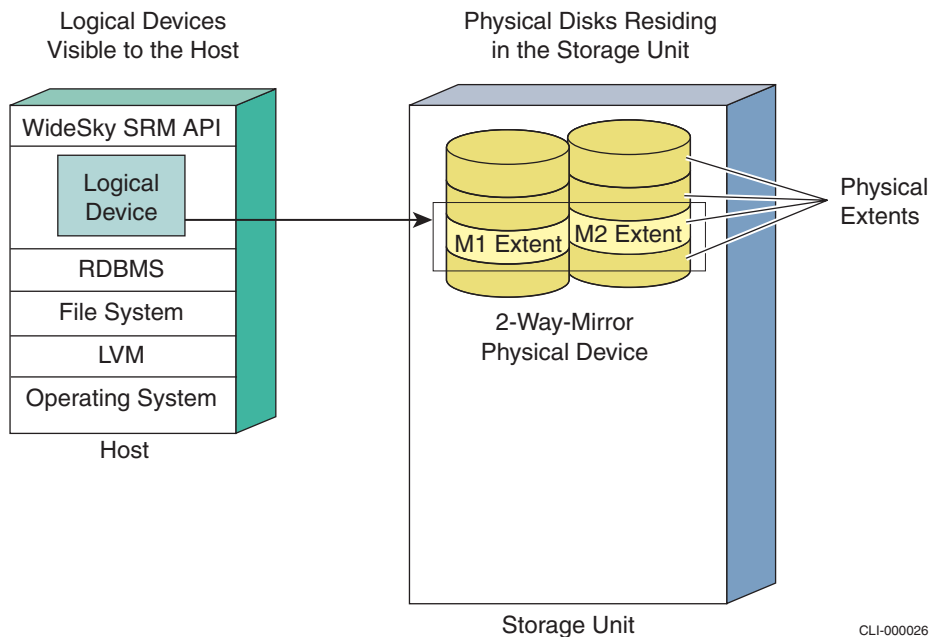


Figure 2. Logical Device Mapped to a Physical Device

The logical device represents some data object. Applications commonly access pieces of data within a data object by opening the object's name, positioning to some location, and then reading or writing some number of bytes. A data object can be as simple as a stream of bytes in a "raw" device or a flat file, or as complex as a relational database table or an OS/390 VSAM/KSDS dataset with multiple indexes and transaction logs. A data object typically has a name (such as `\\.\L:`), a size (such as 10 GB), and one or more mirrors. Each data object mirror has one or more extents.

When data is physically stored on disks, it is not desirable (or usually even possible) for host applications to access physical disks directly. As such, one or more levels of data management software supply an interface to access their data objects, while hiding the actual location of the data on physical disks.

In fact, each data management level often creates several layers of data within its own level. For example, the VERITAS volume manager can create a mirrored logical volume where each mirror is striped and each subdisk within each stripe column is itself mirrored. All of this, of course, is hidden from the file system mounted on top of the logical volume.

Each mapping layer has its own objects, names, mirrors, extents, and block sizes. Each layer uses "maps" to navigate from its own extents to the extents at the next lower layer. For example, it is not unusual to find a stack of 13 or more layers between an Oracle table and a physical disk. Each layer "knows" how to access the layer below it, but has no knowledge about any other layers (higher or lower or peer).

A logical volume manager knows that a file system request to write eight kilobytes of data to offset 50 KB in logical volume `/dev/vx/rdisk/myvg/mylvol` should write the first four kilobytes to offset 124 KB in physical device `/dev/rdsk/c5t4d3s2`, and the last four kilobytes to offset 112 KB in physical device `/dev/rdsk/c1t2d3s3` — but that is all. The logical volume manager knows nothing about Oracle tables or files or Symmetrix striped meta devices.

To build this stack of layers, it is necessary to create the layers from the bottom up. For example, the host devices cannot be created until the Symmetrix or CLARiiON unit is connected. Oracle tablespaces cannot be created until the file systems are created.

Keep in mind that data management software on the host has no knowledge of storage system mirrors (for example, M1, M2, M3, and M4 in Symmetrix/CLARiiON units). However, each layer on the host has the option to mirror some or all of its extents on the layer just below it. At least one layer (the Symmetrix or CLARiiON device layer) configures mirrors for a Symmetrix or CLARiiON device, but there may be two or three other layers that create mirrors. Because mirrors can occur at multiple layers, a single write from Oracle could result in eight writes to physical disks, or even more if TimeFinder® or SRDF® software is being used.

The mirror configurations displayed in the examples sections represent the mirror view at the operating system level (top layer), and not at the storage system level or any of the intermediate levels.

Host System Mapping to a Relational Database

SRM mapping commands examine characteristics of relational database structures, files, tablespaces, and user schemas, and determine host-system mapping to the physical devices associated with the database. An RDBMS allocates space (extents) for the management of a particular database file. Figure 3 on the next page shows two host database files, each mapped to a physical device on the storage unit. At the storage level, each physical device consists of two extents that contain the data of the database file.

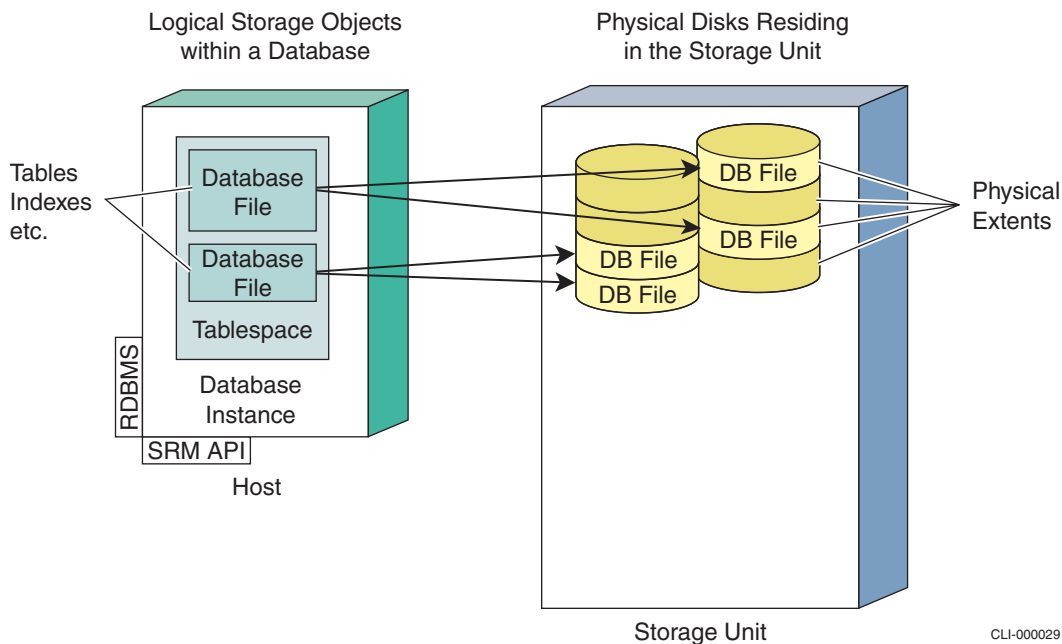


Figure 3. Database Files Mapped to Extents on Two Disks

A *tablespace* is a named storage pool that physically allocates space for the database files. As the illustration shows, one or more table and index structures make up the database files of a tablespace. The data is stored logically in tablespaces and physically on disks that are associated with the database files through the mapped physical devices.

Although the figure shows a single database instance on the host, some RDBMS products allow you to create multiple databases that execute concurrently on the same host, each database accessing its own physical database (disks on the storage unit). Regardless of RDBMS terminology (instance, database, or server), all database instances contain data files, shared memory, and processes.

Connecting to a Database

For SYMCLI to access a database, you need to set the `SYMCLI_RDB_CONNECT` environment variable. The following example sets the `SYMCLI_RDB_CONNECT` variable to a username of “system” and a password of “manager.” (The Bourne and Korn shells use the `export` command to set environment variables; the C shell uses the `setenv` command.)

```
export SYMCLI_RDB_CONNECT=system/manager
```

Before beginning any SYMCLI database command session (`symrdb`), set the appropriate database application variables on the host where the database instance (or server) is running (refer to Table 1). For example, setting `ORACLE_HOME` specifies the location of the Oracle binaries. Setting `ORACLE_SID` specifies the database instance name:

```
export ORACLE_HOME=/disks/symapidvt/oraclehome/api179
export ORACLE_SID=api179
```

Table 1. Defining Environment Variables for Database Applications

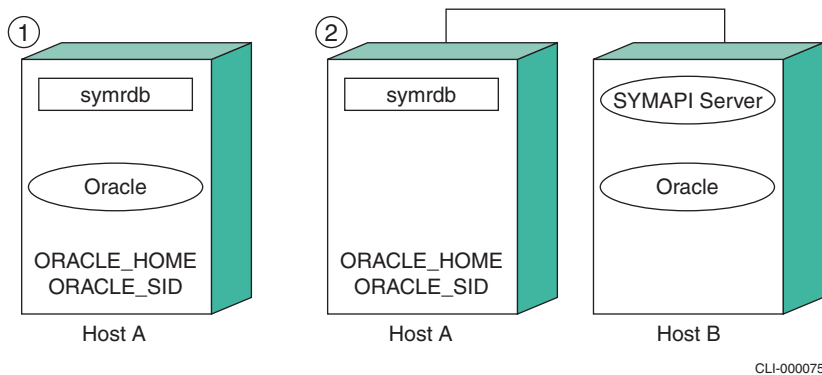
For Database	Set Variables
Oracle	ORACLE_HOME and ORACLE_SID.
Informix Dynamic Server	INFORMIXDIR, ONCONFIG, and INFORMIXSERVER
Sybase	SYBASE and DSQUERY. If the OCS-12_0 directory is not located in the \$SYBASE directory, set LD_LIBRARY_PATH, LIBPATH, or SHLIB_PATH, depending on whether you are running SunOS, AIX, or HP-UX.
IBM DB2/UDB	DB2INSTANCE. Set LD_LIBRARY_PATH, LIBPATH, or SHLIB_PATH, depending on whether you are running SunOS, AIX, or HP-UX.
SQL Server; Exchange	None required.

SQL Server and Exchange do not have variables that specify a server instance. Beginning with Solutions Enabler version 5.1, SYMCLI supports two or more named SQL Server instances running on the same Windows 2000 box. To connect to an SQL server instance, add a service name (an ODBC system DSN) to the RDB_CONNECT variable:

```
set SYMCLI_RDB_CONNECT=username/password@service
```

To create this service name (ODBC system DSN), click on Control Panel, Administrative Tools, Data Sources (ODBC), and the System DSN tab. Create an ODBC system DSN. When prompted to choose a data source from a list of sources, choose "SQL Server." When prompted for the named instance of SQL Server, enter the instance name.

Figure 4 shows two configurations from which to run a SYMCLI database command session (symrdb). In configuration 1, SRM and Oracle are both on the Host A. In configuration 2, SRM on Host A communicates with the SYMAPI server on Host B, where Oracle is located. The database variables defined on Host A are propagated to the Host B server environment at run time.

**Figure 4. Propagating Environment Variables to the SYMAPI Server Environment**

Examining a Database

Various `symrdb` commands examine the mapping of database objects. The `symrdb list` command lists various physical and logical database objects:

- Databases
- Tablespaces, tables, files, or schemas of a database
- Files or tables of a database tablespace or schema

The `symrdb show` command displays information about a specific database object:

- Tablespace, table, file, or schema
- File or table of a database tablespace or schema

The following commands list available databases within the database instance or server that you set using the appropriate database application variable. Most database types (like Informix) allow you to create multiple databases within a database instance; however, Oracle does not. Consequently, this command using `-type oracle` simply lists the current Oracle database instance.

```
symrdb list -type oracle
symrdb list -type informix
```

Other supported database types are `sqlserver`, `exchange`, `sybase`, and `ibmudb`.

The following commands list all files for the current Oracle database instance, and all files of a database (sysutils) created within the context of the current Informix database server (instance).¹

```
symrdb list -type oracle file
symrdb list -type informix file -db sysutils
```

For the `file` argument, you can substitute `tbs`, `table`, or `schema` to list tablespaces, tables, or schemas, respectively.

You can also list files or tables within a tablespace or schema. The following command lists all of the tables in an Oracle tablespace named `tbl_space3`.

```
symrdb list -type oracle table -tbs tbl_space3
```

The following command displays information about the current Oracle database instance: database name, stats, status, total size, used size, and free space.

```
symrdb show -type oracle
```

The following command displays current Oracle *session* statistics every 30 seconds until it has done so 120 times: the time of day, session ID, and memory usage per second. For information on displaying statistics for other targets, refer to the *EMC Solutions Enabler Symmetrix SRM CLI Product Guide*.

```
symrdb stats -type oracle -i 30 -c 120 -target SESSION
```

The following command examines the extent information of an Informix database file named `big_foot`. The file is located within a database named `sysmaster`.

```
symrdb show -type informix -db sysmaster file /demofs/informix/big_foot
```

Similarly, you can show information about other database objects by omitting the `file` argument and instead including the argument for tablespace (`tbs`), table (`table`), or schema (`schema`) followed by the name of the object.

You can also issue a command that shows information about a `file` or `table` object within a tablespace (`-tbs`) or schema (`-schema`). The following command displays information about the table `sys$table` within an Oracle tablespace named `tbl_space3`.

```
symrdb show -type oracle table sys$table -tbs tbl_space3
```

¹ When listing files for Sybase where the database and transaction log are on the same device, include the `-datalog` option to display data and log as separate file-type items. Otherwise, data and log are both displayed as the data file-type.

Using Daemons for Database Mapping

Beginning with Solutions Enabler version 5.1, you can use a `stord daemon` command to start, stop, or query the status of SRM daemons. The daemons improve the speed of database mapping operations by using a persistent database connection, a fast communication mechanism, and parallel operations. Also, by running a daemon as a privileged user, the application does not have to run as a privileged user.

The daemon service facilitates communication between applications using the SRM API for database mapping/control and one or more database servers. Current applications require no source code changes. Applications use the daemon communication mechanism when the daemon service is running and the database client binaries mechanism when the daemon service is not running.

