Storage Provisioning Automation using “Home Grown” Tools

Ken Guest and Sejal Joshi

EMC Proven Professional Knowledge Sharing 2009

Ken Guest
A Large Telecommunications Company
kennethguest@yahoo.com

Sejal Joshi
A Large Telecommunications Company
sejaljoshi@hotmail.com
# Table of Contents

## INTRODUCTION
- Purpose of this Article ................................................................. 4
- Background .................................................................................. 4

## COST SAVING BENEFITS ............................................................... 5
- Impact of Storage to FTE Ratio without Automation Scripts .............. 8
- Impact of Storage Management with Automation Scripts .................. 8

## “LARGE TELECOMMUNICATIONS COMPANY” CASE STUDY .............................................. 9
- Objectives ................................................................................... 9

## REQUIREMENTS ............................................................................ 9
- SAN/Storage Management Server Requirements ................................. 9
- SAN/Storage CLI Software and Documentation .................................. 9
- Software/Tools ............................................................................. 9

## PROCESS ....................................................................................... 9

## STORAGE STANDARDS ................................................................. 10
- Standard Hyper Size and RAID Level .............................................. 10
- Number of Devices per MetaVolume .............................................. 10
- Number of Splits per Physical Disk ................................................. 11
- Number of Logical Disk Groups ..................................................... 11
- FAN-OUT Ratio ........................................................................... 11
- FAN-IN Ratio ............................................................................. 11
- Dynamic RDF Settings ................................................................ 11
- HBA Port Flag Implementation ...................................................... 11
- Host Striping Methodology ............................................................. 11
- Front End Port Cards ................................................................ 12
- Front End Rule of 17 ................................................................... 12
- DRV’s for Optimizer .................................................................... 12
- Hot Spare and Permanent Sparing .................................................. 12

## SAN STANDARDS ........................................................................ 13
- Single Initiator/Multi Target or Single Initiator/Single Target ................ 13
- WWN vs. Port Zoning .................................................................... 13
- SAN Naming Conventions ............................................................ 13
- ZoneSet Activation ..................................................................... 13
- HBA WWN Updates ................................................................... 13
- ISL Oversubscription ................................................................. 14

## HOW TO WRITE AN AUTOMATION SCRIPT
- Input File Format ......................................................................... 14
- How to Process Variables ............................................................ 15
- Validate Variables ....................................................................... 16
- How to Collect Vendor Storage Information .................................... 17
- Processing Device Information ..................................................... 18
- Generating Command Files/Validation Files .................................... 20
- Command File Verification/Processing .......................................... 21
- Final Validation after Storage Allocation ........................................ 22

## REQUEST TRACKING SYSTEM (SRTS) ............................................................................. 24
- SRTS Application Overview .......................................................... 24
- Key SRTS Contributors Roles and Responsibilities .......................... 25
Disclaimer: The views, processes, or methodologies published in this compilation are those of the authors. They do not necessarily reflect EMC Corporation’s views, processes, or methodologies.
Introduction

**Purpose of this Article**

This article provides detailed information to create and implement customized provisioning tools using vendor-provided command line interface (CLI) software. Storage Resource Management (SRM)-based tools for provisioning work well in organizations with a small storage footprint; however, they do not scale easily in large, diverse, enterprise class, multi-data center environments. This is mainly due to the cost of implementing SRM and the increase in Total Cost of Ownership (TCO) due to the number of servers required to implement the SRM solution.

This customized provisioning solution can be implemented with a small data center footprint at a small upfront cost to your company.

**Background**

*Storage provisioning automation using home-grown tools* is one method to improve efficiency and quickly implement a common, reliable standard in a continuous-growth, multi-vendor storage environment. This method enables teams to work more efficiently with less people and eliminates the need for expensive SRM tools that may not meet all business requirements. In our current economic environment, cost efficiency is extremely important; storage groups are being downsized and budgets are being cut even as storage capacity and fiber switch ports are increasing at a breakneck rate.

In 2008, the Enterprise Storage Operations (ESO) group of a "Large Telecommunications Company" provisioned 6.0 PB of enterprise class SAN-based storage, compared to 4.3 PB provisioned in 2007: *an increase of 140% in one year*. Our current environment is growing at a rate of ~100 TB per week. With this growth rate, SRM tools do not scale to meet our requirements and Service Level Agreements (SLAs). With our custom implementation, we can provision ~50TB of storage across multiple frames in less than 1 hour, while significantly reducing overall rework and human error. Our custom implementation includes design validation, LUN creation/masking, and zoning.
Cost Saving Benefits

This section illustrates the cost saving benefits of using automation scripts vs. SRM GUI tools.

Figure 1 compares the time it takes to provision approximately 28 TB of storage across 5 DMX-4 Frames. With automation, the time is drastically reduced compared to using an SRM GUI tool, due to the ability to multitask and the fact that less manual intervention is required.

Figure 1. Automation vs. SRM Provisioning Time Comparison

Figure 2 on the following page reflects how the 28TB of storage was allocated across 9 hosts and 5 frames. Using our automation scripts, the allocation process took approximately 50 minutes regardless of the SAN Administrator’s skill level. The following example details the steps illustrated in the graph above. With the SRM tool, the same process can take approximately 4 ½ hours, depending on the SAN Administrator’s skill level.
Specifically detailed in Figure 2:

- Total Storage allocated - ~28 TB across 5 DMX-4 frames
- 9 HP-UX Host
- 128 x 180 GB Meta (180GB Meta = 11GB Raid-5 3+1 x 16 Hyper = 64 Spindles)
- 57 x 90 GB Meta (90GB Meta = 11GB Raid-5 3+1 x 8 Hyper = 32 Spindles)
- Devices were mapped across 26 different FA ports
- 64 of the 128 180GB Meta were converted to Raid-5 BCV
- Updated 44 Host Zone Members

In this example, we were able to process several requests together for the same data center and complete ~28TB of in 50 minutes. We noticed that the larger the request, the less time it took to complete vs. using the SRM Tool. For example, provisioning 100GB of storage may take the same amount of time regardless of SRM or automation scripts.

Figure 3 on the following page compares the approximate cost savings per week based on 30 requests and 90 TB of storage for that week.
This graph compares the number of hours it takes to provision the same amount of storage using SRM tools vs. automation scripts. The total number of hours to complete a request varies based on the size, complexity, and number of requests that can be combined. Combining similar requests and processing them at the same time is the real benefit that automation provides.

In Figure 4 on the following page, the graph compares the approximate cost savings per year based on 2000 requests and 6.0 PB of storage.
Figure 4. Cost of Provisioning per Year

The Industry Standard average salary for an experienced SAN/Storage Administrator is approximately $90K per year, excluding benefits and bonuses. Figure 4 represents that at this growth rate, we would need 3 additional FTE per year if we did not implement automation scripts.

**Impact of Storage to FTE Ratio without Automation Scripts**

The average industry standard for the Storage to FTE ratio is 500 to 750 TB. Due to our automation efforts, our team manages approximately 2.1 PB per FTE.

**Impact of Storage Management with Automation Scripts**

With automation, we can spend less time on the provisioning of storage and more on storage management. This allows time to work on consolidation and other optimization projects.
“Large Telecommunications Company” Case Study

Objectives

Our objective was to implement a complete, efficient, low-cost, centralized solution for end-to-end storage provisioning automation. Our environment uses multi-vendor storage/switch platforms. Since SRM tools do not provide an end-to-end solution that includes storage design, storage provisioning, and file system creation, we decided to implement automation scripts that would pull storage design information directly from a ticketing systems, provision the storage, and add zones with little manual intervention. Based on centralized automation, our goal was to guarantee standards across multiple data centers and environments and create a centralized reporting tool.

