OpenVMS Technical Journal

V7, January 2006

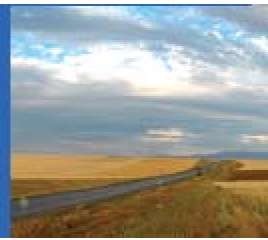


Table of Contents

SLS to ABS/MDMS Migration	5
Using VMS_CHECK to Collect OpenVMS Configuration Data	21
Oracle RDB Monitoring and Alerting on OpenVMS	31
Faking It with OpenVMS Shareable Images	45
WASD in SOAP/XML Transaction-Oriented Environments	143
Reusing OpenVMS Applications from Java	151
Preliminary OLTP Performance Comparisons of Oracle Rdb V7.2 on OpenVMS I64 and Alpha	169
The Development of a High-Performance VAX 6000 Emulator	173
Bringing Seismic Data to the Web with OpenVMS	179

© Copyright 2006 Hewlett-Packard Development Company, L.P.

OpenVMS Technical Journal V7

SLS to ABS/MDMS Migration

Ted Saul - HP Off-site Software Support Consultant

Introduction

The Storage Library System backup application has served the OpenVMS customer based extremely well since the late 1980's. Because ABS/MDMS is the go-forward backup application on OpenVMS operating system environments, this article provides guidance for migrating to ABS/MDMS from your SLS system.

This article describes the tasks the SLS system manager must accomplish during the migration. Each environment is different, and this paper cannot address every possible scenario. However, you can minimize the impact of the migration by understanding what you currently have in SLS and how it equates to ABS/MDMS.

The keys to a successful migration are:

- Understand how SLS relates to ABS
- Understand how the current SLS environment is configured
- Determine the changes that need to be made in the current backup environment
- Plan the objects and policies that need to be created in the ABS/MDMS environment.

Migrating from SLS to ABS/MDMS presents advantages and challenges. By using as much of the new functionality as possible, you can eliminate the drawbacks to using SLS. And by identifying the challenges of the new application, you can implement the migration as seamlessly as possible.

Some of the advantages of the ABS/MDMS application include:

- An object and policy driven software application
- A more robust GUI that can be used from an OpenVMS system or Windows based PC
- Stronger scheduling options
- An opportunity to change current procedures that are ineffective
- A more secure backup environment with less need to customize the code

© Copyright 2006 Hewlett-Packard Development Company, L.P

- Oracle, Windows, and UNIX backups
- Support for new devices introduced by HP

Some or all of the following challenges may affect your migration:

- Sorting out customized code embedded in the current SLS environment
- Changing the processes for operations
- Remembering how and why SLS was originally configured, and translating that information into ABS objects and processes. In some cases, SLS was configured by a person who has long since moved on, without leaving appropriate documentation.
- Designing a testing period as well as parallel production processing time
- Finding and restoring data from SLS history set information.
- Understanding how objects and symbols relate to each other

A written backup policy is a vital part of developing and maintaining a secure backup environment. The process of working through this documentation helps identify operational weaknesses and gives operations the guidance they need when exceptions arise. This plan should include a current layout of your data and the strategy for backing it up. Include information about scheduling and the type of every backup, as well as the retention period for the data. Document a step by step process that describes when backups are to be run so that new personnel can easily understand processes. Exceptions and how to handle events and errors will provide guidance during off-hours when IT managers and supervisors are not available.

The migration period from SLS to ABS/MDMS is an excellent time to develop or review the backup policy. Be sure to include how the project of migration will be accomplished and a summary of the implementation.

The following sections to address how to gather the necessary information for the final configuration of ABS/MDMS. Worksheets and templates help to manage and organize this data.

Differences between SLS and ABS/MDMS

Program / Coding Differences

ABS/MDMS contains mostly executable code, as compared to SLS with its 60% DCL command procedures. As a result, ABS/MDMS cannot be customized to a customer's environment as easily. Many of the SLS customizations are already in ABS/MDMS. Consider each modification in SLS and consider a strategy to implement this functionality in ABS/MDMS. You may have to implement a new process for operations to handle backups.