The connection between an application and a daemon is local. This means that the application and daemon are either on the same host, or the application is on a host that connects to a server that resides on the same host as the daemon. A single daemon can support connections to multiple instances/databases.

During the connection phase, any database environment variables are propagated from the application to the daemon (or for remote connections, to the server), allowing the daemon to set the variables.

You can start and stop a daemon at any time. If an application is already connected via the daemon and the daemon is stopped, the application continues to run (using the database client binaries) but may experience degraded performance. If an application is already connected without the daemon and the daemon is started, the application automatically starts using the daemon.

On the Windows 2000 platform, a daemon should be run as a service. After you install the service, it appears in the service list dialog box and can be accessed by clicking the administrative tools icon. The service does not restart at boot time unless you modify the startup options through the service list dialog box.

You should start a daemon when the system is booted and stop it when the system is shut down. The `stord daemon start` command requires that you specify the name of the daemon. For example, to start the SRM daemon for an Oracle database (`stororad`) and wait five seconds for it to come up (Table 2 lists the daemons that you can use for other database applications):

```
stord daemon start stororad -wait 5
```

Similarly, the `stord daemon shutdown` command stops a daemon.

Table 2. SRM Database Daemons

Daemon Name	Database Application
stororad	Oracle (32-bit)
storora64d	Oracle (64-bit)
storudbd	UDB
storifmxd	Informix
storsqld	SQL
storsybs11d	Sybase version 11
storsybs12d	Sybase version 12
storsybs12.5d	Sybase version 12.5

File System Mapping and File Mapping

The `symhostfs` command allows you to examine attributes of file systems, directories, and files, and determine how these entities are mapped to storage devices. Typically, an operating system manages more than one file system and often more than one *type* of file system. Figure 5 shows two file systems mapped to extents on different physical devices in a storage unit. Each file system type could be VERITAS VxFS or Solaris UFS, one of each, or other supported file systems.

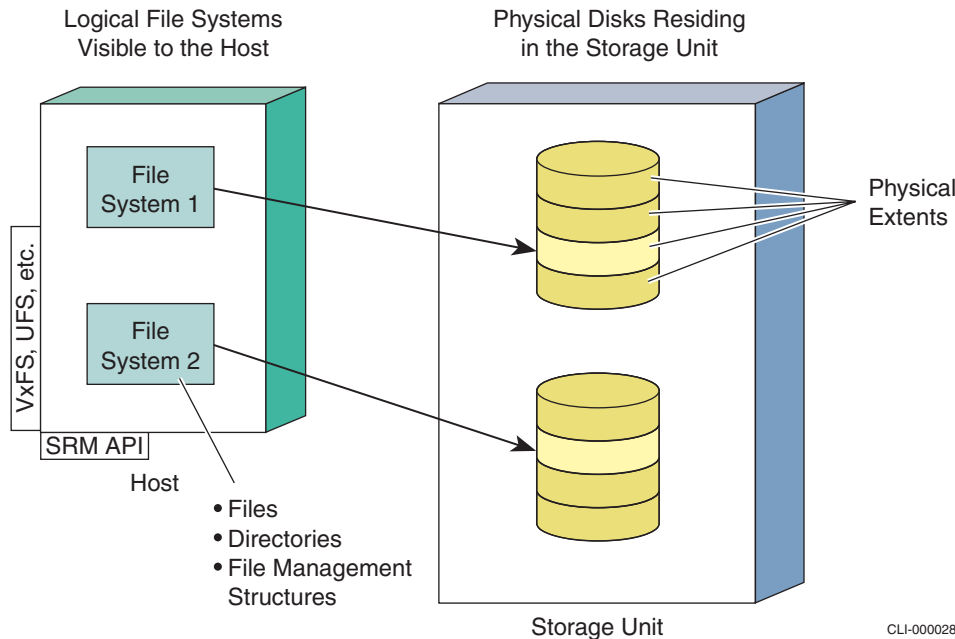


Figure 5. File Systems Mapped to Extents on Two Disks

A file system consists of files, directories, and the file management structures needed to access the files and directories. To display all mounted file systems on your host, enter:

```
symhostfs list
```

The following command displays the attributes of a file system whose mount point is `/mountpt2`, collapsing any allocated physical disk extents that are contiguous and showing extent size in kilobytes.

```
symhostfs -phys_collapse -kb show /mountpt2
```

Among the attributes displayed is a mapping of the file system to the physical disks on which it resides.

The following command includes the `-R` option to list recursively all subdirectories of the `/usr` directory.

```
symhostfs list -R -dir /usr
```

The following command lists all files in directory `/usr/guest`.

```
symhostfs list -file /usr/guest
```

The following command shows detailed information about a file named `big_foot`, collapsing any allocated physical disk extents that are contiguous.

```
symhostfs -phys_collapse show /usr/guest/big_foot
```

For information on collapsing extents, refer to the section “Collapsing a File’s Logical or Physical Extents”.

Logical Volume Mapping

SRM mapping commands allow you to map logical volumes to a detailed view of the underlying storage disks. A logical volume manager (LVM) establishes a volume group by designating a set of physical devices as its members. The physical devices form a pool of disk space that you use to define logical volumes and their specific configurations.

A logical volume's configuration depends on the number of mirrors or plexes of the data that you need to maintain. As a rule, a logical volume maps to a single plex with extents on two different disks to maintain two copies of the data. A logical volume configuration can have multiple physical disks tied together logically (concatenated or striped) to form a volume that is larger than the size of one disk.

A *concatenated volume* is composed of two or more logically connected extents that are located on two or more physical disks. Only one copy of the data is stored on disk, and that data is spread over extents on multiple physical disks. Figure 6 shows that the data layout in a concatenated volume continues to the end of the extent on Disk 1, filling that extent before adding any data on the next disk.

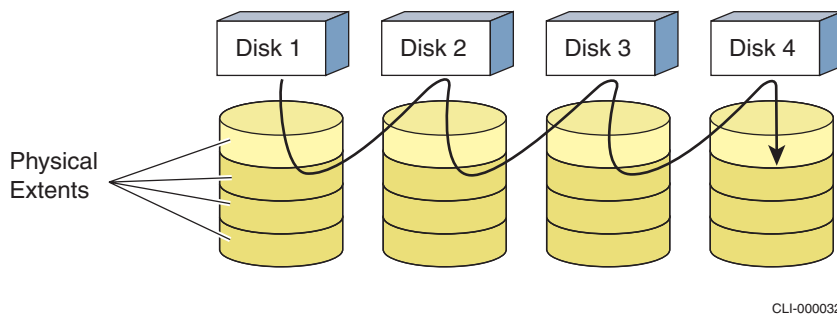


Figure 6. Concatenated Volume on Four Physical Disks

A *striped volume* is one that places data on extents in user-defined stripes or chunks. The stripe size (or chunk size) is the amount of data placed on one disk before moving on to the next disk and placing an equal-size chunk of data there. As Figure 7 illustrates, the extents of a striped volume are configured as a group of *columns*, with each physical disk limited to one column.

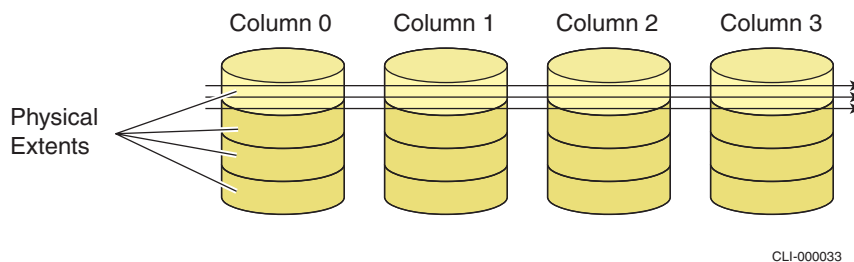


Figure 7. Striped Volume on Four Physical Disks

Equal-size stripes of data are written in sequence first to the extent located on column 0 and then to each remaining column extent. When a stripe has been written to the column-3 extent, the sequence of writing stripes begins again at column 0. Striping continues for the length (capacity) of the columns or, if column lengths vary (as some volume managers allow), until the shortest column is filled.

Striping is unrelated to data protection. Striping is simply a method of placing data on members of the volume to enhance performance. Figure 8 illustrates a striped volume with a 2-way mirror protection configuration. Equal-size stripes of data are written first to the M1 and M2 extents mapped to column 0 and then in sequence to the M1 and M2 extents of each remaining column.

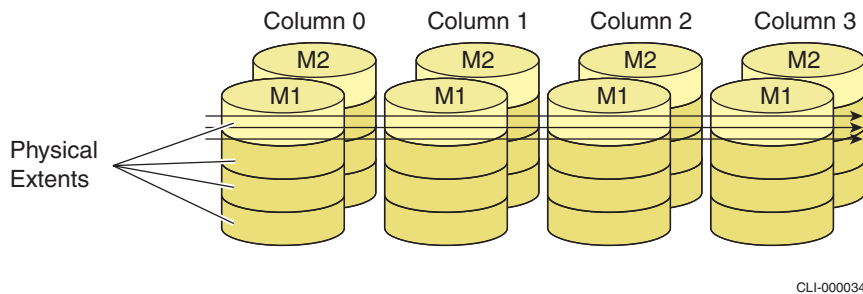


Figure 8. Mirrored Striped Meta Volume on Eight Physical Disks

The physical extents associated with a logical volume can be located on one or more physical disks. Figure 9 shows a striped volume (Vol 1) configured at the host level with two columns. The storage system level has configured two 2-way-mirror physical devices that map to the Vol 1 columns.

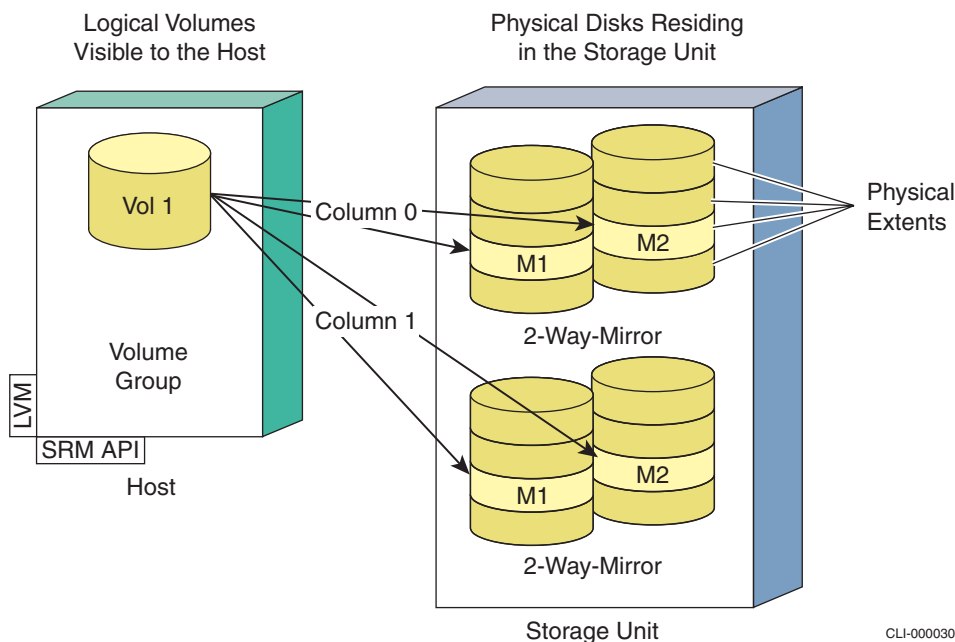


Figure 9. Logical Device Mapped to Two Physical Devices

An identical stripe of data is written to the M1 and M2 extents of column 0. Then a subsequent stripe of data is written to the M1 and M2 extents of column 1. The next stripe of data is written again to the M1 and M2 extents of column 0, and so forth.

You can examine the physical extents associated with each mirror using the following command.

```
symmlv -g ProdBgrp show Vol1 -collapse
```

The information displayed is for a volume named Vol1 in a volume group named ProdBgrp. Including the `-collapse` option collapses any of the volume's physical extents stored contiguously on the disk to form one extent. The offset displayed is the offset of the first extent in the collapsed set.

If you use the `-expand` option when displaying a *striped* meta volume, the extent list takes the form of a unique extent description for each stripe in the mirror. For example:

```
symmlv -g ProdBgrp show Vol1 -expand
```

The extent size is the stripe size. The physical extent count is the size of the logical volume divided by the size of the stripe.

The following commands list all volume groups on your host, examine the details of a volume group named ProdBgrp, and list all logical volumes in the ProdBgrp volume group.

```
symvg list
symvg show ProdBgrp
symmlv -g ProdBgrp list
```

Mapping Logical Objects to Symmetrix Physical Devices

The `symrslv` mapping commands resolve a logical object's mapping to Symmetrix physical devices, drilling down directly to the Symmetrix device layer and locating the extents there. No information about logical extents is returned. The commands provide information about Symmetrix device extents of the following objects:

- Physical devices
- Logical volumes
- Regular files in file systems mounted on physical devices² or on logical volumes of an LVM³
- Directories
- File systems

Figure 10 on the next page shows two host objects, a file and a physical device, each mapped to a 2-way mirrored physical device in the storage unit. The storage level configuration results in each object's physical extents spanning two disks.

² On Windows 2000 and NT 4.0, the `symrslv` command does not support directories of system files like PAGEFILE.SYS.

³ On Windows 2000, the `symrslv` command does not support files stored on NT DiskAdmin logical volumes.