Requirements

The high-level requirements include the following:

- SAN/Storage Management Server
- SAN/Storage CLI Software and Documentation
- Software/Tools

SAN/Storage Management Server Requirements

- One Server per Data Center
- UNIX/Linux Platform (RedHat Linux, HPUX, Solaris, etc.)
- HBAs for in-band storage array connectivity
- IP Connectivity for out-of-band connectivity to switches
- ~10 GB of NAS Storage mountable on all management servers

SAN/Storage CLI Software and Documentation

Vendor-specific CLI documentation: (EMC Solutions Enabler CLI Guide, Cisco CLI, Brocade CLI)

Software/Tools

- Programming Language of choice (Perl, Java, C, C++, PHP)
- Install all required dependencies (tcl/tk, expect, gcc, ssh, rsync, etc)

Process
Before implementing a working solution to implement storage automation provisioning using home-grown scripts, we identified the key piece that enables the overall storage automation process to work: a *centralized ticketing system*. This system tracks a storage request throughout its entire life cycle. The centralized ticketing system is a workflow tool that tracks approvals and design information, provides the feed to the automation scripts, and provides information to the System Administrator to build the file systems based on the design.

We chose Perl running on a UNIX system and decided we would also take advantage of Korn Shell scripts to complete certain tasks. Perl was the most flexible language for us to accomplish our team’s objectives. (See Appendix B: Perl Output) We also built the Perl binaries from scratch, so any needed Perl modules could be easily added. Although this might lead to some additional challenges, we determined it was worth the initial effort.

Next, we specified the detailed standards we would follow. We categorized the standards into Storage and SAN.

**Storage Standards**

We identified the storage standards detailed in the following sections.

**Standard Hyper Size and RAID Level**

This standard determines how devices are laid out across the frame. We use 1 GB and 11 GB RAID-5 3+1 hyper. All hyper devices are placed in the free pool. *Decide data layout on spindle (outer edge of spindle is usually faster). We keep a 1 GB hyper on the outage edge of the spindle, normally for database re-do logs. We use a 300 GB 15K Drives with 85 splits on each spindle.

**Number of Devices per MetaVolume**

This standard determines how many hypers we can group together to create a single device. We normally create a 2-, 4-, 8-, or 16-way single striped Meta device. This limits the number of LUNs presented to the host and the amount of data spread across multiple spindles in the back end.
Number of Splits per Physical Disk

This standard determines how we configure the physical spindle.

Number of Logical Disk Groups

This standard determines how we group the drives. If you want to separate production and local replication loads, you can create logical disk groups in the BIN and pick devices from respective logical groups during provisioning. Ask yourself, do you need to keep production and BCV devices separate to avoid performance bottlenecks?

FAN-OUT Ratio

This standard represents the number of host HBAs that we will connect to a single Storage Array Port. (Check with the storage vendor for the supported number).

FAN-IN Ratio

This standard represents the number of Storage Array Ports we can connect one HBA to. (Check with the storage vendor for the supported number).

Dynamic RDF Settings

This standard enables the Dynamic RDF flag on devices.

HBA Port Flag Implementation

This standard allows us to set OS-specific flags per initiator, versus a global setting on the Front End Adapter (FA).

Host Striping Methodology

This standard determines how the System Administrators (SAs) set up the volumes after allocations. It is extremely important to work closely with the SAs; otherwise, performance issues could arise. Striped volumes provide more queues to push data from the host. If sar or iostat disk performance data is reporting high disk queuing (wait time), you can resolve it by wide striping and spreading the load across more spindles. Stripe depth is very important as well: 128K for OLTP type workload and 256K for batch job type workload.
**Front End Port Cards**

This standard determines the settings for each FA port. You can either set OS-specific flags on each FA port or enable/disable individual flags using the HBA port flag during masking. For more flexibility, you can use common flags (required by each OS) at each front-end port and OS-specific flags using HBA port flag. For e.g. you can set Common_Serial_Number (C) and Init_Point_to_Point (PP) flags at each front-end port level. Then set Volume_Set_Address (V), SCSI_Support1 (OS2007), SPC2_Protocol_Version (SPC2) flag can be set using HBA port flag during masking. You can enable or disable particular flag using HBA port flag. Be careful when you disable or enable flags using HBA port flag.

**Front End Rule of 17**

This standard determines how cables are connected in a dual/mirrored fabric environment. It helps minimize a single point of failure. In the Symmetrix® DMX matrix architecture, it is unnecessary to maintain the rule of 17; however, using this rule helps troubleshoot host path issues quickly.

**DRV for Optimizer**

This standard pertains to the Optimizer, which uses DRV devices for swapping. The number of recommended DRVs for Optimizer is 8 per each device size.

**Hot Spare and Permanent Sparing**

This standard requires that we provide adequate spares to ensure that any failing drive has a permanent spare to fail over to.

**Note:** Drives holding the DMX Power Vault do not participate in Permanent Sparing.

EMC’s standard is to allocate 2 spares per 100 physical drives. Based on the new sparing rules with 5772 code, the following spares are recommended:

<table>
<thead>
<tr>
<th>Drive Size</th>
<th>Number of Drives</th>
<th>Recommended Spares</th>
</tr>
</thead>
<tbody>
<tr>
<td>300GB</td>
<td>960</td>
<td>20</td>
</tr>
</tbody>
</table>
**SAN Standards**

We identified the SAN standards detailed in the following sections.

**Single Initiator/Multi Target or Single Initiator/Single Target**

The initiator is the HBA, and the target(s) are storage ports or tape drives. This standard helps determine the zoning relationship between the host and storage frame. Implementing single-initiator, single-target zoning allows us to isolate traffic for each port. In a big environment, we recommend single initiator/multi target zoning methodology for ease of management.

**WWN vs. Port Zoning**

Port zoning utilizes physical ports, and WWN zoning uses name servers in the switches to either allow or block access to particular World Wide Names (WWNs) in the fabric. The ability to re-cable the fabric without having to recreate the zone information is a major advantage of WWN zoning. It is better to use one of these methodologies in fabric for zoning for ease of management.

**SAN Naming Conventions**

Set up a standard naming convention for host/storage aliases, zones, and zonesets. This standard helps identify host and storage frame ports quickly in a larger fabric.

**ZoneSet Activation**

This standard determines how the zoneset is activated after it is changed. During zoneset activation, each changed zone member gets a Registered State Change Notification (RSCN). In a larger fabric, we recommend performing zoneset activation during the maintenance window. Document the process to add/update aliases/zones and add/remove zones from a zone set.

**HBA WWN Updates**

For WWN zoning, this standard determines the process for replacing HBAs in an environment.
ISL Oversubscription

The ISL oversubscription standard ratio is the ratio of the bandwidth needed by the HBAs to the bandwidth available on the ISLs. You should also consider locality when targets and initiators are attached to the fabric. A set of initiators accessing a particular target should be connected to the same switch if possible, or to the same quad or octet to make it even better. Following this standard eliminates any oversubscription or link contention issues and delivers high throughput.

How to Write an Automation Script

The following sections detail how to write an automation script.

Input File Format

All automation scripts start at this point. First, determine whether detailed information about the initial data is passed to script by an input file or pulled directly from an application using XML. Then, the automation script development can begin the finalization stage. We chose to use a pipe (|) separated file instead of the more common tab or comma separated file (CSV). (In our environment, the automation script can handle it as either an input file or XML call.) The pipe-separate file provides us the most flexibility in handling special cases that occasionally arise.