Backup / Scripting

To set up the environment in which backups will be completed, SLS uses backup scripts known as SBK files. It is easy to insert custom code into the backup processing in these DCL command files in order to make SLS behave differently. SBK files can easily be built on the fly from a list of disks that is created daily, to assist in load balancing. ABS/MDMS uses a system of policies that include SAVE, ARCHIVE, and ENVIRONMENT, to complete its work. SAVE policies can be created dynamically to be run in a "one-time" environment, but you will have to create a system of scripts to achieve this functionality. This policy-based environment of command procedures is different from the SLS environment and requires a ground up approach. SLS scripts will not work with ABS/MDMS.

ABS/MDMS also uses a RESTORE policy to schedule restores from backups, as opposed to the GUI-driven SLS restore screens. Any command procedures written for STORAGE RESTORE must be recreated using new ABS/MDMS commands.

History Sets and Cataloging

ABS/MDMS uses a series of files to create complex catalogs that may take longer to get through history sets. ABS/MDMS catalogs tend to grow larger in size and required a more granular catalog strategy. For example, in SLS, a simple "Image" history set in SLS may be sufficient for tracking a

© Copyright 2006 Hewlett-Packard Development Company, L.P

range of backups, but in ABS/MDMS, for better performance, an "Image" catalog for weekly, monthly and yearly time periods must be created. However, if only one large catalog is created, a similar amount of disk space is used, but the lookup and cleanup processing time improves.

User Interfaces

Both SLS and ABS/MDMS contain DCL interfaces. Almost any command that that can be completed using the ABS/MDMS GUI can also be done from the DCL command line. SLS provides three GUIs based on FMS form handling. Each serves its own purpose in allowing access to data in volume database or managing other databases. ABS/MDMS, on the other hand, contains one GUI; user privileges control the information that is accessible. The ABS/MDMS GUI can be run on an OpenVMS workstation or Windows-based system.

Saveset Format

Like SLS, ABS/MDMS uses the native OpenVMS backup saveset format. This allows for restores using the backup image should it be required. ABS also uses the native BACKUP.EXE image from SYS$SYSTEM making the applying and tracking of OpenVMS ECOs easier.

Scheduling

The ABS/MDMS scheduler is more robust than that of SLS. Where SLS can start its backup based on the day and time, with slight variations, ABS/MDMS schedules can be defined and associated with as many backups as desired.

Configuration

To configure SLS, you edit the SYS$MANAGER:TAPESTART.COM file, setting a series of symbols to the appropriate values to meet the needs of the environment. On the other hand, ABS/MDMS uses MDMS$SYSTARTUP.COM, which contains a series of object settings and uses one startup file for its configuration. Table 1 lists the symbols found in the SLS TAPESTART.COM procedure. Where available, the equivalent ABS/MDMS policy and field is referenced. Settings that must be included in the MDMS$SYSTARTUP.COM procedure are listed as well.

Table 1 - SLS Symbols and Equivalent ABS/MDMS Policies

TAPESTART SYMBOLS	**ABS/MDMS POLICY or Startup File**	**POLICY FIELD**
PRI	NODE MDMS$SYSTARTUP.COM	Database MDMS$DATABASE_SERVERS
DB_NODES	NODE MDMS$SYSTARTUP.COM	Database MDMS$DATABASE_SERVERS
PRIMAST	MDMS$SYSTARTUP.COM	MDMS$ROOT:[DATABASE]
SLS$SYSBAK_LOGS	MDMS$SYSTARTUP.COM	MDMS$ROOT:[LOGFILE]
SLS$MAINTENANCE_LOGS	MDMS$SYSTARTUP.COM	MDMS$ROOT:[LOGFILE]
SLS$SUMMARY_FILES	N/A	N/A
SLS$TEMP_HISTORY	MDMS$SYSTARTUP.COM	MDMS$ROOT:[CATALOG]
NET_REQUEST_TIMEOUT	N/A	
BATN	N/A	
MTYPE_n	MEDIA TYPE	
DENS_n	MEDIA TYPE	/density, /compaction
DRIVES_N	DRIVE	
TAPE_JUKEBOXES	JUKEBOX	