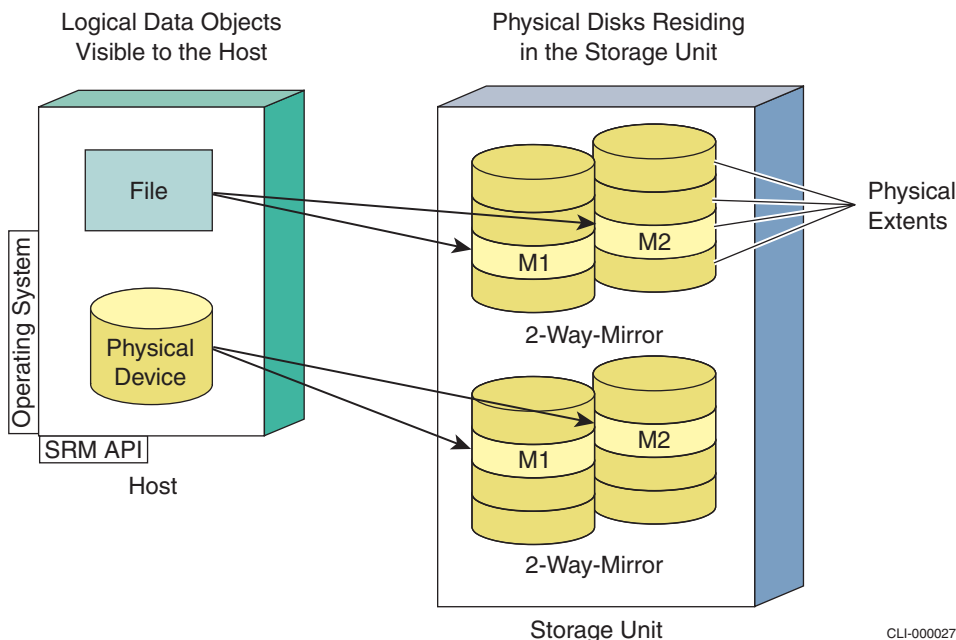


Figure 10. Data Objects Mapped to Physical Devices

You can use the `symhostfs list` command with the `-file` option to display all files in a directory (for example, in the directory `/usr/guest`).

```
symhostfs list -file /usr/guest
```

If your file of interest is named `big_foot`, you can examine the file's data mapping attributes, including extent data in megabytes (`-mb`), which is the default, kilobytes (`-kb`), or blocks⁴.

```
symrslv -blocks file /usr/guest/big_foot
```

Different operating systems employ different formats for naming a physical device. For example, a Solaris host accesses a physical disk using a physical device name such as `c#t#d#`, where `c#` is the controller, `t#` is the target ID, and `d#` is the disk number. If the physical device refers to a partition, then `s#` is appended to the device name. If a partition takes up an entire physical disk, its partition number is `s2`, as shown in the following output.

If a physical device is a Symmetrix or CLARiiON device, you can use the `sympd list` command to display devices that are visible to a host connected to the storage unit (assuming that the appropriate host utilities have been run to recognize devices that have been configured on the storage unit).

For example, the following command lists the physical devices on a Symmetrix storage unit with an ID of 000000003264. The ellipsis (.....) represents truncated output.

⁴ Currently, SRM supports storage units that service both mainframe (CKD) and open system (FBA) hosts. In FBA (fixed block) architecture, one block equals 512 bytes. In CKD, one block equals the size of one track (47,476 or 56,664 bytes, depending on whether the unit is a 3380 or 3390).

sympd -sid 3264 list

Symmetrix ID: 000000003264

Device Name		Directors				Device			
Physical	Sym	SA	:P	DA	:IT	Config	Attribute	Sts	Cap (MB)
/dev/rdsk/c4t0d0s2	084	16B:0		01A:C0		2-Way Mir	N/Grp'd	RW	516
/dev/rdsk/c4t0d1s2	085	16B:0		02A:C0		2-Way Mir	N/Grp'd	RW	516
/dev/rdsk/c4t0d2s2	086	16B:0		01B:C0		2-Way Mir	N/Grp'd	RW	516
/dev/rdsk/c4t0d3s2	087	16B:0		02B:C0		2-Way Mir	N/Grp'd	RW	516

The following command displays physical extent information for physical device /dev/rdisk/c4t0d3s2.

```
symrslv pd /dev/rdisk/c4t0d3s2 -collapse
```

The following commands and command line options display the data mapping attributes of other data objects: a file system with a mount point of / (but omitting the extent information), a directory (/usr) with extent size specified explicitly in blocks, and a logical volume (Lv13) in a logical volume group named ProdBgrp.

```
symrslv -no_extents fs /
symrslv -blocks dir /usr
symrslv -pdev_extents -stripe_column -g ProdBgrp lv Lv13
```

The `-pdev_extents` option displays the physical extents of a meta device without showing the underlying configuration of the meta device. For a striped volume, the `-stripe_column` option includes an extent's stripe column number in the output display.

The section "File System Mapping and File Mapping" provides more information on mapping files and file systems. The section "Logical Volume Mapping" describes mapping logical volumes in greater detail.

Displaying Partition Information about a Physical Device

Beginning with Solutions Enabler version 6.0, you can display detailed information about the geometry and layout of a physical device (the name by which the host system recognizes a storage device). This includes information specific to the partitions of the device, such as partition name, type, attributes, offset into the full device, and the size of the partition. This information is meaningful for platforms that allow partitioning of host-visible devices.

The following command example will display partition information in kilobytes about a physical device (c2t0d2s2) on a SunOS system.

```
sympart -kb show c2t0d2s2
```

The following command example will display partition information in blocks about a physical device (PHYSICALDRIVE9) on a Windows system.

```
sympart -blocks show PHYSICALDRIVE9
```

Naturally, physical devices for which you are obtaining partition information need to have been formatted, labeled, and partitioned beforehand.

Performing Database I/O Control Operations

The SRM component allows you to send control commands to a database server. For example, you can temporarily suspend database writes to enforce a consistent and stable image of the database on multiple disks.

Using Daemons for Database I/O Control Operations

Beginning with Solutions Enabler version 5.1, you can use a `stord daemon` command to start, stop, or query the status of daemons that improve reliability of database control operations. The daemon creates a separate connection thread that maintains a persistent database connection across database freeze/thaw operations. This connection thread avoids problems where a database is frozen and the application is unable to reconnect to thaw the database.

For more information on using daemons, refer to the earlier section “Using Daemons for Database Mapping” and to *EMC Solutions Enabler Symmetrix SRM CLI Product Guide*.

Suspending Writes to the Database

You can place a database in backup mode (for Oracle) or momentarily freeze a database (such as Informix, SQL Server, etc.) for the time it takes you to perform a split, snap, or some other *short* operation that consumes no more than a few seconds.⁵ Using the `symioctl` command allows you to suspend any database updates being written to the disk. You can then perform your short operation and use the `symioctl` command again to resume access to the database.

Freezing an Oracle Database

The following sequence places the Oracle database in hot backup mode, freezes the database (`-type` is `oracle`), performs a short operation, thaws the database, and ends backup mode.

```
symioctl begin backup -type oracle
symioctl freeze -type oracle
<Perform some short operation such as split or snap>
symioctl thaw -type oracle
symioctl end backup -type oracle
```

This operation on an Oracle database does not directly affect meta data I/O. The operation does not affect any Oracle files that are not related to this database instance or any non-Oracle files in the same file system.

Freezing an Informix Database

The following sequence freezes an Informix database (`-type` is `informix`), allowing you to perform some short operation, and then thaws the database.

```
symioctl freeze -type informix
<Perform some short operation such as split or snap>
symioctl thaw -type informix
```

Once you complete your database task, the thaw action resumes full I/O access to the database.

⁵ Interrupting database access for more than a few seconds is not recommended.

Freezing a Sybase Database

The following sequence freezes a Sybase database (`-type is sybase`), allowing you to perform some short operation, and then thaws the database.

```
symioctl suspend -type sybase
<Perform some short operation such as split or snap>
symioctl resume -type sybase
```

Freezing an IBM UDB Database

The following sequence freezes an IBM UDB database (`-type is ibmudb`), allowing you to perform some short operation, and then thaws the database.

```
symioctl suspend -type ibmudb
<Perform some short operation such as split or snap>
symioctl resume -type ibmudb
```

Freezing an SQL 2000 Database

Snapshot commands for SQL Server 2000 and higher support BACKUP and RESTORE database WITH SNAPSHOT operations using the Virtual Device Interface (VDI). The `symioctl begin snapshot` command sends an SQL command to SQL Server, beginning the snapshot backup and suspending writes for the specified database.

The `symioctl begin snapshot` command suspends writes to a database named `mydb` and creates a savefile named `mydb.sav`. Then perform some short operation while the writes are suspended.

```
symioctl begin snapshot -type sqlserver mydb savefile c:\temp\mydb.sav
<Perform some short operation such as split or snap>
symioctl end snapshot -type sqlserver mydb
```

The `symioctl end snapshot` command causes the “BACKUP DATABASE” SQL command for the `mydb` database to complete. Writes to the database can resume, and the snapshot meta data is saved to the savefile. The meta data from the savefile is used by SQL Server to logically restore the database if needed (by using a subsequent `symioctl restore snapshot` command). For example:

```
symioctl restore snapshot -type sqlserver mydb savefile c:\temp\mydb.sav
```

Starting Up and Shutting Down a Database Server Manager

Beginning with Solutions Enabler version 5.3, you can start up and shut down a database server manager using many options. These options vary with the type of database manager and are described in the *EMC Solutions Enabler Symmetrix SRM CLI Product Guide*. For example, you can start up an Oracle database manager in modes such as read-only, read-write, recover, and so forth.

```
symrdb startup -type oracle OPEN_READ_ONLY
```

The following command shuts down an Oracle database manager in mode NORMAL.

```
symrdb shutdown -type oracle NORMAL
```

Performing Control Operations on a Volume Group

The following two sections describe control operations that you can perform in a LVM environment.

Deporting and Importing a Volume Group

Beginning with Solutions Enabler version 5.1, you can deport a logical volume group's meta data from a system to persistent storage and import this same volume group information later to another host (for example, a backup server). Volume group meta data exists on the storage unit (on logical volumes and in files on other disks) and on the production host (in the Registry and in a daemon service). Deporting involves gathering this meta data so that it can be reconstructed in a different location.

The deport operation deports a volume group from one host system so that it can be imported on another host system of the same type. Once you deport a volume group, this group is not available on the system from which it was deported.

For example, if you have a volume group spanning several disks and want a copy of the volume group on another system, you can perform the following sequence:

1. Deport the volume group.
2. Split the disks contained in the volume group as BCVs.
3. Import the volume group encapsulating the BCVs into another host system. The import operation makes the volume group available to this new host system.

For more information on deporting and importing a volume group, refer to "Example 4: Deporting and Importing a Volume Group."

Creating a Volume Group

Beginning with Solutions Enabler version 5.3, you can create a volume group, add and remove devices from the volume group, and recover a failed volume group.

Creating a volume group requires that you specify a volume group name and one or more devices. For example, the volume group named volgrp2 and two devices specified by their physical device names.

```
symvg create volgrp2 /dev/vx/rdmp/c4t1d6s2 /dev/vx/rdmp/c4t1d7s2
```

If you have more than one LVM type on a given host operating system, you may need to specify in the command the `-type VgType` option. For example, the create operation can be performed on two types of LVMs on Solaris operating systems: VERITAS and EMC PowerVolume.™ To specify one of these types, you would include the `-type sun_vxvm` or `-type emc_pvm` option. SRM designates a certain LVM as the default with respect to certain host operating systems. In the case of the Solaris operating systems, omitting the `-type` option means that the default is VERITAS. Refer to *EMC Solutions Enabler Symmetrix SRM CLI Product Guide* for a list of volume group types and supported control operations.

You can use the `symvg destroy` command to remove a volume group.

You can extend a volume group by adding additional devices. For example, extending the volume group by adding two more devices:

```
symvg adddev volgrp2 /dev/vx/rdmp/c4t1d3s2 /dev/vx/rdmp/c4t1d4s2
```

If you decide to remove one or more devices from the volume group, use the `symvg rmdev` command:

```
symvg rmdev volgrp2 /dev/vx/rdmp/c4t1d3s2
```

There may be situations in which you want to recover a failed volume group. The most common example is when a disk belonging to a volume group has gone away and is coming back up again. For example, if you need to recover the failed volume group named `volgrp2`:

```
symvg recover volgrp2
```

Performing Control Operations on Logical Volumes

Beginning with Solutions Enabler version 5.3, you can control logical volumes with operations such as creating and deleting a logical volume, adding mirror images to a logical volume, changing the size of a logical volume, and displaying performance statistics. Refer to *EMC Solutions Enabler Symmetrix SRM CLI Product Guide* for a list of volume group types, supported control operations, and options to the basic control commands.

Creating a logical volume requires that you specify a volume group name as well as the name and size of the logical volume that you are creating. For example, to create a logical volume named `testvol` in the volume group `volgrp2`. The volume group size will be 204800 blocks (512 bytes per block).

```
symlv -g volgrp2 create testvol -size 204800
```

If you have more than one LVM type on a given host operating system, you may need to specify in this and other control commands the `-type VgType` option (see the previous section for more details).

You can add mirrors to a logical volume and optionally specify physical devices for the mirrors or striping parameters or both. The following example adds one mirror to `testvol` and specifies that this mirror should map to a particular physical device.

```
symlv -g volgrp2 add testvol -nmirror 1 /dev/vx/rdmp/c4t1d6s2
```

You can extend the size of a logical volume by specifying the *total* size in blocks. For example, if `testvol` is already at 204800 blocks (as above) and you want extend it to 224800 blocks:

```
symlv -g volgrp2 extend testvol -size 224800
```

You can use `symlv delete`, `remove`, and `reduce` to delete a logical volume, remove its mirrors, or reduce its size, respectively. Refer to *EMC Solutions Enabler Symmetrix SRM CLI Product Guide* for a list of supported LVM types.

It can sometimes be helpful to examine the performance statistics of a logical volume — data such as read and write requests per second, kilobytes read and written per second, and so forth. The `symlv stats` command allows you to examine statistical data of one of more logical volumes. For example, to display statistics 60 times at 30-second intervals for all logical volumes of VxVM on a SUN operating system:

```
symlv stats -i 30 -c 60 -type sun_vxvm
```

Collapsing a File's Logical or Physical Extents

Collapsing a file's extent data is advisable if a file is extremely large (for example, a file allocated on a striped logical volume could have a million extents, requiring a prohibitive amount of time and memory to display them all). Where size and display time are not prohibitive, you have the option to expand a file's extents so that contiguous extents are displayed as separate extents.