Consider the following:

- Input file format (CSV, XML, other)
- Header naming convention
- Required fields

Here is a sample input file:

| sr|Number|itemNo|device_item_no|cluster|complete|designArrayPort|designHyperSize|designLunQty|designMetaStripWidth|designNotes|designStorageFrame|fileSystemName|floorLocator|hostPlatFormType|newConnection|serverName|serverType|snapShot|storageAmount|storageType|volumeName|
| 12965|1|1|1||||10|AAb/7Aa|1|2|4|42119|1|vg31j|db||HPv3|Nhbpgf2|PROD|N8|EMC|vg31j|db|
How to Process Variables

The following code creates variables based on the input file headers. This code allows the input file to be updated without impacting the entire script. It is important to remember that variables explicitly used in the script are validated and/or processed while all others are not used. This allows for future enhancements to be added to the input file without breaking the automation script. Each line should be processed individually before moving to the next line item. With Perl, you can use %hashes or @arrays to store the data.

Consider the following:

- How to parse/process input file
- How to create name and value relationship
- Determine required fields

Here is a snippet of code:

```perl
foreach $ssts_lineno (@array_ssta){
  chomp $ssts_lineno;

  # generates Variable names based on column headers from file or perl input script
  if ( $field_cnt == 0 )
    foreach $str_field (split(/V/, $ssts_lineno))
      chomp $str_field;
      push(@list_of_fields,$str_field);
  }
  $field_cnt++;
  next;
}

$field_index=0;
next if $ssts_lineno =~ /^#$/;
# Associates the variables names to the values, based on position of header and value.
foreach $str_line (split(/A/, $ssts_lineno)){
  $str_field = @list_of_fields[$field_index];
  $str_field = $str_line;
  $field_index++;
}
```
Validate Variables

At this point, the input file has been converted into name value pairs, and validation can begin. Each key variable should be validated before moving to the next phase. This is where the enforcement of standards begins. During this phase, we check cluster settings and BCV devices; confirm that host platform and FA settings are compatible, etc.

Consider the following:

- Validate required fields
- How to handle exceptions

Here is a snippet of code:

```perl
if (!$complete =~ /Y/) {
    trace("SRTS $sLine Item $itemNo already Completed... Skipping.");
    next;
}
if (!$designedHyperSize eq "") {
    trace("nnERROR..... Required Field designedHyperSize is empty...");
    exit 2;
}
if (!$designedLunQty eq "") {
    trace("nnERROR..... Required Field designedLunQty is empty...");
    exit 2;
}
if (!$designedLunQty =~ A-/) {
    print_color("red", "nnERROR..... designedLunQty Field Contains Dashes so Please Split this Line Item into multiple lines based on the number of dashes...");
    print_color("reset");
    exit 2;
}
trace("Validating MetaStripWidth ...");
if (!$designedMetaStripWidth eq "") {
    if ($hash_meta_stripped->{$designedMetaStripWidth}) {
        trace("nnERROR..... Required Field designedMetaStripWidth is empty...");
        if (!$designedMetaStripWidth eq "");
        $dsw =~ s/\//g;
        trace("nnERROR..... Invalid MetaStripWidth found on Line Item: $itemNo. Check SRTS Design. Supported Values=> $dsw ...");
    }
    exit 2;
}
```
How to Collect Vendor Storage Information

After you have validated all key variables, you can move on to the next step. This is where you gather information about the storage frame, using the EMC Solutions Enabler (SE) CLI or any other vendor’s CLI. Store this information in memory so it can be referenced during processing (@ array or % hashes, if using Perl). The sample code on the following page shows how to make the SE CLI call and store the information. Follow a similar process for each command that needs to be run against the frame.

Consider the following:

- What SE CLI commands to execute (symdev, symcfg, symmaskdb)
- How to store the output for the SE CLI output
- How to loop through each item without multiple calls to SE CLI commands
- Create sub functions for items that may be called multiple times

Here is a snippet of code:

```perl
# Information about the Devices in the free Pool will be collected and returned:
get_symdev_noport($storageFrame, $dg, $diskgroup);

sub get_symdev_noport
  
  my $storageFrame = $_[0];
  my $dg = $_[1];
  my $diskgroup = $_[2];
  if (!$hash_check_sym_noport($storageFrame, $dg) eq $storageFrame)
    trace("Using Cached SYM $storageFrame -noport addresses\n");
    return;

  $hash_check_sym_noport($storageFrame, $dg) = $storageFrame;
  trace("Gathering SYM $storageFrame available devices\n");
  debug(SYM_NOPORT_CMD /usr/symcli/bin/symdev -sid $storageFrame -noport list

  @array_symdev = `$(sudo_cmd)/usr/symcli/bin/symdev -sid $storageFrame -noport list
  
  foreach $str_list (@array_symdev){
    chomp $str_list;
    $str_list =~ s/Not Visible/NotVisible/g;
    $str_list =~ s/2-Way Mfr2-WayMfr/g;
    $str_list =~ s/As+/A/g;
    if ($str_list =~ /RD/FSMSIBCV|DR|Unprotected/) { next; };
    ($sym_dev, $spath, $assign_dev, $da, $raid, $group, $dev_status, $dev_size) =
    split(\'/\', $str_list);
    #skips Meta Devices found in the Free Pool. Only hypers are used by this script
    next if $dev_size !~ /Ad+/; # Skips all non numeric values
    $hash_sym_noport($storageFrame, $dg, $dev_size, $size) = $dev_size;
    $hash_sym_noport($storageFrame, $dg, $dev_size, $device)[$sym_dev] = raid;
    $hash_sym_noport($storageFrame, $dg, $dev_size, $lun) = $sym_dev; if
    $dev_size =~ /Ad+/;
    
```

2009 EMC Proven Professional Knowledge Sharing 17
Processing Device Information

After you have gathered all storage frame information, conduct another round of validation to verify the request can be completed as designed. If you find any issues or exceptions, the script should exit with an error telling you what failed and why.

Assuming all validation checks have passed, process the device information, generate all of the required information, and store it in memory.

At this point, all of the information has been collected, validated and stored in memory.

Consider the following:

- How to access stored data in memory (hashes, arrays, etc.)
- Store associated devices to FAs (mapping, devices, etc)
- Verify that enough devices exist
- Loop through each item to create associations
- Determine how to handle BCV devices
- Clustered servers
- Setting special settings (SCSI3, SPC2, etc)

A snippet of code is illustrated on the following page.
The code snippet appears to be a part of a script or program written in a programming language. The code seems to be handling device management, possibly for a storage or network system. It includes logic for splitting, checking devices, and managing device configurations in a loop. The code is not entirely clear without context, but it likely involves device resolution or configuration tasks.

For example, it includes statements like:

```perl
@arr_devices = split('/', $hash_sym_rootport{$storageFrame}{$dg}{$sizeoofluns}{$lun});
```

This line suggests that the code is splitting a path or string to access device-related information. The script appears to be iterating over devices, checking if they are available, and managing configurations.

There are also constructs like loops and conditional checks, which are typical in scripting to handle repetitive tasks and make decisions based on certain conditions.

Overall, the code snippet is part of a larger script that manages device operations, likely within a storage or network management context.
Generating Command Files/Validation Files

The next step is to generate the metacreate_scsi_mappit, devmask, lunmask, command files and associated validations files. (See Files Created by MAP_LUN_EMC) To generate these files, you need to loop through the information stored in memory. The output format is extremely important here, because SE CLI expects information in a particular format.

Consider the following:

- What files are required?
- How will you validate configuration changes?
- Which files require manual editing?
- What format and how to generate the required files (Perl, ksh, cfg, etc)?
- Is debug information required for these files?
- How to handle servers with multiple HBAs
- How to associate the Host HBA WWN Values to variables
- How will the output files be processed (ksh, Perl, etc)?
- Who is the audience for each of the output files?
- How intelligent will the output scripts be?