© Copyright 2006 Hewlett-Packard Development Company, L.P

MGRPRI	N/A	
VERBOSE	N/A	
PRIV_SEEANY	N/A	
PRIV_MODANY	N/A	
PRIV_MAXSCR	N/A	
PRIV_LABEL	N/A	
PRIV_CLEAN	N/A	
PRIV_MODOWN	N/A	
CRLF[0,8]	N/A	
CRLF[8,8]	N/A	
ESC[0,8]	N/A	
ESC_LOAD_BOLD	N/A	
ESC_LOAD_BLNK	N/A	
ESC_LOAD_NORM	N/A	
ESC_ALLOC_BOLD	N/A	
ESC_ALLOC_NORM	N/A	
ESC_MOUNT_OPER	N/A	
ESC_MOUNT_BOLD	N/A	
ESC_MOUNT_NORM	N/A	
LOC	DOMAIN	Onsite_Location
PROTECTION	DOMAIN	/protection
ALLOCSIZE	N/A	
LBL	N/A	
FRESTA	DOMAIN	/deallocate_state
TRANS_AGE	DOMAIN	/transition_time
ALLOCSCRATCH	DOMAIN	/scratch_time
BACKUPSCRATCH	N/A	
MAXSCRATCH	DOMAIN	/maximum_scratch_time
TAPEPURGE_WORK	N/A	
TAPEPURGE_MAIL	N/A	
VLT	DOMAIN	/offsite_location
ALLDEV	DRIVE	/shared
SELDEV	DRIVE	/shared
ALLTIM	N/A	

© Copyright 2006 Hewlett-Packard Development Company, L.P

TOPERS	DOMAIN	/opcom_classes
QUICKLOAD	N/A	
QUICKLOAD_RETRIES	N/A	
UNATTENDED_BACKUPS	N/A	
BACKUPSIZE	MEDIA TYPE	/length
BAKFMT	SELECTION	/data_selection_type
BAKOPT	N/A	
BACKUP_DEFAULT_REEL	N/A	
BAKQUE	ABS$nodename	
BACKUP_FINISH	ENVIRONMENT	/notification
HISNAM_1	CATALOG	
HISDIR_1	CATALOG	/directory
HISTYP_1	CATALOG	/type
RESOPT	N/A	
RESQUE	ABS$nodename	
RESTORE_FINISH	ENVIRONMENT	/notification
CLEANUP_Q	MDMS$SYSTARTUP.COM	MDMS$SCHEDULED_ACTIVITIES _START_HOUR
SYSCLN_RUN		
JUKEBOX	JUKEBOX	/control
JUKEBOX_1_LOWER	JUKEBOX	/cap
JUKEBOX_1_UPPER	JUKEBOX	/cap
SBARLOG	N/A	
SBARINT	N/A	
SBACLAS	N/A	

CLEANUP_Q in SLS defines the queue that it will run in, as well as the hour to start. ABS/MDMS uses SCHEDULER policies to start cleanup activities on the default queue ABS$*nodename*.

Note that the root directory for SLS is defined in the SLS$STARTUP.COM procedure. The root directory for ABS/MDMS is defined as the logical MDMS$ROOT in the MDMS$SYSTARTUP.COM procedure. If you want to move the ABS/MDMS directory from its default location of SYS$COMMON:[MDMS.], redefine this logical.

Databases

Table 2 lists the SLS database files stored in the location pointed to by the logical name SLS$MASTER. Where applicable, the equivalent ABS/MDMS database is listed.