The `-phys_collapse` option collapses any of the file's physical disk extents that are stored contiguously and forms one larger extent from smaller contiguous extents. For example, either of the following commands locates and examines the collapsed *physical* extents of a file named `big_foot`:

```
symhostfs show /usr/guest/big_foot -phys_collapse
symrslv file /usr/guest/big_foot -phys_collapse
```

To collapse a file's *logical* extents, use the `-collapse` option instead.

The `symhostfs` command displays file information as well as extent data, whereas `symrslv` examines just the extent information at the physical device layer. The `symrslv` command drills down directly to the physical device layer and locates the extents there; it does not display anything else.

Collapsing logical extents collapses two contiguous extents *if this operation does not limit your ability (or an application's ability) to reconstruct the original object by expanding the collapsed extents*. For example, consider a file that has been allocated two 4-kilobyte extents, E1 and E2. If these extents are reversed on the disk (that is, E2 is at byte offset 4096, and E1 is at byte offset 8192), a logical collapse does *not* collapse E1 and E2, because the extents are not located in an ascending sequence (E1 is not located at the lowest byte offset). Expanding logical extents always returns extents in ascending sequence (lowest to highest byte offset).

On the other hand, a physical collapse does return one extent in this case, because SRM does not evaluate the sequence of the extents prior to this operation.

Caution: Do not perform a physical collapse if your application needs to reconstruct the original file from the collapsed extents. If you do, your reconstructed file may not match the original file.

A *logical* collapse is required when an application needs to reconstruct (or access) the original file using only the collapsed extents. Consider an application in Figure 11 that needs to locate 120 kilobytes of data in a file, beginning at byte offset 240 KB. If a logical collapse returns twenty 100-KB extents for the file, the application can calculate that the data of interest is 60 kilobytes at offset 40 KB in the third extent, and 60 kilobytes at offset 0 in the fourth extent. Locating the data in this way is possible because the twenty extents are guaranteed to always remain in ascending sequence.

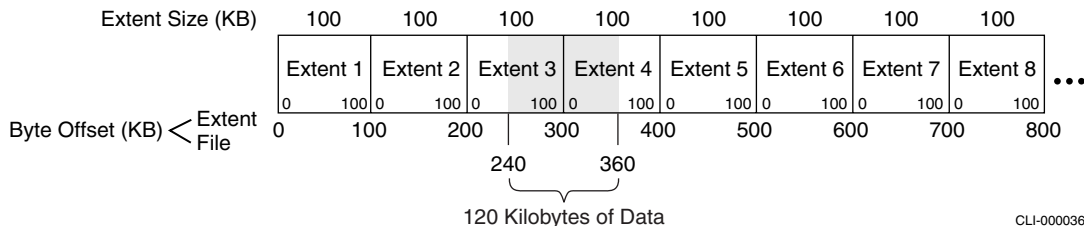


Figure 11. Sequential Order of File Extents in a Logical Collapse

On the other hand, an application like EMC's Double Checksum for Oracle needs to protect data blocks on those tracks accessed for a 128-kilobyte file (for example, tracks 6, 5, 4, 10 on one disk as shown in Figure 11). The capacity of each track is 32 kilobytes. The application is not concerned with locating the data on these tracks, just which tracks on the disk are accessed when writing to this file. In this case, it doesn't matter whether byte offset N of the file is on track 6 or 5 or 4 or 10. An SRM operation that performs a physical collapse on this file returns two extents (96 kilobytes at offset 128 KB, and 32 kilobytes at offset 320 KB), indicating which tracks have data blocks that need to be protected.

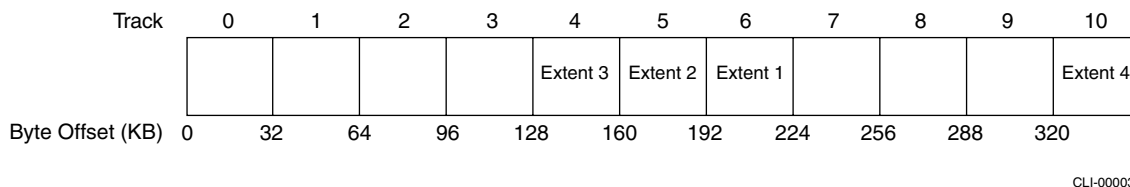


Figure 12. Random Order of File Extents in a Physical Collapse

Extent Transformation in the Mapping Layers

The number of extents that are allocated for a logical device can differ in the mapping layers, affecting SRM's return of extent data at a particular layer. At the highest mapping layer (what the host sees), the logical device might appear as a simple 100-character file. At the lowest mapping layer (what the storage unit sees), if that same device were configured as a 2-way mirror, it might appear as two identical 100-character files on separate disks. Figure 13 shows how the number and size of extents allocated for a file might differ between mapping layers Level A (the host level) and Level B (the storage level), depending on the device configuration.

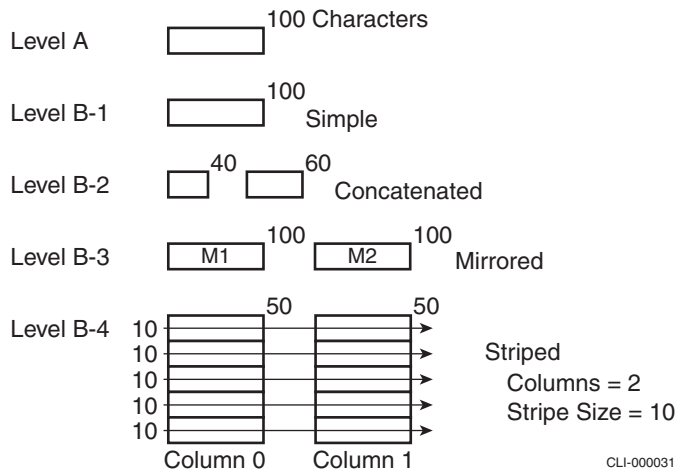


Figure 13. Transforming a Device's Extents in the Mapping Layers

Level A represents a logical device to which the host system has allocated a 100-character extent. If the storage-level configuration requires a simple 100-character mapping to disk, the device's extent size and character appear the same at Level B. However, if the physical device configuration is concatenated, its appearance at Level B might be two extents (for example, 40 characters and 60 characters). If the device is a 2-way mirror, that single extent at Level A appears as two 100-character extents (M1 and M2) at Level B. If the physical device is configured as striped with two columns and a 10-character stripe size, the transformation at Level B is to two 50-character extents (each write operation consuming a 10-character stripe on each column extent).

Thus, the mapping commands allow you to drill down to various storage layers to locate and examine a device's extents at that layer. For example, you can drill down from an upper-layer Oracle database to the physical device, array logical device, or meta/composite device layers.

- Oracle database
- Tablespace
- Data file
- File system
- Logical volume
- Plex
- Subdisk
- Physical device
- Array logical device
- Meta or composite device
- Physical disk

Examples: Symmetrix Storage Devices

Example 1: Displaying Relational Database Objects

This example examines database objects for three different types of databases: Oracle, IBM UDB, and Informix.

Examining Oracle Database Objects

The hardware/software setup for this example consists of a Solaris host connected to a local Symmetrix unit. Oracle version 8.1.7.0.0 software is installed on the host.

For Solutions Enabler to access a database, set the `SYMCLI_RDB_CONNECT` environment variable to the username and password of the system administrator's account. This first `export` command sets the variable to a username of "system" and a password of "manager."

```
# export SYMCLI_RDB_CONNECT=system/manager
```

Setting `ORACLE_HOME` specifies the location of the Oracle binaries. Setting `ORACLE_SID` specifies the database name (64817).

```
# export ORACLE_HOME=/db1/solaris/2.6/oracle/8.1.7
# export ORACLE_SID=64817
```

The `symrdb list` command lists the current Oracle database (64817) and allows you to test basic database connectivity.

```
# symrdb list -type oracle
```

```
DATABASE NAMES (ORACLE 8.1.7.0.0):
```

```
Database Name
-----
```

```
64817
```

The `symrdb list file` command identifies all the files of the current Oracle database.

```
# symrdb list file -type oracle
```

```
DATABASE FILE NAMES (ORACLE 8.1.7.0.0):
```

DB File Name -----	Type -----	Status -----
/usr/oracle/oradata/64817/control01.ctl	Control	Online
/usr/oracle/oradata/64817/rbs01.dbf	Data	Online
/usr/oracle/oradata/64817/system01.dbf	Data	Online
/usr/oracle/oradata/64817/temp01.dbf	Data	Online
/usr/oracle/oradata/64817/redo01.log	Log	Online
/usr/oracle/oradata/64817/redo02.log	Log	Online
/usr/oracle/oradata/64817/redo03.log	Log	Online

The `symrdb list table` command displays all tables in the tablespace named SYSTEM. The ellipsis (.....) indicates where output was omitted for brevity.

```
# symrdb list table -type oracle -tbs SYSTEM
```

```
TABLE SPACE NAME      : SYSTEM
DATABASE TABLE NAMES (ORACLE 8.1.7.0.0) :
```

Table Name	Owner	Type
-----	-----	-----
OL\$	OUTLN	Simple Table
OL\$HINTS	OUTLN	Simple Table
ACCESS\$	SYS	Simple Table
AQ\$_MESSAGE_TYPES	SYS	Simple Table
AQ\$_PENDING_MESSAGES	SYS	Simple Table
AQ\$_PROPAGATION_STATUS	SYS	Simple Table
AQ\$_QUEUE_STATISTICS	SYS	Simple Table
AQ\$_QUEUE_TABLE_AFFINITIES	SYS	Simple Table
AQ\$_SCHEDULES	SYS	Simple Table
ARGUMENT\$	SYS	Simple Table
ASSOCIATION\$	SYS	Simple Table
ATTRCOL\$	SYS	Cluster
ATTRIBUTE\$	SYS	Cluster
AUD\$	SYS	Simple Table
AUDIT\$	SYS	Simple Table
AUDIT_ACTIONS	SYS	Simple Table
BOOTSTRAP\$	SYS	Simple Table
CCOL\$	SYS	Cluster
CDEF\$	SYS	Cluster
CLU\$	SYS	Cluster
COL\$	SYS	Cluster
COLLECTION\$	SYS	Cluster
COLTYPE\$	SYS	Cluster
COM\$	SYS	Simple Table
CON\$	SYS	Simple Table
CONTEXT\$	SYS	Simple Table
DBMS_ALERT_INFO	SYS	Simple Table
DBMS_LOCK_ALLOCATED	SYS	Simple Table
.....		
AQ\$_QUEUES	SYSTEM	Simple Table
AQ\$_QUEUE_TABLES	SYSTEM	Simple Table
AQ\$_SCHEDULES	SYSTEM	Simple Table
DEF\$_AQCALL	SYSTEM	Simple Table
DEF\$_AQERROR	SYSTEM	Simple Table
DEF\$_CALLDEST	SYSTEM	Simple Table
DEF\$_DEFAULTDEST	SYSTEM	Simple Table
DEF\$_DESTINATION	SYSTEM	Simple Table
DEF\$_ERROR	SYSTEM	Simple Table
DEF\$_LOB	SYSTEM	Simple Table
DEF\$_ORIGIN	SYSTEM	Simple Table
DEF\$_PROPAGATOR	SYSTEM	Simple Table
DEF\$_PUSHED_TRANSACTIONS	SYSTEM	Simple Table
DEF\$_TEMP\$LOB	SYSTEM	Simple Table
SQLPLUS_PRODUCT_PROFILE	SYSTEM	Simple Table

Beginning with Solutions Enabler version 5.1, you can use the `symrdb list tbs` command to display tablespaces, their allocated size, and their free space. Megabytes (mb) is the default.

```
# symrdb -type oracle list tbs
```

TABLE SPACE NAMES (ORACLE 8.1.7.0.0):

Table Space Name	Type	Status	Size (mb)	Free (mb)
-----	-----	-----	-----	-----
LOTSAFILES1	Permanent	Online	3	1
LOTSAFILES10	Permanent	Online	3	1
LOTSAFILES11	Permanent	Online	3	1
LOTSAFILES12	Permanent	Online	3	1
LOTSAFILES13	Permanent	Online	3	1
LOTSAFILES14	Permanent	Online	3	1
LOTSAFILES15	Permanent	Online	3	1
LOTSAFILES16	Permanent	Online	3	1
LOTSAFILES17	Permanent	Online	3	1
LOTSAFILES18	Permanent	Online	3	1
LOTSAFILES19	Permanent	Online	3	1
LOTSAFILES2	Permanent	Online	3	1
LOTSAFILES20	Permanent	Online	3	1
LOTSAFILES21	Permanent	Online	3	1
LOTSAFILES22	Permanent	Online	0	0
LOTSAFILES3	Permanent	Online	3	1
LOTSAFILES4	Permanent	Online	3	1
LOTSAFILES5	Permanent	Online	3	1
LOTSAFILES6	Permanent	Online	3	1
LOTSAFILES7	Permanent	Online	3	1
LOTSAFILES8	Permanent	Online	3	1
LOTSAFILES9	Permanent	Online	3	1
SYSTEM	Permanent	Online	75	23
USERS	Permanent	Online	1	0

Examining IBM DB2/UDB Database Objects

The hardware/software setup for this example consists of a Solaris host connected to a local Symmetrix unit. IBM UDB version 7.1.0 software is installed on the host.

For Solutions Enabler to access a database, set the `SYMCLI_RDB_CONNECT` environment variable to the username and password of the system administrator's account. This first `export` command sets the variable to a username of "db2v7in1" and a password of "redsox."

```
# export SYMCLI_RDB_CONNECT=db2v7in1/redsox
```

When running a DB2/UDB instance with Solaris, you need to set `LD_LIBRARY_PATH` as follows. Also, set the variable `DB2INSTANCE` to the name of the database instance (for example, db2v7in1).

```
# export LD_LIBRARY_PATH=/usr/db2v7in1/sqllib/lib
# export DB2INSTANCE=db2v7in1
```

The `symrdb list` command lists the current databases (QU1 and SAMPLE) within the IBM UDB database instance defined above (db2v7in1).

```
# symrdb list -type ibmudb
```

```
DATABASE NAMES (IBMUDB 7.1.0):
```

```
Database Name
-----
```

```
QU1
SAMPLE
```

The `symrdb list tbs` command displays tablespaces in the database named SAMPLE. Note that the database (`-db`) name that you enter is case sensitive.

```
# symrdb list tbs -type ibmudb -db SAMPLE
```

```
DATABASE          : SAMPLE
TABLE SPACE NAMES (IBMUDB 7.1.0):
```

Table Space Name	Type	Status
-----	-----	-----
SYSCATSPACE	Permanent	Online
TEMPSPACE1	Temporary	Online
USERSPACE1	Permanent	Online

Examining Informix Database Objects

The hardware/software setup for this example consists of a Solaris host connected to a local Symmetrix unit. Informix version 7.3.1 software is installed on the host.