Here is a snippet of code:

```perl
print "\n## Generate metacreate_scsi/Mapping File\n";
foreach $frame (keys %array_metacreate){
    print META "\nARRAY_METACREATE\n";
}

foreach $frame (keys %array_map){
    print META "\nARRAY_MAP\n";
    print "\n## Generate Mapping File\n";
}

foreach $frame (keys %array_symmask){
    print META "\nARRAY_SYMMASS\n";
    foreach $host (keys %array_symmask){
        print DEVMASK "\nARRAY_SYMMASS\n";
    }
}
```

---

2009 EMC Proven Professional Knowledge Sharing

20
Command File Verification/Processing

At this point, all processing is complete and it is time for one final review of the files before starting the allocation. It is very important to define how the files are executed and in what sequence. This is where automation can really save time. With EMC, only one configuration change can occur on any given frame at one time. If multiple frames are involved, run each symconfigure and devmask process in parallel (see symconfigure_cmd.ksh)

Consider the following:

- Order of execution
- How to validate files
- What process to resolve errors or issues
- What process can be run in parallel
- What information will be updated manually (i.e. WWNs, See wwn.cfg)
- What other tasks will be completed here (zoning, etc.)

A sample directory structure output:

```
-rwxr-xr-x 1 root  root  462 Feb 14 09:42 sample_file
-rwxr-xr-x 1 root  root  530 Feb 14 09:44
-rwxr-xr-x 1 root  root  446 Feb 14 09:44 symconfigure_cmd.ksh
-rwxr-xr-x 1 root  root  12 Feb 14 09:44 symfile.txt
-rwxr-xr-x 1 root  root  688 Feb 14 09:44 4119.0
-rwxr-xr-x 1 root  root  93612 Feb 14 09:44 srs_logfile.txt
-rwxr-xr-x 1 root  root  215 Feb 14 09:44 wwn.cfg
-rwxr-xr-x 1 root  root  3669 Feb 16 12:21 ...
```

```
4119.0:
total 144
-rwxr-xr-x 2 root  root  530 Feb 14 09:44 ...
-rwxr-xr-x 2 root  root  396 Feb 14 09:44 validation
-rwxr-xr-x 3 root  root  688 Feb 14 09:44 ...
-rwxr-xr-x 1 root  root  419 Feb 14 09:44 4119.xmlpos
-rwxr-xr-x 1 root  root  454 Feb 14 09:44 mstacreate_scsi_mapdata
-rwxr-xr-x 1 root  root  5213 Feb 14 09:44 devconfig
-rwxr-xr-x 1 root  root  650 Feb 14 09:44 unmask
-rwxr-xr-x 1 root  root  108 Feb 14 09:44 mkready
-rwxr-xr-x 1 root  root  318 Feb 14 09:44 zones
```

```
4119.0/validation:
total 96
-rwxr-xr-x 2 root  root  396 Feb 14 09:44 ...
-rwxr-xr-x 3 root  root  688 Feb 14 09:44 ...
-rwxr-xr-x 1 root  root  663 Feb 14 09:44 dev_config
-rwxr-xr-x 1 root  root  374 Feb 14 09:44 fa_verify
-rwxr-xr-x 1 root  root  64 Feb 14 09:44 devmask_verify
-rwxr-xr-x 1 root  root  10 Feb 14 09:44 devices
```
Final Validation after Storage Allocation

This is an extremely important part of the entire allocation process, regardless of whether or not you use automation. It is important to have the SAN Administrator verify that the scripts completed successfully. Execute any and all verification scripts to confirm the request was completed accurately and successfully. Resolve any exception manually, and notify the automation scripter of any issues or suggestions that may improve the entire process/experience.

Consider the following:

- File output
- What information needs to be validated
- How to validate the information

Figure 5 and Figure 6 thereafter illustrate sample validation scripts and output.

Figure 5. Sample Validation Scripts and Output-Part 1

```bash
# cat /dev/config
echo 12E7
symdev -sid 4174 show 12E7 > vol.txt

cat vol.txt | grep 'MegaBytes' ;
cat vol.txt | grep 'Device Status' ;
cat vol.txt | grep 'Device Configuration' ;
cat vol.txt | grep 'SCSI-3 Persistent Reserve.' ;

# /dev/config

12E7
  MegaBytes : 45139
  Device Status : Ready (RW)
  Device Configuration : RAID-5 (Meta Head,
  SCSI-3 Persistent Reserve: Disabled
  Cache Partition: DEFAULT_PARTITION
  Not Visible N/A FA 048.0 RW 000.00 166 N/A
  Not Visible N/A FA 13B.1 RW 000.00 166 N/A

# cat /devmask_verify
/usr/symcli/bin/symmaskdb -sid 4174 -dev 12E7 list assign.

# /devmask_verify

Symmetrix ID: 000190104174

Device Identifier Type Dir.
----------------- -------------
12E7 6001430001724008 FIBRE FA-48.0
      6001430001724008a FIBRE FA-48.0
```
#cat /fa_verify

# HBA WWNNs have been moved to centralized file under base SRTS DIR called wwn.cfg
#/opt/perl5.8/bin/perl

# runs foundations with an isolated WWN
do "./../wwn.cfg"

system("/usr/symcli/bin/symmaskdb -sid 4174 -dir 13B -p 1 -v -wwn \${bhpPhi1aSWWNN1} list database");

system("/usr/symcli/bin/symmaskdb -sid 4174 -dir 48 -p 0 -v -wwn \${bhpPhi1bPWW/N2} list database");

# /fa_verify

Symmetrix ID : 000190104174

Database Type : Type6
Last updated at : 11:13:42 AM on Tue Dec 30, 2008

Director Identification : FA-13B
Director Port : 1

Originator Port wwn : 500143800172400a
Type : Fibre
User-generated Name : bhpPhi1b/500143800172400a
Visibility : No
FCID Lockdown : No
Lun Offset : No
Heterogeneous Host : No
Port Flag Overrides : Yes

Enabled : Volume_Set_Addressing
SPC2_Protocol_Version(SPC2)
SCSI_Support(0S2007)

Disabled : N/A
Dynamic Addressing : No
Authentication State : N/A
Devices : 000F 12E7 12EB
**Request Tracking System (SRTS)**

The centralized Storage Request Tracking System (SRTS) is the most important piece of our automation. This system provides a single point for all storage requests and improves our ability to automate the end-to-end process as much as possible. It also reduces re-work due to manual entry. All storage requests are processed using the same Work Flow, Process, and Procedure. A consistent, repeatable process is vital to storage automation; it simplifies the design, allocation, and validation. SRTS provides a single interface for all parties involved in the provisioning of storage and contains a central reporting tool for management to determine storage usage, growth, and chargeback.

**SRTS Application Overview**

SRTS is a Web-based front-end tool with an Oracle database in the back end. The database allows all involved parties to verify information before the storage is provisioned. Figure 7 shows some partial information about SRTS; specifically, it shows how the storage is designed and which line items are associated with each file system.

![Figure 7. Partial Shot of SRTS Information](image-url)
This screen is where the System Administrator and SAN Administrator validate the design before storage is allocated. The interface provides an easy-to-read format. Information is pulled from the SRTS via a Perl XML script during the preparation phase.

**Key SRTS Contributors Roles and Responsibilities**

Figure 8 illustrates an SRTS Workflow with approval status information.

**Figure 8. SRTS Workflow with Approval Status Information**

![Approval Status Information](image)

**Storage Engineering**

An email notification is sent to Storage Engineering after a requestor has correctly entered and submitted a storage request. The Storage Engineering team reviews the request and changes the storage request's status to “Initial Review.” If the Storage Engineering group identifies any issues with the request, they place it in “PENDING” status.
After the pending issues are resolved and the design is completed, the Storage Engineer alters the storage request status to “DESIGN COMPLETE.” S/he then assigns the storage frame, device sizes and any other pertinent SAN/Storage information.