Table 2 - SLS and Equivalent ABS/MDMS Database Files

SLS DATABASE	ABS/MDMS EQUIVILENT	DESCRIPTION
TAPEMAST.DAT	MDMS$VOLUME.DAT	Volume Information

© Copyright 2006 Hewlett-Packard Development Company, L.P

POOLAUTH.DAT	MDMS$POOL_DB.DAT	Pool Information
SLS$MAGAZINE_MASTER_FILE.DAT	MDMS$MAGAZINE_DB.DAT	Magazine Information
SLOTMAST.DAT		Slots outside Jukebox
HOLIDAYS.DAT		Holidays to skip
VALIDATE.DAT		Remote node access
	MDMS$ARCHIVE_DB.DAT	ARCHIVE Policy
	MDMS$DOMAIN_DB.DAT	DOMAIN Objects
	MDMS$DRIVE_DB.DAT	DRIVE Objects
	MDMS$ENVIRONMENT_DB.DAT	ENVIRONMENT Policy
	MDMS$GROUP_DB.DAT	GROUP Objects
	MDMS$JUKEBOX_DB.DAT	JUKEBOX Objects
	MDMS$LOCATION_DB.DAT	LOCATION Objects
	MDMS$MEDIA_DB.DAT	MEDIA Objects
	MDMS$NODE_DB.DAT	NODE Objects
	MDMS$REPORT_DB.DAT	REPORT Objects
	MDMS$RESTORE_DB.DAT	RESTORE Policy
	MDMS$SAVE_DB.DAT	SAVE Policy
	MDMS$SCHEDULE_DB.DAT	Schedule Objects
	MDMS$SELECTION_DB.DAT	SELECTION Objects

Capturing the SLS Environment

Now that you understand the differences between the SLS and ABS/MDMS, the next step is to gather information about the currently running SLS system. For long-term SLS users, this is a good time to review why SLS was set up as it is. Table 3 shows an example worksheet that is helpful for capturing information about the current environment.

Table 3 - Backup Information Worksheet

Image SBK Name	Incremental SBK Name	Disk	Media	Start Days	History

© Copyright 2006 Hewlett-Packard Development Company, L.P

To gather the information you need, follow this procedure:

1. Determine the actively scheduled backups and their starting times and days.

If no other scheduler than SLS is involved, you can identify current backups by searching the SLS$SYSBAK:*.COM procedures by entering the following command:

```
$ search *.com "days_"/window=4
```

For each SBK where these scheduling parameters are in use, something similar to the following, along with the SBK file name, is returned:

```
$!
$       DAYS_1 :== Monday,Tuesday,Wednesday,Thursday,Friday
$       TIME_1 :== 21:00
$       NODE_1 :==
$!
$!       DAYS_2 :==
$!       TIME_2 :==
$!       NODE_2 :==
$!
$!       DAYS_3 :==
$!       TIME_3 :==
$!       NODE_3 :==
```

In this case, a backup has been discovered that will run Monday through Friday at 9:00 PM on the current node. If the symbols are commented out, the SBK file is not scheduled and potentially is a manually-submitted backup.

2. Determine the type of each scheduled backup.

The name of the SBK usually indicates the type of backup (image or incremental). You can check this by editing the SBK and examining the /QUALIFIERS line for IMAGE, INCREMENTAL, SINCE, etc. After you determine the backup type, record the name, start time, and day of week to run, on the worksheet.

3. Determine the disks that are being backed up in each SBK.

This information can be found in the FILES_n qualifiers in each SBK. Record the disks that are backed up, and associate them with the SBK, on the worksheet. For every disk being backed up there is one associated FILES_n.

4. Determine what type of device is to be used during the backup.

The symbol MEDIA_TYPE in the SBK indicates the tape drives that are used and what types they are. For example, a TYPE of TK89 or DLT probably indicates that they are these types of drives. If the type is not evident, you can trace the MEDIA_TYPE in the SYS$MANAGER:TAPESTART.COM and the

© Copyright 2006 Hewlett-Packard Development Company, L.P

associated media triplet. The DRIVES_n that is matched with MTYPE_n is the correct device. Use SHOW DEVICE /FULL from the OpenVMS command prompt state to determine the type of drive is being used. Record this information with the corresponding SBK information.

5. Determine the SLS history set from which information about each SBK run is recorded.

Each SBK file contains the symbol HISTORY_SET with a defined value. Record this information in the worksheet. The name of the history set probably corresponds to its type. For example, IMAGE may indicate that all backups recorded here are of the image type. Take this into consideration when you create new CATALOGS in ABS/MDMS.

6. Check for manually submitted backups.

In some environments, SBKs are started manually by using the STORAGE STARTUP SYSTEM_BACKUP command. Any SBKs without scheduling parameters may be manual runs. Unfortunately, the only way to determine which SBKs are a part of the daily runs is to refer to the company's backup policy or procedures. It is to be hoped that any SBK that is run manually are documented.

Log files in the SLS$SUMMARY_FILES, SLS$MAINTENANCE_LOGS, and SLS$SYSBAK_LOGS directories tell which backups are run on any one day. By default, SLS places log files from all backup runs in these directories. If you find any manual backups, record the information that is described in steps 1 through 5.