For Solutions Enabler to access a database, set the `SYMCLI_RDB_CONNECT` environment variable to the username and password of the system administrator's account. The `export` command sets the variable to a username of "informix" and a password of "Informix."

```
# export SYMCLI_RDB_CONNECT=informix/Informix
```

The `export INFORMIXDIR` command specifies the location of the Informix binaries. The `export ONCONFIG` command specifies the Informix configuration file. The `export INFORMIXSERVER` command specifies the server name (that is, the test64731 database instance).

```
# export INFORMIXDIR=/db1/solaris/2.6/informix/7.3.1
# export ONCONFIG=onconfig.test64731
# export INFORMIXSERVER=test64731
```

The `symrdb list` command lists current databases for the Informix server defined above (test64731).

```
# symrdb list -type informix
```

```
DATABASE NAMES (INFORMIX 7.31.UD1):
```

```
Database Name
-----
```

```
sysmaster
sysutils
```

The `symrdb show file` command locates extent data on a sysmaster database file named `llog.dbs`, including the physical devices on which the file data extents reside and the mirror configuration at the operating system level. "Database File Block Size" means that this version of Informix reads a minimum of four blocks on accessing the file. This file is configured as a striped mirror with two columns and a stripe size of 256 blocks. Each column maps to a different physical disk.

```
# symrdb show -type informix -db sysmaster -blocks \
               file /demofs/informix/test64731/llog.dbs
```

```
DATABASE          : sysmaster
DATABASE FILE NAMES (INFORMIX 7.31.UD1):
```

```
Database File Name : /demofs/informix/test64731/llog.dbs
```

```
Database File Type      : Log
Database File Status    : Online
Database File Size      : 4000b
```

```
Database File Data Offset : 0b
Database File Block Size  : 4b
```

```

Absolute Path           : /demofs/informix/test64731/llog.dbs
Resolved Object Type    : SunOS UFS File
Resolved Object Size    : 4000b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 2
File System Mount Point : /demofs
File System Device Name : /dev/vx/dsk/doug/Striped

```

```
Number of Mirrors for object (1):
```

```

{
1) Mirror Configuration

Mirror Stripe Columns : 2
Mirror Stripe Size    : 256b

```

As this example illustrates below, it is not necessary for a file's extents to align with the boundaries of a striped logical volume. The first three extents on Dev 00BC are "backwards" in the stripe. The fourth and fifth extents on Dev 00BC begin 1008 blocks later (from Offset 35728 to 36736). Finally, the first two extents are contiguous but in reversed order and, therefore, are not collapsed for this "expanded" list.

```
Mirror Physical Extents (20):
```

```

{
-----

```

Size	Array	Dev	Offset	PPdevName	Offset
80b	00541	00BC	35728b	/dev/vx/rdmp/c2t0d0s2	35728b (S)
48b	00541	00BC	35680b	/dev/vx/rdmp/c2t0d0s2	35680b (S)
16b	00541	00BC	35648b	/dev/vx/rdmp/c2t0d0s2	35648b (S)
48b	00541	00BC	36736b	/dev/vx/rdmp/c2t0d0s2	36736b (S)
128b	00541	00BC	36800b	/dev/vx/rdmp/c2t0d0s2	36800b (S)
256b	00541	00BD	35712b	/dev/vx/rdmp/c2t0d1s2	35712b (S)
256b	00541	00BC	36928b	/dev/vx/rdmp/c2t0d0s2	36928b (S)
256b	00541	00BD	35968b	/dev/vx/rdmp/c2t0d1s2	35968b (S)
256b	00541	00BC	37184b	/dev/vx/rdmp/c2t0d0s2	37184b (S)
256b	00541	00BD	36224b	/dev/vx/rdmp/c2t0d1s2	36224b (S)
256b	00541	00BD	36992b	/dev/vx/rdmp/c2t0d1s2	36992b (S)
256b	00541	00BC	38208b	/dev/vx/rdmp/c2t0d0s2	38208b (S)
256b	00541	00BD	37248b	/dev/vx/rdmp/c2t0d1s2	37248b (S)
256b	00541	00BC	38464b	/dev/vx/rdmp/c2t0d0s2	38464b (S)
96b	00541	00BD	37504b	/dev/vx/rdmp/c2t0d1s2	37504b (S)

```

}

```

```
Mirror Physical Devices (2):
```

```

{
-----

```

Array	Dev	PPdevName	PdevName
00541	00BC	/dev/vx/rdmp/c2t0d0s2	/dev/vx/rdmp/c2t0d0s2 (S)
00541	00BD	/dev/vx/rdmp/c2t0d1s2	/dev/vx/rdmp/c2t0d1s2 (S)

```

}
-----

```

Examining Microsoft Exchange Database Objects

Beginning with Solutions Enabler version 5.1, you can examine Microsoft Exchange database objects. The hardware/software setup for this example consists of a Windows 2000 host connected to a local Symmetrix unit. Microsoft Exchange version 6.0.4712.2 software is installed on the host.

Microsoft Exchange does not require setting the `SYMCLI_RDB_CONNECT` variable or any database environmental variables in order to accessing the database.

The following `symrdb list file` command displays database files in an Exchange storage group named SG2.

```
C:\> symrdb -type exchange list file -tbs SG2
```

```
TABLE SPACE NAME      : SG2
DATABASE FILE NAMES (EXCHANGE 6.0.4712.2):
```

DB File Name -----	Type -----	Status -----
T:\SG2PrivA.edb	Data	Online
T:\SG2PrivA.stm	Data	Online
T:\SG2PrivB.edb	Data	Online
T:\SG2PrivB.stm	Data	Online
L:\E01.log	Log	Online
L:\E0100009.log	Log	Online
L:\E010000A.log	Log	Online
L:\res1.log	Log	Online
L:\res2.log	Log	Online

Example 2: Mapping Files and Other Disk Storage Objects

The hardware setup for this example consists of a Sun Solaris host connected to a Symmetrix storage unit. The files displayed are within a Sun UFS file system. The SunOS VxVM logical volume manager manages a logical volume included in the example. In this example and all examples sections, an object is mapped to the Symmetrix device level (in some cases, to the Symmetrix *meta* device level).

The `symhostfs list` command with the `-file` option displays all of the files in a specified directory (`/fs_test1`). The display includes file size and access information.

```
# symhostfs list -file /fs_test1
```

```
Directory Name : /fs_test1
FileSystem Type : SunOS UFS File
```

File Name	Size	Perms	Owner	Group
-----	----	-----	-----	-----
file1	1m	-rw-r--r--	0	1
file1k	1k	-rw-r--r--	0	1

The `symhostfs show` command expands the extents of a file named `file1` and displays that extent data in blocks (1500) along with other characteristics of the file. "Device Name" in the display indicates the physical device on which the file is stored.

```
# symhostfs show /fs_test1/file1 -expand -blocks
```

```
File Name           : /fs_test1/file1
File Type           : SunOS UFS File
File Size           : 1500b
Number of Trailing Bytes : 0
Extent byte offset to data : 0

File Mode           : 100644
File Permission     : -rw-r--r--
File Owner ID       : 0
File Group ID       : 1

Number of Symbolic Links : 1

Last Access Time    : Fri 12-Apr-2002 16:16
Last Modification Time : Fri 12-Apr-2002 16:16
Last Status Change Time : Fri 12-Apr-2002 16:16

Device Name         : /dev/vx/rdisk/fsvg/Striped
Inode Number        : 4
File System Fragment Size : 2b
```

“Offset in File” shows the ascending sequence of the logical extents in blocks (b). “Offset in Device” relates to the host device or logical volume and has no predictable sequence. The ellipsis (.....) indicates where output was omitted for brevity. “Metadata,” while associated with the file, is not technically part of the file and thus is “N/A” in the column “Offset in File.”

```

Number of Extents in File      : (95)
{
-----
Extent      Offset in      Extent      Offset
Type        Device        Size        in File
-----
Data         1680b         16b         0b
Data         1696b         16b         16b
Data         1712b         16b         32b
Data         1728b         16b         48b
Data         2000b         16b         64b
Data         1776b         16b         80b
Data         1792b         16b         96b
Data         1824b         16b        112b
Data         2208b         16b        128b
Data         2224b         16b        144b
Data         2176b         16b        160b
Data         2320b         16b        176b
Metadata     100368b        16b         N/A
Data         100432b        16b        192b
Data         100384b        16b        208b
.....
Data        103824b         16b       1472b
Data        103840b         12b       1488b
}

```

SRM locates the physical extents on the physical devices where the file data resides and displays the mirror configuration at the operating system level, not the storage system level. (Data management software on the host has no knowledge of storage system mirrors — for example, Symmetrix mirrors M1, M2, M3, and M4. However, each layer on the host has the option to mirror some or all of its extents on the layer just below it.) In this case, an LVM on the host configured the data object as one mirror, meaning that data blocks for this object are written to one place. The LVM also configured the object as a striped mirror with four columns and a stripe size of 32 blocks. Each column maps to a different physical disk.

```

Absolute Path      : /fs_test1/file1
Resolved Object Type : SunOS UFS File
Resolved Object Size : 1500b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 4
File System Mount Point : /fs_test1
File System Device Name : /dev/vx/dsk/fsvg/Striped

```

Number of Mirrors for object (1):

```

{
1) Mirror Configuration

Mirror Stripe Columns : 4
Mirror Stripe Size    : 32b

```

The four Symmetrix device names (Dev 00A0, 00A4, 00A8, and 0009) correspond to host physical device names (PPdevName): /dev/vx/rdmp/c4t2d0s2, /c4t2d1s2, /c4t2d2s2, and /c5t0d1s2. The Symmetrix devices were configured arbitrarily as three meta devices and one simple Symmetrix device to illustrate that physical disks do not have to be the same configuration type, and may actually be reconfigured as storage needs expand. The “Dev Offset” value here is the same as the “PPdevName Offset” value because the disks are not partitioned. When disks are partitioned, these offsets will be different. Column “Array” provides the ID of the storage array on which the disk is located, in this case, Symmetrix units 00541 and 03122.

Mirror Physical Extents (94):

```
{
```

Size	Array	Dev	Offset	PPdevName	Offset	
16b	00541	00A0	3312b	/dev/vx/rdmp/c4t2d0s2	3312b	(S)
16b	00541	00A4	3296b	/dev/vx/rdmp/c4t2d1s2	3296b	(S)
16b	00541	00A4	3312b	/dev/vx/rdmp/c4t2d1s2	3312b	(S)
16b	00541	00A8	3296b	/dev/vx/rdmp/c4t2d2s2	3296b	(S)
16b	00541	00A8	3376b	/dev/vx/rdmp/c4t2d2s2	3376b	(S)
16b	03122	0009	3312b	/dev/vx/rdmp/c5t0d1s2	3312b	(S)
16b	00541	00A0	3328b	/dev/vx/rdmp/c4t2d0s2	3328b	(S)
16b	00541	00A4	3328b	/dev/vx/rdmp/c4t2d1s2	3328b	(S)
16b	00541	00A4	3424b	/dev/vx/rdmp/c4t2d1s2	3424b	(S)
16b	00541	00A4	3440b	/dev/vx/rdmp/c4t2d1s2	3440b	(S)
16b	03122	0009	28800b	/dev/vx/rdmp/c5t0d1s2	28800b	(S)
16b	03122	0009	28816b	/dev/vx/rdmp/c5t0d1s2	28816b	(S)
16b	00541	00A0	28832b	/dev/vx/rdmp/c4t2d0s2	28832b	(S)
16b	00541	00A0	28848b	/dev/vx/rdmp/c4t2d0s2	28848b	(S)
12b	00541	00A4	28832b	/dev/vx/rdmp/c4t2d1s2	28832b	(S)

```
}
```

Mirror Physical Devices (4):

```
{
```

Array	Dev	PPdevName	PdevName	
00541	00A0	/dev/vx/rdmp/c4t2d0s2	/dev/vx/rdmp/c4t2d0s2	(M)
00541	00A4	/dev/vx/rdmp/c4t2d1s2	/dev/vx/rdmp/c4t2d1s2	(M)
00541	00A8	/dev/vx/rdmp/c4t2d2s2	/dev/vx/rdmp/c4t2d2s2	(M)
03122	0009	/dev/vx/rdmp/c5t0d1s2	/dev/vx/rdmp/c5t0d1s2	(S)

```
}
```

Legend for the Attribute of Devices:

- (C): CLARiiON Device.
- (S): Symmetrix Device.
- (M): Symmetrix Device Meta Head.
- (m): Symmetrix Device Meta member.