After storage is allocated and the workflow is changed to “STORAGE ALLOCATED,” the Storage Engineering completes the “Verify” request.

When verification is complete, the request status is changed to “CONFIRMATION SENT.” At this time, the request is considered complete and closed.

**Systems Administration**

When a storage request is placed into “Initial Review” status, the Systems Administrator determines if the requested storage for the file system can be extended using existing storage. If storage is available on the hosts, the System Administrator extends the file system using existing space, and changes the SRTS to “System Admin Complete.” If additional storage is required, the System Administrator completes the review process, adds a host WWN and host striping information, and changes “System Admin Review” to “Complete.” After storage allocation, the System Administrator pulls LUN information from the SRTS for each line item and then proceeds to create/extends file system on the host.

**Storage Administration**

After a storage request has entered “DESIGN COMPLETE” status, a Storage Administrator is assigned the request. At this point, Storage Administration alters the request status to “OPERATION IN PROGRESS.” If the Storage Administrator identifies any issues with the request, the request status is altered to “OPERATION ISSUES.” After all operational issues are resolved and the request is satisfied, the request status is altered to “STORAGE ALLOCATED,” and “add LUN” information is included for each line item.

Figure 9 illustrates the SRTS inputs and outputs process.
Figure 9. SRTS Inputs and Outputs Process Diagram
**Automation Process and Procedure**

The ESO team’s automation scripting process provides a consistent and repeatable way to provision storage. By using this process, we can enforce certain design standards that otherwise would be difficult to do with point and click SRM tools. Regardless of the data center, personnel, or vendor platform, all storage allocations are completed in the same manner. To enforce standards, the scripts were designed to be quite intelligent.

The automation process is broken down into 3 phases:

- Preparation
- Implementation
- Validation

**Preparation Phase**

Figure 10 on the following page illustrates the preparation phase process flow.
During the Preparation phase, the SRTS process takes the ticket from creation request through the time it is ready for allocation. Many different groups play key roles. These include the Project Managers, Storage Engineers, System Administrator, and Storage Operations. Each of these groups needs to validate the information and approve it before it moves to the next step.
After the ticket passes through the approval process, Storage Engineers design the ticket based on the requirements entered by the System Administrator and Project Manager. Storage Operations reviews the storage request for accuracy, reviews the System Administrator’s notes, and validates that storage standards were followed correctly. After all of this has taken place, the SRTS ticket is ready for allocation.

At this point, the automation process and procedure complete the request. The Storage Administrator connects to the Management Server located in the data center where the storage will be allocated. The administrator then creates a directory identified by the SRTS #. Next, the administrator will move into the directory and begin using the automated storage provisioning script named MAP_LUN_EMC. The SRTS number is passed as a command line argument to MAP_LUN_EMC.

```
#map_lun_emc  -s <srts_number>
```

The storage request information is pulled from SRTS using a Perl XML Call. The chance of a typo is minimized because this piece avoids manual entry. Perl provides many modules that help complete tasks quickly and easily.

After all data has been collected from the SRTS system, MAP_LUN_EMC collects all information about the Storage frame including FA information, device settings, number of free devices, LUN size, free LUN numbers on a per FA basis, etc. MAP_LUN_EMC. It compares the frame information against the request and verifies that the host platform is compatible with FA settings.

Following these verifications, MAP_LUN_EMC generates the required files (symconfigure input file, devmask file, zoning and lun mapping file) to complete the request. At this point, nothing has been changed on the Frame or SAN; changes will be applied during the Implementation Phase.

**Additional Information**

These validations include:

- checking device sizes
- verifying device count
- ensuring the storage and host are connected to the same switch and/or fabric
- enforcing the Rule of 17
- verifying the Front End compatibility with the server type, etc.

After all test cases are met, the script generates the files required to finish the request; including:
• MetaCreate/Mapping/Bit settings
• Device masking
• Zoning information
• LUN Information file that will be used by the SA to build the File Systems
• Other validation files

We pulled all of the design information directly from SRTS using XML. This method enables the scripts to take the data directly from SRTS without manual intervention, preventing typos and other errors when data is entered manually. If the script fails due to a design issue, we notify the Storage Engineering Team to make the needed changes before placing the ticket back into our queue for processing.

Implementation Phase

Figure 11 illustrates the implementation phase process flow.
Figure 11. Implementation Phase Process Flow

Implementation Phase

D

Start Implementation Phase

Update wwn.cfg file with all required HBA Information

Was SAN Validation Successful?
Yes

Execute Device Create, Mapping and Bit Setting Script

Did Script Complete Successfully?
No

Irresolvable Issue/ Ticket Redesign Required

Yes

Execute Devmask Script

Did Masking Complete Successfully?
No

Yes

Execute Zoning script that was created by devmask script?
Resolve Zoning Conflict or Issue

Did Zoning Complete Successfully?

Yes

Continue to Validation Phase

Implementation Phase Completed

B

C

A
During the implementation phase, the storage provisioning process creates devices, maps devices, sets bit settings on the frame, and masks the devices to each server impacted by the change. The SAN Administrator updates the WWN in the configuration file with the associated Host WWNs needed to complete the request. After the WWN is updated, the SAN Administrator completes the request, using the command files generated by MAP_LUN_EMCEMC script.

The next step is to execute the symconfigure_cmd.ksh script to execute symconfigure against all the storage frames in the request. This command script validates that the host and storage are connected to the same fabric before creating and mapping devices. Following successful completion, the next step is to complete device masking.

During the device masking process, the devmask file will mask devices to corresponding host WWNs, add WWN/host alias information, set HBA Port Flags as needed and then create the SAN zoning file needed to complete the request. The zoning file is passed as an argument to the zoning script. The zoning script validates the host/storage alias information before adding the zone members to the zones. At this point, the storage provisioning process is complete with the exception of final verification. If any errors or issues arise during the validation phase, the SAN Administrator works to resolve them before moving to the validation phase.

Validation Phase

Figure 12 illustrates the validation phase process flow.
Figure 12. Validation Phase Process Flow
During the verification phase, the SAN Administrator verifies the entire request, including:

- number of devices created
- device size
- device masking
- Front End Adapter/WWN bit settings
- Confirm zoning was completed successful

The MAP_LUN_EMC script generates all validation command files that need to run during the validation/preparation phase. This phase is very important to the development of the automation scripts. Most process improvements are found during this phase. The entire automation process is constantly changing and improving.

**Script Environment and Prerequisites**

The following list provides a snapshot of our provisioning environment.