7. Check for third-party or custom schedulers.

As with manually submitted backups, the manager of the SLS environment should have information about backups that run using a third-party scheduler. To integrate them into the ABS/MDMS environment. document the information described in Steps 1 through 5 for them. ABS/MDMS can use most third-party schedulers in place of its own. You designate the scheduler when you configure ABS/MDMS.

8. Capture other appropriate files.

Keep the SLS configuration file TAPESTART.COM in order to set the appropriate objects and policies in ABS/MDMS. Other data that is needed may come from the VALIDATE.DAT file and POOLAUTH.DAT, both of which are visible from the SYSMGR menu. The HOLIDAYS.DAT file indicates days that certain backups are to be skipped. Check this file for any custom scheduling.

9. Documenting customized code.

It is important to understand any customized code implemented in your SLS environment. Document what has been changed and why. Knowing the goal of the changes helps make sure that ABS/MDMS carries out the same behavior.

Analyze the Current Backup Environment

The migration process allows you to change the SLS backup environment. Over time, system managers often identify more efficient methods for their backup strategies. This is an excellent time to implement these strategies. In addition, the backup policy documentation should be modified to reflect these changes.

Some questions you should ask are:

- Are files backed up often enough for a reasonable restore?
- How long does it take to recover from a disk failure?
- Are users satisfied with their backups and ability to restore?
- Are all tape devices being used to their fullest capacity?
- Does ABS/MDMS solve any issues with the current backup policy?
- Can a complete disaster be recovered?
- What are the steps to bare metal restore?
- Where is the application historical data stored?

© Copyright 2006 Hewlett-Packard Development Company, L.P

- Who is responsible for managing the DR strategy?
- How are notifications handled by applications?
- Do Windows or UNIX servers requiring backing up?
- Do Oracle databases require backing up?

Its a good idea to Interview operators and system administrators, as well as users who rely on the restore of data. All the stakeholders who are affected by the backup strategy should be taken into consideration. It is also appropriate, with the current focus on security and disaster recovery, to analyze the safety of the current environment. Refer to the *Archive Backup System for OpenVMS – Guide to Operations* for more information.

ABS/MDMS Policies and Objects

ABS/MDMS uses objects and policies to define its work and its environment. In order to configure ABS/MDMS, you need to understand the purpose each object and policy.

Table 4 describes the objects to consider when you install ABS/MDMS and migrate from SLS. Each object must be defined before a SAVE can be initiated. In Configuring ABS/MDMS using data gathered from SLS. each of the symbols in SLS are associated with these objects.

Table 4 - ABS/MDMS Objects

Object	Description
DRIVES:	A physical resource that can read and write data to tape volumes.
GROUPS:	A logical grouping of nodes to be allow common actions to be taken at one time.
JUKEBOXES:	Any robot-controlled device that supports automatic loading of tapes.
LOCATIONS:	An object that describes a physical location where tapes may be placed. You can define a hierarchy of locations.
MAGAZINES:	A logical object that contains a set of volumes that will need to be moved as a group.
MEDIA TYPES:	A logical object that describes certain attributes of tape volume media.
NODES:	A list of the nodes in the domain that run ABS/MDMS.
POOLS:	A logical object that contains a set of volumes that can be allocated to authorized users.
VOLUMES:	A single piece of tape media that the ABS/MDMS application can use to store its data.

A final object to consider is the DOMAIN. The DOMAIN is special because it encompasses all objects that are served by a single MDMS database. A domain consists of objects (including DRIVES, JUKEBOXES, NODES, and VOLUMES), as well as logical objects (including MEDIA TYPES, POOLS, and MAGAZINES). It is important to understand what devices in your computer room will belong to your ABS/MDMS DOMAIN and associate them accordingly.

Table 5 describes the policies used in the ABS/MDMS backup environment.