The `symrslv dir` command examines a host directory named `/usr` within the root file system (`/`). Solaris, at installation time, mounted the root file system on slice 0 (`s0`) of an internal disk. The size of the directory is two blocks (`2b`), and its single collapsed extent begins at offset `936896b` of `/dev/rdisk/c0t0d0s0`. By default, the directory's extents are collapsed. Because SRM is examining a local device here rather than a Symmetrix or CLARiiON device, SRM cannot return Array, Dev, and Dev Offset. Therefore, these columns are not applicable (N/A) with this output display.

```
# symrslv dir /usr -blocks
```

```
Absolute Path           : /usr
Resolved Object Type    : Directory
Resolved Object Size    : 2b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 1
File System Mount Point : /
File System Device Name : /dev/dsk/c0t0d0s0
```

```
Number of Mirrors for object (1):
```

```
{
```

```
1) Mirror Configuration
```

```
  Mirror Physical Extents (1):
```

```
  {
```

```

-----
              Size  Array Dev      Offset PPdevName              Offset
-----
              2b    N/A   N/A      N/A /dev/rdisk/c0t0d0s0      936896b
  }
```

```
  Mirror Physical Devices (1):
```

```
  {
```

```

-----
Array Dev  PPdevName              PdevName
-----
N/A   N/A   /dev/rdisk/c0t0d0s0    /dev/rdisk/c0t0d0s2
  }
```

```
}
```

Legend for the Attribute of Devices:

```
(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.
```

The `symrslv file` command expands the extents of the file named `file1` and displays its 94 physical extents in blocks (1500 total). Unlike the previous `symhostfs show` command, `symrslv` focuses strictly on the extent data at the Symmetrix device layer. The `symrslv` command drills down directly to the Symmetrix devices (Dev) on which the file is stored and locates the extents there.

```
# symrslv file /fs_test1/file1 -expand -blocks
```

```
Absolute Path           : /fs_test1/file1
Resolved Object Type     : SunOS UFS File
Resolved Object Size     : 1500b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 4
File System Mount Point  : /fs_test1
File System Device Name  : /dev/vx/dsk/fsvg/Striped
```

```
Number of Mirrors for object (1):
```

```
{
```

```
1) Mirror Configuration
```

```
Mirror Stripe Columns   : 4
Mirror Stripe Size      : 32b
```

```
Mirror Physical Extents (94):
```

```
{
```

Size	Array	Dev	Offset	PPdevName	Offset
16b	00541	00A0	3312b	/dev/vx/rdmp/c4t2d0s2	3312b (S)
16b	00541	00A4	3296b	/dev/vx/rdmp/c4t2d1s2	3296b (S)
16b	00541	00A4	3312b	/dev/vx/rdmp/c4t2d1s2	3312b (S)
16b	00541	00A8	3296b	/dev/vx/rdmp/c4t2d2s2	3296b (S)
16b	00541	00A8	3376b	/dev/vx/rdmp/c4t2d2s2	3376b (S)
16b	03122	0009	3312b	/dev/vx/rdmp/c5t0d1s2	3312b (S)
16b	00541	00A0	3328b	/dev/vx/rdmp/c4t2d0s2	3328b (S)
16b	00541	00A4	3328b	/dev/vx/rdmp/c4t2d1s2	3328b (S)
16b	00541	00A4	3424b	/dev/vx/rdmp/c4t2d1s2	3424b (S)
16b	00541	00A4	3440b	/dev/vx/rdmp/c4t2d1s2	3440b (S)

16b	03122	0009	28800b	/dev/vx/rdmp/c5t0d1s2	28800b (S)
16b	03122	0009	28816b	/dev/vx/rdmp/c5t0d1s2	28816b (S)
16b	00541	00A0	28832b	/dev/vx/rdmp/c4t2d0s2	28832b (S)
16b	00541	00A0	28848b	/dev/vx/rdmp/c4t2d0s2	28848b (S)
12b	00541	00A4	28832b	/dev/vx/rdmp/c4t2d1s2	28832b (S)

```
}
```

```
Mirror Physical Devices (4):
```

```
{
```

Array	Dev	PPdevName	PdevName
00541	00A0	/dev/vx/rdmp/c4t2d0s2	/dev/vx/rdmp/c4t2d0s2 (M)
00541	00A4	/dev/vx/rdmp/c4t2d1s2	/dev/vx/rdmp/c4t2d1s2 (M)
00541	00A8	/dev/vx/rdmp/c4t2d2s2	/dev/vx/rdmp/c4t2d2s2 (M)

```

03122 0009 /dev/vx/rdmp/c5t0d1s2 /dev/vx/rdmp/c5t0d1s2 (S)
}
}

```

Legend for the Attribute of Devices:

(C): CLARiiON Device.
 (S): Symmetrix Device.
 (M): Symmetrix Device Meta Head.
 (m): Symmetrix Device Meta member.

The `symrslv file` command with the `-collapse` option *logically* collapses the file's extent data, which means that the file can be reconstructed if a subsequent expand operation is performed. Extents whose size is now greater than 16b were formed from contiguous extents that were collapsed (for example, a 32b extent represents two contiguous 16b extents that were collapsed to form one extent).

```
# symrslv file /fs_test1/file1 -collapse -blocks
```

```

Absolute Path           : /fs_test1/file1
Resolved Object Type    : SunOS UFS File
Resolved Object Size    : 1500b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 4
File System Mount Point : /fs_test1
File System Device Name : /dev/vx/dsk/fsvg/Striped

```

Number of Mirrors for object (1):

```
{
1) Mirror Configuration
```

```

Mirror Stripe Columns : 4
Mirror Stripe Size    : 32b

```

Mirror Physical Extents (27):

```
{
-----
```

Size	Array	Dev	Offset	PPdevName	Offset	
16b	00541	00A0	3312b	/dev/vx/rdmp/c4t2d0s2	3312b	(S)
32b	00541	00A4	3296b	/dev/vx/rdmp/c4t2d1s2	3296b	(S)
16b	00541	00A8	3296b	/dev/vx/rdmp/c4t2d2s2	3296b	(S)
16b	00541	00A8	3376b	/dev/vx/rdmp/c4t2d2s2	3376b	(S)
16b	03122	0009	3312b	/dev/vx/rdmp/c5t0d1s2	3312b	(S)
16b	00541	00A0	3328b	/dev/vx/rdmp/c4t2d0s2	3328b	(S)
16b	00541	00A4	3328b	/dev/vx/rdmp/c4t2d1s2	3328b	(S)
32b	00541	00A4	3424b	/dev/vx/rdmp/c4t2d1s2	3424b	(S)
16b	00541	00A0	3424b	/dev/vx/rdmp/c4t2d0s2	3424b	(S)
16b	00541	00A0	3472b	/dev/vx/rdmp/c4t2d0s2	3472b	(S)
16b	00541	00A8	27984b	/dev/vx/rdmp/c4t2d2s2	27984b	(S)
32b	00541	00A4	27968b	/dev/vx/rdmp/c4t2d1s2	27968b	(S)
16b	00541	00A8	27968b	/dev/vx/rdmp/c4t2d2s2	27968b	(S)
32b	00541	00A4	28384b	/dev/vx/rdmp/c4t2d1s2	28384b	(S)
16b	00541	00A8	28384b	/dev/vx/rdmp/c4t2d2s2	28384b	(S)
16b	00541	00A0	27968b	/dev/vx/rdmp/c4t2d0s2	27968b	(S)
16b	00541	00A8	28496b	/dev/vx/rdmp/c4t2d2s2	28496b	(S)

```

-----

```

```

        32b 00541 00A4 28512b /dev/vx/rdmp/c4t2d1s2 28512b (S)
        32b 00541 00A0 28544b /dev/vx/rdmp/c4t2d0s2 28544b (S)
        32b 03122 0009 28480b /dev/vx/rdmp/c5t0d1s2 28480b (S)
        16b 00541 00A8 28512b /dev/vx/rdmp/c4t2d2s2 28512b (S)
        16b 00541 00A8 28560b /dev/vx/rdmp/c4t2d2s2 28560b (S)
        32b 03122 0009 28544b /dev/vx/rdmp/c5t0d1s2 28544b (S)
        256b 00541 00A8 28576b /dev/vx/rdmp/c4t2d2s2 28576b (S)
        256b 03122 0009 28576b /dev/vx/rdmp/c5t0d1s2 28576b (S)
        256b 00541 00A0 28608b /dev/vx/rdmp/c4t2d0s2 28608b (S)
        236b 00541 00A4 28608b /dev/vx/rdmp/c4t2d1s2 28608b (S)
    }

```

Mirror Physical Devices (4):

```

{
-----
Array Dev  PPdevName                      PdevName
-----
00541 00A0  /dev/vx/rdmp/c4t2d0s2  /dev/vx/rdmp/c4t2d0s2  (M)
00541 00A4  /dev/vx/rdmp/c4t2d1s2  /dev/vx/rdmp/c4t2d1s2  (M)
00541 00A8  /dev/vx/rdmp/c4t2d2s2  /dev/vx/rdmp/c4t2d2s2  (M)
03122 0009  /dev/vx/rdmp/c5t0d1s2  /dev/vx/rdmp/c5t0d1s2  (S)
}
}

```

Legend for the Attribute of Devices:

(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.

The `symrslv file` command with the `-phys_collapse` option *physically* collapses the same logically collapsed file from 27 extents to 22 extents, but the file can no longer be reconstructed if a subsequent expand operation is performed. For example, the first 32b extent (Dev 00A0) displayed represents two 16b extents that were not contiguous extents of the file but are contiguous on the disk. Thus, these two extents were not logically collapsible (in the previous command), but they are physically collapsible as long as you do not need to reconstruct the file.

symrslv file /fs_test1/file1 -phys_collapse -blocks

```

Absolute Path          : /fs_test1/file1
Resolved Object Type    : SunOS UFS File
Resolved Object Size    : 1500b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 4
File System Mount Point : /fs_test1
File System Device Name  : /dev/vx/dsk/fsvg/Striped

```

Number of Mirrors for object (1):

```

{
1) Mirror Configuration

Mirror Stripe Columns : 4
Mirror Stripe Size    : 32b

```

Mirror Physical Extents (22):

```
{
```

Size	Array	Dev	Offset	PPdevName	Offset	
32b	00541	00A0	3312b	/dev/vx/rdmp/c4t2d0s2	3312b	(S)
48b	00541	00A4	3296b	/dev/vx/rdmp/c4t2d1s2	3296b	(S)
16b	00541	00A8	3296b	/dev/vx/rdmp/c4t2d2s2	3296b	(S)
16b	00541	00A8	3376b	/dev/vx/rdmp/c4t2d2s2	3376b	(S)
16b	03122	0009	3312b	/dev/vx/rdmp/c5t0d1s2	3312b	(S)
32b	00541	00A4	3424b	/dev/vx/rdmp/c4t2d1s2	3424b	(S)
16b	00541	00A0	3424b	/dev/vx/rdmp/c4t2d0s2	3424b	(S)
16b	00541	00A0	3472b	/dev/vx/rdmp/c4t2d0s2	3472b	(S)
16b	00541	00A8	27984b	/dev/vx/rdmp/c4t2d2s2	27984b	(S)
32b	00541	00A4	27968b	/dev/vx/rdmp/c4t2d1s2	27968b	(S)
16b	00541	00A8	27968b	/dev/vx/rdmp/c4t2d2s2	27968b	(S)
32b	00541	00A4	28384b	/dev/vx/rdmp/c4t2d1s2	28384b	(S)
16b	00541	00A8	28384b	/dev/vx/rdmp/c4t2d2s2	28384b	(S)
16b	00541	00A0	27968b	/dev/vx/rdmp/c4t2d0s2	27968b	(S)
32b	00541	00A8	28496b	/dev/vx/rdmp/c4t2d2s2	28496b	(S)
32b	00541	00A4	28512b	/dev/vx/rdmp/c4t2d1s2	28512b	(S)
32b	00541	00A0	28544b	/dev/vx/rdmp/c4t2d0s2	28544b	(S)
32b	03122	0009	28480b	/dev/vx/rdmp/c5t0d1s2	28480b	(S)
272b	00541	00A8	28560b	/dev/vx/rdmp/c4t2d2s2	28560b	(S)
288b	03122	0009	28544b	/dev/vx/rdmp/c5t0d1s2	28544b	(S)
256b	00541	00A0	28608b	/dev/vx/rdmp/c4t2d0s2	28608b	(S)
236b	00541	00A4	28608b	/dev/vx/rdmp/c4t2d1s2	28608b	(S)

```
}
```

Mirror Physical Devices (4):

```
{
```

Array	Dev	PPdevName	PdevName	
00541	00A0	/dev/vx/rdmp/c4t2d0s2	/dev/vx/rdmp/c4t2d0s2	(M)
00541	00A4	/dev/vx/rdmp/c4t2d1s2	/dev/vx/rdmp/c4t2d1s2	(M)
00541	00A8	/dev/vx/rdmp/c4t2d2s2	/dev/vx/rdmp/c4t2d2s2	(M)
03122	0009	/dev/vx/rdmp/c5t0d1s2	/dev/vx/rdmp/c5t0d1s2	(S)

```
}
```

Legend for the Attribute of Devices:

(C): CLARiion Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.

The `symrslv fs` command with the `-no_extents` option displays the attributes of a file system (/fs_test1) but omits any extent data.

```
# symrslv fs /fs_test1 -no_extents
```

```

Absolute Path           : /fs_test1
Resolved Object Type    : File System
Resolved Object Size    : 4430m
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 5
File System Mount Point : /fs_test1
File System Device Name : /dev/vx/dsk/fsvg/Striped

```

```
Number of Mirrors for object (1):
```

```
{
1) Mirror Configuration
```

```

Mirror Stripe Columns    : 4
Mirror Stripe Size       : 16k

```

```
Mirror Physical Extents (0):
```

```
Mirror Physical Devices (5):
```

```
{
```

```
-----
```

Array	Dev	PPdevName	PdevName	
00541	00A0	/dev/vx/rdmp/c4t2d0s2	/dev/vx/rdmp/c4t2d0s2	(M)
00541	00A4	/dev/vx/rdmp/c4t2d1s2	/dev/vx/rdmp/c4t2d1s2	(M)
00541	00A8	/dev/vx/rdmp/c4t2d2s2	/dev/vx/rdmp/c4t2d2s2	(M)
03122	0009	/dev/vx/rdmp/c5t0d1s2	/dev/vx/rdmp/c5t0d1s2	(S)
03122	000A	/dev/vx/rdmp/c5t0d2s2	/dev/vx/rdmp/c5t0d2s2	(S)

```
}
```

```
}
```

Legend for the Attribute of Devices:

```

(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.