- Sun T2000 running Solaris 10
- Perl 5.8.8 (see Appendix B. Perl-V Output)
- EMC Solutions Enabler Version V6.5.1.0 or higher commands used in automation script:
  - symmaskdb –sid <sid> list database (Query VCMDB Database)
  - symmaskdb -sid <sid> -dir <FA> -p <#>-v –wwn list database (Query VCMDB database for particular FA ports with verbose)
  - symcfg –sid <sid> list (List of frames manage on the server)
  - symdev –sid <sid> -noport list (Query list of devices in free pool, not mapped or masked)
  - symdev –sid <sid> -noport list –disk_group 1 (Query list of devices in free pool in disk group 1)
  - symdisk –sid <sid> list –by_diskgroup (Query physical spindle by different logical disk group)
  - symmaskdb -sid <sid> -dev <dev> list assign (Query VCMDB and pull masking record for one or more devices)
  - symmaskdb -sid <sid> -wwn <wwn> list devs (Query VCMDB and pull masking record for certain WWN)
  - symcfg -sid <sid> -dir <FA> -p <#> -address -available list (Query FA port mapping and find out available addresses on that FA port)
  - symcfg -sid <sid> -fa all list (List all the available FA ports on frame)
• symcfg -sid <sid> -fa all list –port (list all the available FA ports and their connectivity status)
• symmask -sid <sid> -dir <FA> -p <#> list logins (Query individual FA ports and get FCID and fabric connectivity information)
• symmask -sid <sid> -dir <FA> -p <#> -wwn <wwn> add dev <dev> (Update Masking Record)
• symmask -sid <sid> -dir <FA> -p <#> -wwn <wwn> set hba_flags on <flags> –enable (Set HBA port flag for initiator WWN during masking)
• symmask -sid <sid> -dir <FA> -p <#> -wwn <wwn> set hba_flags on <flags> –disable (Disable HBA port flag for initiator WWN during masking)
• symmask -sid <sid> -dir <FA> -p <#> -wwn <wwn> rename <hostname>/<wwn> (Add Host alias in VCMDB)
• symdev -sid <sid> ready <dev> -nop (Make Devices Ready)
• symconfigure –sid <sid> -f <file> commit –nop (Make Configuration changes, create new Meta devices, map devices, change device flag, set SCSI-3 etc)
• EMC Storage Arrays: DMX-2, DMX-3 and DMX-4
• Cisco MDS-9513 running SAN-OS 3.3.1c
• Brocade 24K and 48k running FOS 5.2.0a
• Shared NAS Mount is used for scripts
• RSYNC is used to keep Perl and all other requirements synced across environment.
Appendix A. Standard DMX4 BIN Configuration

**Raid Stripe Size:** We decided to configure the newly acquired DMX4-4500 all 300GB Drive Arrays as 3+1 RAID5 for both Birmingham and Nashville.

**Drive Replacement:** EMC recommends spanning all DA’s and Fiber Loops with an even distribution of 300GB drives to best utilize the available Disk Adapter (4 DA Pairs) processing capability. Power Vault devices will be allocated to 300GB drives.

**Permanent Sparing:** It has been verified that Permanent Sparing will only spare to a like size and speed of drive. After a permanent spare has been invoked, it will assume the group number of the drive that is being spared. Adequate spares must be provided to insure that any failing drive will have a Permanent Spare to fail over to.

*(Note: Drives holding the DMX Power Vault will not participate in Permanent Sparing).*

EMC’s standard is to allocate 2 spares per 100 physical drives. Based on the new sparing rules with 5772 code, the following spares are recommended:

<table>
<thead>
<tr>
<th>Drive Size</th>
<th>Number of Drives</th>
<th>Recommended Spares</th>
</tr>
</thead>
<tbody>
<tr>
<td>300GB</td>
<td>604</td>
<td>12</td>
</tr>
</tbody>
</table>

**Group Naming:** We will define Disk Groups in the BIN file. Each group of devices will be in a unique disk-group pool. This group number will be used by the SYMCLI scripts in selecting disk drives:

<table>
<thead>
<tr>
<th>Drive Size</th>
<th>Disk Group Number</th>
<th>Number of Drives</th>
</tr>
</thead>
<tbody>
<tr>
<td>300GB</td>
<td>0</td>
<td>12 (Spares)</td>
</tr>
<tr>
<td>300GB</td>
<td>1</td>
<td>384 (Production)</td>
</tr>
<tr>
<td>300GB</td>
<td>2</td>
<td>208 (BCV’s/Rs’s)</td>
</tr>
</tbody>
</table>
### LUN Sizes:

It was discussed that the 300GB drives will be set up using the current standard 11GB LUN sizes. Extra space on each physical drive remaining after allocating the 11GB LUNs will be made available for 1GB LUNs and GateKeepers. EMC has done some preliminary work, and we estimate that each 3+1 RAID Group can hold, assuming 11GB and 1GB volume sizes:

<table>
<thead>
<tr>
<th>Drive Size</th>
<th>11GB (12037 Cyl)</th>
<th>Vault</th>
</tr>
</thead>
<tbody>
<tr>
<td>300GB w/vault</td>
<td>74</td>
<td>12 (Spares)</td>
</tr>
<tr>
<td>300GB w/o vault</td>
<td>75</td>
<td>384 (Production)</td>
</tr>
</tbody>
</table>

### DRVs for Optimizer:

We determined that the number of recommended DRVs for Optimizer is 8. Therefore, both Birmingham and Nashville will have 8 DRVs per device size.

### Save Pool for RDF DSE:

TBD based on outcome of Replication Workshop.

### LUN Placement on Physicals:

When laying out the logical volumes on the physicals, the 1GB volumes will be placed on the outer cylinders, with the 11GB volumes following. This is needed for the 1GB Oracle Redo Logs to reside on the fastest possible area of the physical disks.

### Internal SYMM Numbers:

11GB volumes will occupy the first range of internal Symmetrix numbers (assigned in the BIN file), followed by the 1GB volumes, with all the Gatekeepers occupying the last ranges.

### Front-End Mapping:

Only the Volume Logix Database and GateKeeper devices are presented to the front end with the BIN file. All other devices are exposed during provisioning. All D ports on the array should be set up as RDF for all arrays in Birmingham and Nashville. The remaining 48 FE ports should be set as heterogeneous ports.
Appendix B. Perl-V Output

The information in Figure 13 and Figure 14 shows the options that were used to compile Perl. For additional information, go to [www.perl.org](http://www.perl.org) or [cpan.perl.org](http://cpan.perl.org).

Figure 13. Perl-V Output-Part 1

```
Summary of my perl5 (revision 5 version 8 subversion 8) configuration:
Platform
  osname=solaris, osvers=2.10, architecture=sun4-solaris-64int:
  uname='sunos bsdrid147 5.10 generic_127111-06 sun4v sparc sunfirma
  config_args=-Dprefix=/opt/perl -Dcc=-gcc -Duse64bitint
  hint=recommended, useposix=true, _d_sigaction=define
  usethreads=undef use5005threads=undef useithreads=undef
  usemultiplicity=undef
  useperlio=define _d_slio=undef uselargefiles=define usesocks=undef
  use64bitint=define use84bitall=undef uselongdouble=undef
  usenymalloc=n, bincompat5005=undef
Compiler:
  cc='gcc', cflags='-fno-strict-aliasing -pipe -Wdeclaration-after-statement -
  /usr/local/include -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -
  DPERL_USE_SAFE_PUTENV',
  optmize='-O3',
  copflags='-fno-strict-aliasing -pipe -Wdeclaration-after-statement -
  /usr/local/include',
  ccversion='3.4.8', gccosandvers='solaris2.10',
  intsize=4, longsize=4, ptrsize=4, doublesize=8, byteorder=87654321
  d_longlong=define, longlongsize=8, d_longdouble=define, longdoublesize=16
  ivtype='long long', ivsize=8, nvtype='double', nvsise=8, Off='off',
  |seeksize=8
  alignbytes=8, prototype=define
Linker and Libraries:
  ld='gcc', ldflags ='-L/usr/local/lib',
  libpath=/usr/local/lib /usr/lib /usr/ccs/lib
  libm=lsqot -lnsl -ldl -lm -lc
  perlbin=lsqot -lnsl -ldl -lm -lc
  libperl=libperl.a
  glm_libc_version=''
Dynamic Linking:
  dlsrc=dl/dlopen.xs, dext=so, d_dsymrun=undef, ccflags=''
  cccflags='-IIC', ldflags='-'G' -L/usr/local/lib'
```
Appendix C. MAP_LUN_EMC Script Sample Output

```
Using input File--> /2942_12951_12965_file
Creating Debug File...
Script/Config Base Directory--> /usr/Conf

Processing SRTS 12942: Line Item # 1_1...
12942[1]1[1][1][bhp][61][1][N][1][147689][3B][1][16][4176_1][swap][HP][1][1][bhp][61][1][PROD][N][200][EMC][swap]
Validating MetaStrip/MATH...

1_1[1][bhp][61][HP][4176_1][3B][1][1475][1][116
Disk Group is 1 and variance is 1
Server Variable Name=>bhp[61]
Number of HBA Pairs => 1
Gathering SYM 4175 Information...
Gathering SYM 4175 available devices...
Checking FA Pairings using Rule of 17...
Gathering HF Lun Information ...
GENERIC: /usr/symcli/bin/symcfg -std 4176 -dir 3B -p 0 -address list
GENERIC: /usr/symcli/bin/symcfg -std 4176 -dir 14B -p 1 -address list
Found Valid Lun Size...
Checking for enough free devices...
Total Devices Needed : 16
Total Devices Available: 3157
Generating Device information...
...
Generating Device information...
...
Generating Output Files...... for Frame 4091...
Generating Output Files...... for Frame 4119...
Generating Output Files...... for Frame 4121...
Generating Output Files...... for Frame 4173...
Generating Output Files...... for Frame 4176...
Done....

Processing Successfully Completed...
```
Appendix D. Files Created by MAP_LUN_EMC