Table 5 - ABS/MDMS Policies

Policy	Description
ARCHIVE:	Defines the media type as well as other characteristics about where backup data is to be stored. One single ARCHIVE can then be associated with many SAVE policies.
CATALOGS:	Defines where data about each backup will be stored. Different types of

© Copyright 2006 Hewlett-Packard Development Company, L.P

	CATALOG policies serve different purposes, including how backup will restore. It is important to understand which type will be most efficient for your site.
ENVIRONMENTS:	Describes the criteria under which save and restore requests must execute. As with the ARCHIVE, one ENVIRONMENT can be associated with many SAVES.
SAVE:	Defines unique information about the backup. This policy has other policies associated with it.
RESTORE:	Defines unique information about a restore in order to return data from a tape to disk. Data can be restored from entire disks, down to individual files. The ability to restore depends on the type of CATALOG defined.
SELECTIONS:	Holds information about files, disks, paths and databases to be saved or restored. A SELECTION can be generated automatically using the /include qualifier on a SAVE. or it can be generated manually using either the CREATE SELECTION DCL command or the MDMS GUI. Then the SELECTION needs to be associated with a SAVE or RESTORE.
SCHEDULES:	Defines backup schedules. ABS/MDMS provides a flexible scheduler. By default, an associated SCHEDULE is created with a SAVE. SCHEDULES can also be created manually and then associated with a SAVE or RESTORE.

The SAVE, RESTORE and associated policies are created using ABS/MDMS objects. This strategy reduces the amount of redundant data that needs to be entered. For example, in SLS, the MEDIA_FORMAT had to be modified on every SBK. In ABS/MDMS, only the MEDIA FORMAT object needs to be entered once on the policy called an ARCHIVE. This ARCHIVE can then be associated with many SAVE policies.

Configuring ABS/MDMS using data gathered from SLS.

After you install ABS/MDMS, following the procedure in the *ABS/MDMS Installation Guide*, it is time to tailor ABS/MDMS to match the way SLS accomplished your backups. Be sure to follow the post-installation tasks in the manual, as well.

it is a good practice to go through each of the policies to determine what will you will need to create in ABS/MDMS. You must also decide how the fields in each policy will be set. The tables of objects and policies include information about SLS equivalents, where appropriate. Make a list of each object and policy and the quantity that will needed to be created. For example, if you have two jukeboxes where one contains a TZ89 drive and the other contains a DLT7000, note that a total of six objects must be created: two jukeboxes, two drives, and two media types.

Continue this process until you have accounted for all the resources in the ABS/MDMS environment.

ABS/MDMS Object Creation

The MDMS$SYSTEM:MDMS$CONFIGURE.COM command procedure creates the required objects. You are prompted for the information required to create the policy. After each policy has been created, you can fine tune it to meet the needs of the environment. For help within the command procedure, enter two question marks (??) at the prompt.

Before you start, make sure you have at least the following information:

- Media type (user definable)
- Onsite location (user definable)
- Offsite location (user definable)
- IP domain name for node (if using the TCPIP transport)
- Name of your jukebox (user definable)
- Robot name (i.e., GKB601:)
- Drive name (user definable)

© Copyright 2006 Hewlett-Packard Development Company, L.P

- OpenVMS device names (i.e., ALERAB$MKB200:)
- Volume naming strategy (should match bar code label)

After all objects have been successfully created, prepare the jukebox for use by inventorying and initializing the volumes in the jukebox.

To synchronize your ABS/MDMS with your jukebox, enter the following command:

```
$ MDMS INVENTORY JUKEBOX your_jukebox_name
```

This command updates the MDMS database with the information it finds in the jukebox's firmware. When the inventory is complete and the volumes in the jukebox need to be initialized, enter the following command:

```
$ MDMS INIT VOLUME/JUKEBOX=your_jukebox_name/SLOT=(beginrange,
endrange)
```

Once these commands have completed, your volumes are ready for use with ABS/MDMS. These commands also test the robotic configurations to ensure that they all work.

ABS/MDMS Policy Creation

When all the objects are in place, you can create the backup policies. There is no conversion utility to create these policies from SBK files; therefore, you have to enter them manually. This step allows you to make changes and to improve any backup inefficiencies that may be taking place. You can use either the ABS/MDMS GUI or DCL commands to create the policies. Create the policies in the following order:

1. CATALOGS
2. ARCHIVES
3. ENVIRONMENTS
4. SAVEs

RESTORES are created as needed. SELECTIONS and SCHEDULES are usually created automatically with the SAVEs.