```

The `symrslv lv striped` command examines a logical volume named `striped` within a volume group (`-g`) named `fsvg`. The `-pdev_extents` option shows information about pdev-level extents only — without displaying any underlying meta device configuration. The `-stripe_column` option adds the stripe column number to the display of physical extents, a useful option when dealing with a striped device. Note that stripe column 3 spans two devices (0009 and 000A) whose combined size equals the column size of each of the other devices. Column-3 devices are located on a different storage unit (03122) from the others.

```
# symrslv lv striped -pdev_extents -stripe_column -g fsvg -blocks
```

```

Absolute Path           : /dev/vx/rdsk/fsvg/Striped
Resolved Object Type    : SunOS VxVM Logical Volume
Resolved Object Size    : 9216000b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 5

```

```
Number of Mirrors for object (1):
```

```
{
```

```
1) Mirror Configuration
```

```

Mirror Stripe Columns    : 4
Mirror Stripe Size       : 32b

```

```
Mirror Physical Extents (5):
```

```
{
```

```
-----
```

Stripe Column	Size	Array Dev	Offset	PPdevName	Offset
0	2304000b	00541 00A0	2880b	/dev/vx/rdmp/c4t2d0s2	2880b (S)
1	2304000b	00541 00A4	2880b	/dev/vx/rdmp/c4t2d1s2	2880b (S)
2	2304000b	00541 00A8	2880b	/dev/vx/rdmp/c4t2d2s2	2880b (S)
3	1961280b	03122 0009	2880b	/dev/vx/rdmp/c5t0d1s2	2880b (S)
3	342720b	03122 000A	3840b	/dev/vx/rdmp/c5t0d2s2	3840b (S)

```
}
```

```
Mirror Physical Devices (5):
```

```
{
```

```
-----
```

Array Dev	PPdevName	PdevName
00541 00A0	/dev/vx/rdmp/c4t2d0s2	/dev/vx/rdmp/c4t2d0s2 (M)
00541 00A4	/dev/vx/rdmp/c4t2d1s2	/dev/vx/rdmp/c4t2d1s2 (M)
00541 00A8	/dev/vx/rdmp/c4t2d2s2	/dev/vx/rdmp/c4t2d2s2 (M)
03122 0009	/dev/vx/rdmp/c5t0d1s2	/dev/vx/rdmp/c5t0d1s2 (S)
03122 000A	/dev/vx/rdmp/c5t0d2s2	/dev/vx/rdmp/c5t0d2s2 (S)

```
}
```

```
}
```

Legend for the Attribute of Devices:

```

(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.

```

The `symrslv lv striped -g fsvg -blocks` command examines the same logical volume (named striped) as the previous command, but this time displays (by default) the underlying meta device configuration for each extent. An (M) at the end of a row of extent data indicates that this extent is located on a meta device.

```
# symrslv lv striped -g fsvg -blocks
```

```
Absolute Path           : /dev/vx/rdsk/fsvg/Striped
Resolved Object Type     : SunOS VxVM Logical Volume
Resolved Object Size     : 9216000b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 5
```

```
Number of Mirrors for object (1):
```

```
{
```

```
1) Mirror Configuration
```

```
Mirror Stripe Columns   : 4
Mirror Stripe Size      : 32b
```

```
Mirror Physical Extents (5):
```

```
{
```

```
-----
      Size  Array Dev    Offset PPdevName                      Offset
-----
2304000b  00541 00A0    2880b /dev/vx/rdmp/c4t2d0s2      2880b (M)
2304000b  00541 00A4    2880b /dev/vx/rdmp/c4t2d1s2      2880b (M)
2304000b  00541 00A8    2880b /dev/vx/rdmp/c4t2d2s2      2880b (M)
1961280b  03122 0009    2880b /dev/vx/rdmp/c5t0d1s2      2880b (S)
342720b   03122 000A    3840b /dev/vx/rdmp/c5t0d2s2      3840b (S)
}
```

```
Mirror Physical Devices (5):
```

```
{
```

```
-----
Array Dev  PPdevName                      PdevName
-----
00541 00A0  /dev/vx/rdmp/c4t2d0s2  /dev/vx/rdmp/c4t2d0s2  (M)
00541 00A4  /dev/vx/rdmp/c4t2d1s2  /dev/vx/rdmp/c4t2d1s2  (M)
00541 00A8  /dev/vx/rdmp/c4t2d2s2  /dev/vx/rdmp/c4t2d2s2  (M)
03122 0009  /dev/vx/rdmp/c5t0d1s2  /dev/vx/rdmp/c5t0d1s2  (S)
03122 000A  /dev/vx/rdmp/c5t0d2s2  /dev/vx/rdmp/c5t0d2s2  (S)
}
```

Legend for the Attribute of Devices:

```
(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.
```

The `symrslv pd` command examines extents of one of the physical devices from the previous display. As shown in that display, device `/dev/rdisk/c4t2d0s2` is the meta head (M) of a Symmetrix meta device that includes other meta members (m). This display identifies those meta members as Symmetrix devices 00A1, 00A2, and 00A3.

```
# symrslv pd /dev/rdisk/c4t2d0s2 -blocks
```

```

Absolute Path           : /dev/rdisk/c4t2d0s2
Resolved Object Type    : Physical Device
Resolved Object Size    : 35349120b
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 1

```

```
Number of Mirrors for object (1):
```

```
{
```

```
1) Mirror Configuration
```

```
Mirror Physical Extents (4):
```

```
{
```

```

-----
      Size  Array Dev      Offset PPdevName      Offset
-----
      8837760b  00541 00A0      0b /dev/rdisk/c4t2d0s2      0b (M)
      8837760b  00541 00A1      0b /dev/rdisk/c4t2d0s2    8837760b (m)
      8837760b  00541 00A2      0b /dev/rdisk/c4t2d0s2   17675520b (m)
      8835840b  00541 00A3      0b /dev/rdisk/c4t2d0s2   26513280b (m)
}

```

```
Mirror Physical Devices (4):
```

```
{
```

```

-----
Array Dev  PPdevName      PdevName      (M)
-----
00541 00A0  /dev/rdisk/c4t2d0s2  /dev/rdisk/c4t2d0s2  (M)
00541 00A1  /dev/rdisk/c4t2d0s2  /dev/rdisk/c4t2d0s2  (m)
00541 00A2  /dev/rdisk/c4t2d0s2  /dev/rdisk/c4t2d0s2  (m)
00541 00A3  /dev/rdisk/c4t2d0s2  /dev/rdisk/c4t2d0s2  (m)
}

```

```
}
```

Legend for the Attribute of Devices:

(C): CLARiiON Device.

(S): Symmetrix Device.

(M): Symmetrix Device Meta Head.

(m): Symmetrix Device Meta member.

Beginning with Solutions Enabler version 5.1, you can use the `symrslv file` command to display the extents of a Celerra HighRoad MPFS file.

symrslv file /mpfstest/system

```

Absolute Path           : /mpfstest/system
Resolved Object Type    : Celerra HighRoad MPFS File
Resolved Object Size    : 2k
Number of Trailing Bytes : 257
Extent byte offset to data : 0
Number of Physical Devices : 1
File System Mount Point : /mpfstest
File System Device Name  : api242:/nmk_fs

```

Number of Mirrors for object (1):

{

1) Mirror Configuration

Mirror Physical Extents (1):

{

Size	Array	Dev	Offset	PPdevName	Offset
8k	00229	03A3	7881m	/dev/rdisk/c3t1d3s2	7881m (S)

}

Mirror Physical Devices (1):

{

Array	Dev	PPdevName	PdevName
00229	03A3	/dev/rdisk/c3t1d3s2	/dev/rdisk/c3t1d3s2 (S)

}

}

Legend for the Attribute of Devices:

(C): CLARiiON Device.
 (S): Symmetrix Device.
 (M): Symmetrix Device Meta Head.
 (m): Symmetrix Device Meta member.

Example 3: Displaying Volume Groups and Logical Volumes

The hardware setup for this example consists of a Sun Solaris host connected to a Symmetrix storage unit. The SunOS VxVM logical volume manager manages the volume groups and logical volumes.

The `symvg list` command displays logical volume groups defined for this host. The N/A columns are not applicable to a SunOS VxVM logical volume manager (although HP-UX and AIX LVMs use them).

```
# symvg list
```

```
VOLUME GROUPS (SunOS VxVM):
```

Name	State	Attribute	PE Size	Max Devices	Max Volumes	Num Devices	Num Volumes
rootdg	Enabled	N/A	N/A	N/A	N/A	1	0
bcv_fs_lvm_sin*	Enabled	N/A	N/A	N/A	N/A	1	1
fsvg	Enabled	N/A	N/A	N/A	N/A	6	2
nasty	Enabled	N/A	N/A	N/A	N/A	4	7
testvg	Enabled	N/A	N/A	N/A	N/A	3	1

The `symvg show` command displays information on the volume group named `nasty`, including the set of four physical devices that the LVM allocated for this volume group. The display includes each device's physical device name, Symmetrix device name, status (RW), and capacity in megabytes. The physical devices in this display are on Symmetrix unit 00541, and three of the devices are meta (M) devices.

```
# symvg show nasty
```

```
Volume Group Name : nasty
Volume Group Type : SunOS VxVM
```

```
Volume Group State : Enabled
```

```
Volume Group Attributes : N/A
```

```
Group's Physical Extent Size : N/A
```

```
Max Number of Devices in Group : N/A
```

```
Max Number of Volumes in Group : N/A
```

```
Number of Devices in Group : 4
```

```
Number of Volumes in Group : 7
```

```
Physical Device Members (4):
```

```
{
```

PdevName	SymID	Sym Dev	Att.	Sts	Cap (MB)
/dev/vx/rdmp/c4t1d6s2	00541	009B		RW	4315
/dev/vx/rdmp/c4t1d7s2	00541	009C (M)		RW	17261
/dev/vx/rdmp/c4t2d3s2	00541	00AC (M)		RW	17261
/dev/vx/rdmp/c4t2d4s2	00541	00B0 (M)		RW	17258

```
}
```

The `symlv list` command lists the logical volumes of the volume group (nasty) and shows each volume's configuration at the operating system level, state, condition, number of mirrors, and number of logical extents.

```
# symlv -g nasty list
```

```
Volume Group Name: nasty
Volume Group Type: SunOS VxVM
```

Name	Configuration	State	Cond	Num Mirrors	Num Log Extents
MirrorStripe	Striped Mir	Enabled	Sync	2	N/A
RAID5	RAID5	Enabled	Sync	1	N/A
Simple	Simple	Enabled	Sync	1	N/A
StripeMirror	Striped Mir	Enabled	Sync	2	N/A
Striped	Striped	Enabled	Sync	1	N/A
Wide	Striped	Enabled	Sync	1	N/A
Wide2	Striped	Enabled	Sync	1	N/A

The `symlv show` command examines a logical volume named Simple, displaying its mirror extent size in blocks. SRM locates the physical extents on the physical devices where the logical volume resides and displays the mirror configuration at the operating system level. In this case, an LVM on the host configured this logical volume as one mirror, meaning that data blocks for this volume are written to one place. The LVM also configured the volume as a "simple" mirror.

```
# symlv -g nasty -blocks show Simple
```

```
Logical Volume Name       : Simple
Logical Volume Pathname   : /dev/vx/rdisk/nasty/Simple
Volume State              : Enabled
Volume Configuration      : Simple
Volume Condition          : Sync
Volume Attributes         : N/A
```

```
Logical Volume Size       : 204800b
```

```
Number of Logical Volume Mirrors (1):
```

```
{
1) Mirror Configuration   : Simple
   Mirror State           : Enabled
   Mirror Flags           : None
   Mirror Condition       : Sync
```

```
Number of device Partitions : 1
Number of Physical devices  : 1
Number of Storage devices   : 1
```

The display shows one 205440-block extent on Symmetrix device 009B (partitioned physical device name /dev/vx/rdmp/c4t1d6s2). The condition of the extent is Sync (all physical extents in all mirrors of the logical volume are synchronized). Other possible extent conditions are listed in Table 3.

```
Mirror Physical Extents (1):
{
-----
Condition      Size  Array Dev    Offset PPdevName          Offset
-----
Sync           205440b  00541 009B  7331520b /dev/vx/rdmp/c4t1d6s2  7331520b
}

Mirror Physical Devices (1):
{
-----
Array Dev    PPdevName          PdevName
-----
00541 009B   /dev/vx/rdmp/c4t1d6s2  /dev/vx/rdmp/c4t1d6s2      (S)
}
}
```

Legend for the Attribute of Devices:

(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.

Table 3. Extent Conditions

Extent Condition	Description
Stale	The logical volume contains a mirror with some physical extents that are not consistent with corresponding extents in another mirror.
Sync	All physical extents in all mirrors of the logical volume are synchronized.
Degraded RAID 5	For VxVM, DiskADM, and LDM only. One of the devices in the RAID 5 set is unavailable. Inaccessible data is being generated using the parity check formula.
Degraded Volume	A mirror is off line.
Empty	There is no data on the volume.
Unusable RAID 5	For VxVM, DiskADM, and LDM only. The RAID 5 volume is not usable. This condition implies that access to two subdisks has failed.
Syncing	A member of the volume is being regenerated.
NeedSync	A mirror is stale.
NoDev	A physical device is unavailable.
Degraded	In a simple or striped logical volume, there is at least one good copy of data available.
Initializing	The logical volume is initializing.