MAP_LUN_EMC will create the following files and directories.

Files and Directories Created in the Base Directory

Create the following files and directories in the base directory:

- **wwn.cfg**: Centralized Host HBA WWN file.
- **symconfigure_cmd.ksh**: Performs Fabric Check. Script will execute all of the required symconfigure commands using nohup and in the background.
- **<Frame>.0 Directories**: 4091.0, for example
- **zone_file**: This file will be created by the devmask file. Used by targettool

Files and Directories Created Under the Frame Directory

Create the following files and directories under the Frame directory:

- **metacreate_scsi_mappit**: This file will create metas, map devices and set required device parameters on the EMC Frame.
- **devmask**: This file will mask the devices to each wwn. Uses wwn.cfg
- **lunmask**: Contains device to LUN mapping information.
- **.zones**: This file will be used to validate hba/fa's fabric information
- **.<frame>.xmlpost**: This file is used to upload the device mapping data to srts via srts_post.pl
- **mkready**: Enables Devices that were allocated. (This is only created if needed.)

Validation Directory

Create the following files and scripts under the Validation directory:

- **fa_verify**: Shows symmaskdb information on a per Host WWN and FA basis.
- **devmask_verify**: Shows devices that were assigned during the allocation. Validate that correct WWN's are listed next to each device.
- **devices**: List of devices allocated during provisioning.
- **dev_config**: Shows size, Device Status, Device Configuration, SCSI Parameters and FA information for each device allocated
Appendix E. metacreate_scsi_mappit

A sample of this file:

```
form meta from dev 1C2B, config=striped, stripe_size=1920;
    add dev 1C2C to meta 1C2B;
    add dev 1C2D to meta 1C2B;
    add dev 1C2E to meta 1C2B;
form meta from dev 1C2F, config=striped, stripe_size=1920;
    add dev 1C30 to meta 1C2F;
    add dev 1C31 to meta 1C2F;
    add dev 1C32 to meta 1C2F;

map dev 1C2B to dir 10A 1, lun=1E4;
map dev 1C2E to dir 7A:0, lun=1E4;
map dev 1C2F to dir 10A 1, lun=1E5;
map dev 1C2F to dir 7A:0, lun=1E5;
```

Appendix F. Zone File

An example of this file, which is passed to the zone automation script:

```
# Zone Configuration
#
# Note: Not all of these will require zoning.
# Note: Please check SRTS Notes and with the SA before you complete the zoning.
#
# Set up zoning:
hpgf2: 5001438002fe0504:4119_10Ab:50060482D5305DE9
hpgf2: 5001438002fe0504:4119_07AA:50060482D5305DC6
```

Appendix G: Lunmask File

An example of this file, which is sent to System Administrators:

```
Storage Port WWN
EMC4118_10Ab= 50060482D5305DE9
EMC4118_7AA= 50060482D5305DC6

#### hpgf2: 1,1 (Device Size: 45 G) 1,1

SYM FA Dev# Target LUN# SRTS# Unetem
4119 10Ab 1C2B E 4 12865 1,1
4119 10Ab 1C2F E 5 12865 1,1

SYM FA Dev# Target LUN# SRTS# Unetem
4119 7AA 1C2B E 4 12865 1,1
4119 7AA 1C2F E 5 12865 1,1
```
Appendix H: wwn.cfg file

An example of this file:

```bash
#!/opt/perf/4.0/bin/perl

# HBA WWNs required for devmask files
$bbpff2f5WWN1="500143800C2fe0904 casting 1C2B-noprompt
$bbpff2f5WWN1="500143800C2fe0924 casting 1C2F-noprompt
```

Appendix I: mkready file

An example of this file:

```
/usr/symcl/bin/symdev -sid 4176 ready 1C2B -noprompt
/usr/symcl/bin/symdev -sid 4176 ready 1C2F -noprompt
```

Appendix J: symconfigure_cmd.ksh

An example of this file:

```
#!/bin/sh

# Script will execute each frames symconfigure
# Validates Host HBA and FA are connected to the same fabric
# chf fab pl-f /sym/file.dat
if [ $? -eq 0 ]; then
    echo "chf fab pl failed...";
    exit 1;
fi

cd 4091.0
pushd symconfigure -sid 4091 -f /metacreate_scsi_map.dat commit-nop &
cd 4119.0
pushd symconfigure -sid 4119 -f /metacreate_scsi_map.dat commit-nop &
cd
```
Appendix K: Sample devmask file

An example of this file is shown in Figure 15 below and Figure 16 on the following page.