This `symlv show` command displays a logical volume named `MirrorStripe`, which the LVM has configured as striped and mirrored (Striped Mir) with two logical volume mirrors.

```
# symlv -g nasty -blocks -collapse show MirrorStripe
```

```
Logical Volume Name      : MirrorStripe
Logical Volume Pathname  : /dev/vx/rdsk/nasty/MirrorStripe
Volume State             : Enabled
Volume Configuration    : Striped Mir
Volume Condition        : Sync
Volume Attributes       : DRL enabled
```

```
Logical Volume Size      : 204800b
```

```
Number of Logical Volume Mirrors (2):
```

This part of the display describes one (1) of the logical volume mirrors. This mirror maps to two physical devices (`/dev/vx/rdmp/c4t1d6s2` and `/dev/vx/rdmp/c4t1d7s2`), which are configured at the storage level to two storage devices (Symmetrix devices 009B and 009D). The extent marked by an (m) is located on a device that is a member of a Symmetrix meta device.

```
{
1) Mirror Configuration : Striped
   Mirror State         : Enabled
   Mirror Flags         : None
   Mirror Condition     : Sync
   Number of Striped Devices : 2
   Stripe Size          : 128b
```

```
Number of device Partitions : 2
Number of Physical devices  : 2
Number of Storage devices   : 2
```

```
Mirror Physical Extents (2):
```

```
{
-----
Condition      Size  Array Dev  Offset PPdevName      Offset
-----
(m) Sync        102720b  00541 009B  7228800b /dev/vx/rdmp/c4t1d6s2  7228800b
   Sync        102720b  00541 009D  2992320b /dev/vx/rdmp/c4t1d7s2  11830080b
}
```

```
Mirror Physical Devices (2):
```

```
{
-----
Array Dev  PPdevName      PdevName      (S)
-----
00541 009B  /dev/vx/rdmp/c4t1d6s2  /dev/vx/rdmp/c4t1d6s2
00541 009D  /dev/vx/rdmp/c4t1d7s2  /dev/vx/rdmp/c4t1d7s2  (m)
}
```

This part of the display describes the second (2) mirror. This mirror maps to two physical devices (/dev/vx/rdmp/c4t2d3s2 and /dev/vx/rdmp/c4t2d4s2) that are configured at the storage level as Symmetrix meta devices, each with four meta members. Physical extents exist on all eight of the storage devices.

```

2) Mirror Configuration      : Striped
   Mirror State              : Enabled
   Mirror Flags              : Striped META
   Mirror Condition          : Sync
   Number of Striped Devices : 2
   Stripe Size               : 128b

```

```

Number of device Partitions : 2
Number of Physical devices  : 2
Number of Storage devices   : 8

```

Mirror Physical Extents (10):

```

{
-----
Condition      Size  Array Dev  Offset PPdevName                      Offset
-----
(m) Sync        960b  00541 00AD 2957760b /dev/vx/rdmp/c4t2d3s2 11830080b
(M) Sync        960b  00541 00B0 2959680b /dev/vx/rdmp/c4t2d4s2 11830080b
(m) Sync       26880b 00541 00AE 2956800b /dev/vx/rdmp/c4t2d3s2 11831040b
(m) Sync       26880b 00541 00B1 2956800b /dev/vx/rdmp/c4t2d4s2 11831040b
(m) Sync       24960b 00541 00AF 2956800b /dev/vx/rdmp/c4t2d3s2 11832960b
(m) Sync       26880b 00541 00B2 2956800b /dev/vx/rdmp/c4t2d4s2 11834880b
(M) Sync       24960b 00541 00AC 2958720b /dev/vx/rdmp/c4t2d3s2 11834880b
(m) Sync       24960b 00541 00B3 2956800b /dev/vx/rdmp/c4t2d4s2 11838720b
(m) Sync       24960b 00541 00AD 2958720b /dev/vx/rdmp/c4t2d3s2 11836800b
(M) Sync       23040b 00541 00B0 2960640b /dev/vx/rdmp/c4t2d4s2 11842560b
}

```

Mirror Physical Devices (8):

```

{
-----
Array Dev  PPdevName                      PdevName                      (M)
-----
00541 00AC  /dev/vx/rdmp/c4t2d3s2  /dev/vx/rdmp/c4t2d3s2      (M)
00541 00AD  /dev/vx/rdmp/c4t2d3s2  /dev/vx/rdmp/c4t2d3s2      (m)
00541 00AE  /dev/vx/rdmp/c4t2d3s2  /dev/vx/rdmp/c4t2d3s2      (m)
00541 00AF  /dev/vx/rdmp/c4t2d3s2  /dev/vx/rdmp/c4t2d3s2      (m)
00541 00B0  /dev/vx/rdmp/c4t2d4s2  /dev/vx/rdmp/c4t2d4s2      (M)
00541 00B1  /dev/vx/rdmp/c4t2d4s2  /dev/vx/rdmp/c4t2d4s2      (m)
00541 00B2  /dev/vx/rdmp/c4t2d4s2  /dev/vx/rdmp/c4t2d4s2      (m)
00541 00B3  /dev/vx/rdmp/c4t2d4s2  /dev/vx/rdmp/c4t2d4s2      (m)
}
}

```

Legend for the Attribute of Devices:

```

(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.

```

Example 4: Deporting and Importing a Volume Group

Beginning with Solutions Enabler version 5.1, you can deport volume group meta data from a system to persistent storage and import this same volume group information later to another host (for example, a backup server). The hardware setup for this example consists of a Sun Solaris host connected to a Symmetrix storage unit. The SunOS VxVM logical volume manager manages the volume groups and logical volumes.

The `symvg show` command displays information on the volume group named `testvg`, including the set of three physical devices that the LVM allocated for this volume group. The display includes each device's physical device name, Symmetrix device name, status (RW), and capacity in megabytes. The physical devices in this display are on Symmetrix (S) unit 03122.

```
# symvg show testvg
```

```
Volume Group Name : testvg
Volume Group Type : SunOS VxVM
```

```
Volume Group State          : Enabled
```

```
Volume Group Attributes     : N/A
```

```
Group's Physical Extent Size : N/A
```

```
Max Number of Devices in Group : N/A
```

```
Max Number of Volumes in Group : N/A
```

```
Number of Devices in Group   :    3
```

```
Number of Volumes in Group   :    1
```

```
Physical Device Members (3):
```

```
{
-----
PdevName          Array  Dev  Att. Sts      Cap
-----
/dev/vx/rdmp/c5t1d0s2  03122 0028 (S)  RW       960
/dev/vx/rdmp/c5t1d2s2  03122 002A (S)  RW       960
/dev/vx/rdmp/c5t1d7s2  03122 002F (S)  RW       960
}
```

Legend for the Attribute of Devices:

(C): CLARiiON Device.

(S): Symmetrix Device.

(M): Symmetrix Device Meta Head.

(m): Symmetrix Device Meta member.

The `symvg deport` command deports the meta data of volume group `testvg` from the system and stores that information in persistent storage. Some platforms may require that you include the `-mapfile` option and a file name. If so, you need to remember the file name and path for a subsequent import operation.

```
# symvg deport testvg
```

```
The Control Operation Completed Successfully
```

Another `symvg show` command verifies that volume group `testvg` is no longer available in the system.

```
# symvg show testvg
```

Either no volume groups or the specified volume group does not exist

The `symvg import` command imports volume group `testvg` into the system from which you issue the command.

```
# symvg import testvg
```

The Control Operation Completed Successfully

Another `symvg show` command verifies that the volume group was successfully imported and that information about the volume group is available again.

```
# symvg show testvg
```

Volume Group Name : `testvg`

Volume Group Type : `SunOS VxVM`

Volume Group State : `Enabled`

Volume Group Attributes : `N/A`

Group's Physical Extent Size : `N/A`

Max Number of Devices in Group : `N/A`

Max Number of Volumes in Group : `N/A`

Number of Devices in Group : `3`

Number of Volumes in Group : `1`

Physical Device Members (3):

```
{
-----
PdevName                Array  Dev  Att. Sts      Cap
(MB)
-----
/dev/vx/rdmp/c5t1d0s2    03122 0028 (S)  RW        960
/dev/vx/rdmp/c5t1d2s2    03122 002A (S)  RW        960
/dev/vx/rdmp/c5t1d7s2    03122 002F (S)  RW        960
}
```

Legend for the Attribute of Devices:

(C): CLARiiON Device.

(S): Symmetrix Device.

(M): Symmetrix Device Meta Head.

(m): Symmetrix Device Meta member.

Examples: CLARiiON Storage Devices

Example 5: Mapping Files

The hardware setup for this example consists of a Windows 2000 host connected to a CLARiiON storage unit. The files displayed are within a Windows NTFS file system.

The `symrslv file` command expands the extents of the file named `clartest` and displays all 3201 of its physical extents. The ellipsis (.....) indicates where output was omitted for brevity. This command focuses on the extent data at the CLARiiON device layer. The `symrslv` command drills down directly to the CLARiiON devices (Dev) on which the file is stored and locates the extents there.

```
C:\> symrslv file p:\clartest -expand
```

```

Absolute Path           : P:\CLARTEST
Resolved Object Type    : Windows NTFS File
Resolved Object Size    : 200m
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 2
File System Mount Point : P:\
File System Device Name : \\.\P:

```

```
Number of Mirrors for object (1):
```

```
{
1) Mirror Configuration
```

```

Mirror Stripe Columns    : 2
Mirror Stripe Size       : 64k

```

```
Mirror Physical Extents (3201):
```

```
{
-----
```

Size	Array	Dev	Offset	PPdevName	Offset
60k	00316	0018	127m	\\.\PHYSICALDRIVE24	127m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)
64k	00316	0017	128m	\\.\PHYSICALDRIVE23	128m (C)
64k	00316	0018	128m	\\.\PHYSICALDRIVE24	128m (C)

```

-----
}
```

```

64k  00316 0017      128m \\.\PHYSICALDRIVE23      128m (C)
64k  00316 0018      128m \\.\PHYSICALDRIVE24      128m (C)

```

```

64k  00316 0018      227m \\.\PHYSICALDRIVE24      227m (C)
64k  00316 0017      227m \\.\PHYSICALDRIVE23      227m (C)
64k  00316 0018      227m \\.\PHYSICALDRIVE24      227m (C)
64k  00316 0017      227m \\.\PHYSICALDRIVE23      227m (C)
64k  00316 0018      227m \\.\PHYSICALDRIVE24      227m (C)
64k  00316 0017      227m \\.\PHYSICALDRIVE23      227m (C)
64k  00316 0018      227m \\.\PHYSICALDRIVE24      227m (C)
64k  00316 0017      227m \\.\PHYSICALDRIVE23      227m (C)
64k  00316 0018      227m \\.\PHYSICALDRIVE24      227m (C)
4k   00316 0018      227m \\.\PHYSICALDRIVE24      227m (C)

```

```

}

```

```

Mirror Physical Devices (2):

```

```

{

```

Array Dev	PPdevName	PdevName	
00316 0018	\\.\PHYSICALDRIVE24	\\.\PHYSICALDRIVE24	(C)
00316 0017	\\.\PHYSICALDRIVE23	\\.\PHYSICALDRIVE23	(C)

```

}

```

```

}

```

Legend for the Attribute of Devices:

```

(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.

```

The `symrslv file` command with the `-collapse` option *logically* collapses the same file's extent data, which means that the file can be reconstructed if a subsequent expand operation is performed. Extents whose size is now 100 megabytes were formed from collapsing contiguous 64-kilobyte extents shown in the previous "expanded" display for the file `clartest`.

```
C:\> symrslv file p:\clartest -collapse
```

```
Absolute Path           : P:\CLARTEST
Resolved Object Type     : Windows NTFS File
Resolved Object Size     : 200m
Number of Trailing Bytes : 0
Extent byte offset to data : 0
Number of Physical Devices : 2
File System Mount Point  : P:\
File System Device Name  : \\.\P:
```

```
Number of Mirrors for object (1):
```

```
{
```

```
1) Mirror Configuration
```

```
Mirror Stripe Columns   : 2
Mirror Stripe Size      : 64k
```

```
Mirror Physical Extents (3):
```

```
{
```

```
-----
```

Size	Array Dev	Offset	PPdevName	Offset
60k	00316 0018	127m	\\.\PHYSICALDRIVE24	127m (C)
100m	00316 0017	128m	\\.\PHYSICALDRIVE23	128m (C)
100m	00316 0018	128m	\\.\PHYSICALDRIVE24	128m (C)

```
}
```

```
Mirror Physical Devices (2):
```

```
{
```

```
-----
```

Array Dev	PPdevName	PdevName
00316 0018	\\.\PHYSICALDRIVE24	\\.\PHYSICALDRIVE24 (C)
00316 0017	\\.\PHYSICALDRIVE23	\\.\PHYSICALDRIVE23 (C)

```
}
```

```
}
```

Legend for the Attribute of Devices:

```
(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.
```

Example 6: Displaying a Logical Volume

The hardware setup for this example consists of a Windows 2000 host connected to a CLARiiON storage unit.

The `symlv show` command examines a logical volume named P: in the volume group clardg, displaying the volume's mirror extent size as 2039 megabytes. SRM locates the physical extents on the physical devices where the logical volume resides and displays the mirror configuration at the operating system level. An LVM on the host configured this logical volume as one "striped" mirror, with data blocks for this volume written to two devices. The display shows two 1020-megabyte extents on CLARiiON devices 0017 and 0018. The extent condition is Sync (see Table 3 for other possible extent conditions).

```
C:\> symlv -g clardg show P:
```

```
Logical Volume Name      : P:
Logical Volume Pathname  : \\.\P:
Volume State             : Enabled
Volume Configuration     : Striped
Volume Condition         : Sync
Volume Attributes       : N/A
```

```
Logical Volume Size      : 2039m
```

```
Number of Logical Volume Mirrors (1):
```

```
{
1) Mirror Configuration : Striped
   Mirror State          : Enabled
   Mirror Flags          : None
   Mirror Condition      : Sync
```

```
Number of Striped Devices : 2
Stripe Size               : 64k
```

```
Number of device Partitions : 2
Number of Physical devices  : 2
Number of Storage devices   : 2
```

```
Mirror Physical Extents (2):
```

```
{
-----
Condition      Size  Array Dev   Offset PPdevName          Offset
-----
(C) Sync       1020m 00316 0017    31k  \\.\PHYSICALDRIVE23    31k
(C) Sync       1020m 00316 0018    31k  \\.\PHYSICALDRIVE24    31k
}
```

```
Mirror Physical Devices (2):
```

```
{
-----
Array Dev   PPdevName          PdevName
-----
00316 0017  \\.\PHYSICALDRIVE23  \\.\PHYSICALDRIVE23  (C)
00316 0018  \\.\PHYSICALDRIVE24  \\.\PHYSICALDRIVE24  (C)
}
```

}

Legend for the Attribute of Devices:

(C): CLARiiON Device.
(S): Symmetrix Device.
(M): Symmetrix Device Meta Head.
(m): Symmetrix Device Meta member.