Figure 15. Sample devmask File-Part 1

```perl
#!/usr/bin/perl
use emc_module;
# HBA WWNs have been centralized to a file
# under the base SRTS DIR called wwn.cfg
do "./wwn.cfg";
# Custom Validation Module
ezmod_task->chk_wwn("4119");
ezmod_task->chk_hba("4119");
# Updating VCMDB with Device Changes
print "Running=synmac -sid 4119 discover
";
synmac -sid 4119 discover;
if ($? ne 0)
{
    print "SYNMACF Discover Failed. Sleeping 60"
    print "seconds and then script will retry
";
    print "Re-running synmac -sid 4119 discover
";
synmac -sid 4119 discover;
    if ($? ne 0)
        print "SYNMACF Discover Failed Again...
";
        print "Please run synmac -sid 4119 discover manually
";
        exit 1
;
} # SRTS: 12965 SubLine: 1 1 Frame: 4119 #
print "aRunning=synmac -sid 4119 -dir 10A -p 1 -wwn $(bhpfl2SWWN1) add dev 1C2B,1C2F"
print "Running=synmac -sid 4119 -dir 10A -p 1 -wwn $(bhpfl2SWWN1) add dev 1C2B,1C2F"
print "Running=synmac -sid 4119 -dir 7A -p 0 -wwn $(bhpfl2SWWN1) add dev 1C2B,1C2F -noprnt"
print "Running=synmac -sid 4119 -dir 10A -p 1 -wwn $(bhpfl2SWWN1) set hba flags on V,OS2007,SPC2-enable"
print "Running=synmac -sid 4119 -dir 10A -p 1 -wwn $(bhpfl2SWWN1) set hba flags on V,OS2007,SPC2-enable"
print "bHBA WWN already exists in the VCMDB, b\tSkipping HBA Port Flags Set\n";
}``
if(!(flash_port_flag_chk(4119)("7A"){$(bhpfl2PWWN1)}))
{
  print "arlSetting hba port flags on V,OS2007,SPC2-enable on ${bhpfl2PWWN1}";
  print "unRunning->symmask -sid 4119 -dir 7A -p 0 -wwn ${bhpfl2PWWN1} set hba_flags on V,OS2007,SPC2-enable ";
  system("symmask -sid 4119 -dir 7A -p 0 -wwn ${bhpfl2PWWN1} set hba_flags on V,OS2007,SPC2-enable ");
}
else
{
  print "arlWWN already exists in the VCMDB-AllowSkipping HBA Port Flag Settings"
}
print "unRunning->symmask -sid 4119 -dir 10A -p 1 -wwn ${bhpfl2SWWN1} rename bhpfl2S($bhpfl2SWWN1)";
system("symmask -sid 4119 -dir 10A -p 1 -wwn ${bhpfl2SWWN1} rename bhpfl2S($bhpfl2SWWN1)");
print "unRunning->symmask -sid 4119 -dir 7A -p 0 -wwn ${bhpfl2PWWN1} rename bhpfl2P($bhpfl2PWWN1)";
system("symmask -sid 4119 -dir 7A -p 0 -wwn ${bhpfl2PWWN1} rename bhpfl2P($bhpfl2PWWN1)");

#########################################################################
print "unRunning->symmask -sid 4119 refresh -noprompt <..ln";
system("symmask -sid 4119 refresh -noprompt ");
#########################################################################
# Releavse frame lock that was created by map_lun_sync
' release_lock < 4119';
# Copies zoning file to base SRTS Directory
'cat zones >> ..zone_file';
# Sends Device Information to SRTS
'srts_post_pl -f 4119.xmlpost' if(1 -e "./xmlpost_successful");
Appendix L. SAN Switch Topology Diagram

Our any-to-any SAN fabric topology supports our over-all automation efforts. We have 3-Tier SAN architecture, Host Edge, Core Edge and Storage Edge. Due to any-to-any connectivity, we don’t have to worry about where the host and storage are connected in the fabric. We use WWN based zoning and pull physical slot/port info from where the host/storage port is connected from the name server. We use one of the Core switches in fabric to make all zone changes. Figure 17 illustrates this SAN topology.

Figure 17. SAN Topology
Appendix M. FA Port methodology and fabric connectivity

This is an illustration of the FA port allocation methodology used in fabric. It is very important to follow the same standards for each frame for automation to work correctly.

Figure 18 illustrates this topology.

Figure 18. DMX-4 FA Port Allocation Methodology
Appendix N. Related Documentation

The following EMC manuals were referenced during the creation of this paper:

- Solutions Enabler Symmetrix Array Management CLI Product Guide P/N 300-002-181
- Solutions Enabler Symmetrix CLI Version 6.5 Command Reference P/N 300-002-181 REV A05
Glossary

**BCV.** Business Continuance Volume.

**FA.** Fiber Adapter (Symmetrix®). See also Symmetrix®.

**Host Bus Adapter (HBA).** An input/output controller that provides an interface between a computer system's input/output (I/O) bus. A card that contains ports for host systems.

**HP-UX.** Hewlett Packard-UNIX.

**Logical Unit Number (LUN).** An address for an end storage device such as disk, tape, floppy, CDROM, changer, etc. The address is a subdivision of a SCSI target identifier (ID). SCSI-2 specifications provide 8 logical units per SCSI target ID numbered 0-7. See also SCSI.

**LUN id.** The SCSI address of the LUN within the target as defined in the SCSI standard. See LUN. See also target and SCSI.

**LUN masking.** A method of access control where a hardware device presenting itself as a SCSI LUN does not reveal its address or WWN to another device, except for those devices explicitly permitted access. This is a method of access control that works fine, except for the fact that the access control list is held by the device configured to use LUN masking. If the device ever goes down, so does the access control mechanism. LUN masking per device is not preferred, because it introduces single points of failure. See also LUN, SCSI, WWN, and Masking.

**Masking.** The process of using a LUN masking driver to hide the LUN of a storage device, so the operating system of the server cannot access it. Typically used to prevent the corruption of networked storage devices while transferring backup data to a server on the network that has suffered data loss, often as the result of a disaster. The masking process prevents the risk of corruption resulting from access conflicts. See also LUN.

**RSCN.** Registered State Change Notification.

**SCN.** State Change Notification
Storage Area Network (SAN). A specialized high-speed network that connects computer systems, or host servers, to high performance storage subsystems. The SAN components include host bus adapters (HBAs) in the host servers, switches that help route storage traffic, cables, storage processors (SPs), and storage disk arrays. Storage devices are attached to servers in such a way that to the operating system, the devices appear as locally attached. Primarily employed in

Solutions Enabler (EMC® Solutions Enabler). An EMC package containing the SYMAPI runtime libraries and the SYMCLI command line interface. See SYMAPI and SYMCLI.

Solutions Enabler Storage Resource Management (SRM) component. See Storage Resource Management (SRM) component.

SRM. Storage Resource Management


SYMCLI. Symmetrix® Command Line Interface. A part of the EMC® Solutions Enabler package. This set of commands can be invoked within scripts or from the command line. These commands can be used to monitor device configuration and status and perform control operations on devices and data objects within a storage complex. See also Symmetrix and Solutions Enabler.

Target. A SCSI term meaning the device receiving an I/O exchange with an initiating device. Generally refers to storage devices in a SAN context.

TCO. Total Cost of Ownership.

WWN. World Wide Name.

XML. Extensible Markup Language.
**Zone.** A collection of fabric or loop nodes. Nodes in a zone cannot access nodes that are outside the zone, when hardware enforced. There can be more than one zone in a particular SAN. Zone enforcement can be performed either by hardware or by software. See Hardware zoning or Software zoning. See also SAN.

**Zoning.** A method of allowing or preventing access between devices in a SAN. Restricts server access to storage arrays not allocated to that server. Zones define which HBAs can connect to which SPs. See also SAN, HBA, and SP.
Biography: Ken Guest
Ken Guest has been working in the IT Industry since 1998. He received his Bachelors of Business Administration from Texas A&M University in 1998. Since then, he has worked for several companies in various IT roles including UNIX Systems Administration, Database Administrator, SAN/Storage Administration, Implementation Consultant and Technical Consultant. During the past 4 years, he has been working in the SAN/Storage Arena supporting Large Multi-Vendor Multi-Data Center Environments. As a Sr. Systems Manager/Automation Lead for a large telecommunications company, his day-to-day responsibilities include: storage provisioning, automation, performance monitoring/evaluation, SAN/Storage implementation and developing/implementing custom scripts that create/enforce standards and procedures.

Professional Certifications
- HP Certified IT Professional
- SUN Certified System and Network Administrator
- EMC Technology Foundations
- IT Service Management (ITIL)
- SNIA Storage Network Foundations

Biography: Sejal Joshi
Sejal Joshi has more than 15 years of experience in the IT industry. He completed a Bachelor of Engineering degree in Electronics in India in 1993, and is currently working for a large telecommunication company as a Sr. Technical Team Lead in the SAN/Storage Operations team. He and his colleagues manage Storage frames and SANs from multiple vendors. Also, he provides support for local and remote replication. Sejal’s primary responsibilities include storage provisioning, automation, performance monitoring/evaluation, and SAN/Storage implementations.

Professional Certifications
- HP Certified IT Professional (HP-UX Operating System)
- SUN Certified System and Network Administrator (SUN Solaris Operating System)
- EMC Technology Foundation (EMC)
- IT Service Management (ITIL)
- Brocade Certified Fabric Professional (BCFP)
- Brocade Certified SAN Manager (BCSM)
- Brocade Certified Fabric Designer (BCFD)
- Brainbench Certified UNIX Administration (HP-UX 11.00)