Once the CATALOGS have been created, use the SLS information on your worksheet to create the ARCHIVE, ENVIRONMENT, and SAVE policies for each SBK file. ARCHIVE and ENVIRONMENT policies can be used to group similar types of backups. For example, if you have backups that use the same CATALOG policy and MEDIA TYPE, you can create one ARCHIVE for all these SAVES. Similarly image backups of a RAID array that are run on a regular basis can be grouped together. Information about the backups can be retained in the same CATALOG for the same length of time. Data that cannot be grouped includes monthly and weekly backups where the retention times are different. In this case, use different CATALOGS with separate ARCHIVE policies. Obviously, the policy structure must be well thought out and planned before beginning the configuration of ABS/MDMS. Once it is in place, ABS/MDMS provides efficient and logical methods for tracking backups.

Testing the environment

The ABS/MDMS testing procedures cover three areas: devices, databases, and running the actual saves.

Devices

Test your Jukebox by entering the following commands:

```
$ MDMS LOAD VOLUME/DRIVE=your_drive your_volume
$ MOUNT/FOREIGN your_drive
$ DISMOUNT your_drive
$ MDMS UNLOAD VOLUME your_volume
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

If these commands complete successfully, you can assume that your JUKEBOX and DRIVES policies are configured correctly. Use the SHOW DRIVE/CHECK command to physically access the drive. If this command fails, there is probably a connectivity problem to the drive. Use the MRU application to diagnose whether the problem is with ABS/MDMS, or with the configuration of the drive.

Remember: if the drive is working with SLS, then the problem probably lies in the ABS/MDMS configuration.

Databases

Use the $SHOW MDMS commands to ensure database connectivity. If any of these commands fail or hang, call the HP customer support center. Test all the SHOW commands because, unlike SLS, ABS/MDMS uses many different database files to store data.

SAVES and RESTORES

SAVES and RESTORES can be tested online any time. You can test general ARCHIVE, SELECTION, and SCHEDULE policies by first creating "one_time_only" SAVEs that automatically purge themselves. Use a test CATALOG, set the ARCHIVE with a short retention period, or plan to recreate these databases before putting ABS/MDMS into production.

Once you are comfortable with the way ABS/MDMS schedules and releases backups, run them in parallel with your existing SLS backups, as described in Running SLS and ABS/MDMS Simultaneously. Check the log files in the directory pointed to by the logical SLS$SYSBAK_LOGS for errors. See Troubleshooting for additional troubleshooting tips.

Define the cut-off date for SLS backups, when the ABS/MDMS environment will take over full responsibility for the backups.

Additional Issues

Oracle Backups

As in SLS, Oracle RDB backups are available in ABS/MDMS. When you create a SAVE request , select the appropriate DATA_SELECT_TYPE for the version and area of RDB to be backed up. This automatically creates the SELECTION policy. This backup should be scheduled like any other ABS/MDMS backup. Though it is not required, it is a good practice to create a separate CATALOG for these types of backups.

ABS/MDMS can also back up Oracle 8i and 9i databases. (This functionality is not available in SLS.) Refer to the ABS/MDMS documentation to learn how to set up this type of database.

What to do with SLS data in History Files?

At this time, there is no way to read the SLS history files from ABS/MDMS and initiate a backup. You can recover pre ABS/MDMS data from SLS using one of the following methods:

1. Keep an instance of SLS running for history lookups only. The SLS and ABS licenses support this environment, and once the information is located in SLS, you can initiate a manual restore command using the Backup utility.
2. Capture data from SLS volumes and feed it directly into ABS/MDMS catalogs. This can be a large task, so consider carefully which backups need to be restored mostly commonly.

To catalog existing savesets, create a SAVE policy by enter the following command:

```
# MDMS CREATE SAVE saveset_catalog-
/INCLUDE=tape:*-
/DATA_TYPE=VMS_SAVESET-
/ARCHIVE=archive-
/ENVIRONMENT=environment
/START=start_time
```

The data from the tape is written to the catalog designated by the ARCHIVE policy.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Running SLS and ABS/MDMS Simultaneously

SLS and ABS/MDMS can run simultaneously on one system. To set up this environment, follow this procedure:

1. In the SYS$STARTUP:MDMS$SYSTARTUP.COM file, set the logical MDMS$VERSION3 to False. This logical enables STORAGE commands to look at the ABS/MDMS databases. Though this specific functionality is no longer applicable, setting the logical to False ensures that the two applications will not interfere with one another.
2. Install the test ABS/MDMS environment on the appropriate node.

CAUTION

The greatest danger in running SLS and ABS/MDMS together on the same node is that you may cross volumes between the two applications. Make sure that each application knows only about the volumes that it can use. You can accomplish this by identifying which volumes in the jukebox will be designated for use by SLS and by the ABS/MDMS applications, and then do one of the following:

- Create a pool in ABS/MDMS called "SLS" and place all of the volumes to be used by SLS in this pool.
- Create a pool in SLS called "ABS" and place all the volume to be used by ABS in this pool

This procedure sets aside volumes in each application that the other will be using, preventing them from allocating and using one another's volumes.

Another way to prevent use by each other's application is to not define the volumes in ABS. Though the jukebox will be loaded with both, inventory and allocations will overlook volumes they don't know about.

Troubleshooting

This section describes additional troubleshooting procedures.

Startup Issues

The ABS/MDMS equivalent to the SLS command procedure SLS$ROOT:[000000]TAPESTART*nodename*.COM is MDMS$LOG:MDMS$STARTUP_*nodename*.LOG. This procedure reveals issues during the startup of the product. As with SLS, turning on OPCOM may reveal problems such as syntax errors and licensing issues.

Save and Restore Issues

SLS users can read log files for system backups from the directory SLS$SYSBAK_LOGS. Similarly, ABS/MDMS puts its log files in the directory pointed to by ABS$LOG. You can check SAVEs and RESTOREs for completion status by scanning the associated log files. By default, the log files have the same name as the SAVE policy itself. To monitor these logs, enter the following command:

```
$ TYPE/TAIL/CONT
```

This command shows you the end of the file as the log buffer is dumped to disk.

History / Catalog Issues

The ABS$CATALOG:Catalog_*n*.LOG file tracks the way files are processed and staged for catalogs. Check this file to determine whether recently backed-up data is not showing up in the appropriate catalog. The SLS equivalent file is SLS$SBUPDT.LOG in SLS$MAINTENANCE_LOGS.

The ABS$CATALOG:ABS$CATALOG_CLEANUP.LOG file contains information about the daily cleanup of catalogs and removal of obsolete records. Check this file if you suspect that your catalogs are not be cleaned up as volumes freed. In SLS, the SLS$DATA:SYSCLN.LOG and SLS$DATA:CLEANUP.LOG contain this information.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Miscellaneous logs (no SLS equivalents)

The ABS$CATALOG:ABS$COORD_CLEANUP_nodename.LOG file reveals any problems with the ABS/MDMS coordinator, which is responsible for a number of different functions. If you suspect there is a problem with the coordinator, this log is a starting point for troubleshooting.

The MDMS$LOG:MDMS$LOGFILE_DBSERVER.LOG file tracks system events and errors detected by the MDMS$SERVER process.

Other files in the MDMS$LOGFILE directory, as well as additional settings that provide more in-depth trouble shooting information, may be used at the direction of HP Services if the need arises.

Summary

The keys to a successful migration include:

- Understanding your current environment in order to capture it in ABS/MDMS. Without this understanding, you may end up reinventing how you do your backups.
- Understanding the differences between SLS and ABS/MDMS, in order to know what changes need to be made to your current processes. In general, ABS/MDMS handles backups in a more efficient manner than SLS. Your site may have adapted to using SLS a certain way and the change may be difficult. Create a change management plan as a part of your migration plan to explain the reasons for the migration and the benefits of moving to ABS/MDMS.
- Knowing how you want to change your backup policy, in order to incorporate these changes into your new ABS/MDMS environment. Now is a good time to improve the company backup plan that may have been in place for years.
- Running SLS and ABS/MDMS simultaneously for a designated period of time shows that ABS/MDMS is capable of performing necessary backups safely. A successful migration period reduces the risk to the company, as well the fears that the application won't be able to handle your required backup strategy

The HP Services Value Added Services Team is available to assist with your migrations. There are many ways and levels that HP can help including:

- Migration planning. HP can help develop your plan to ensure its success.
- Migration execution. If time is limited and it is difficult to begin this project, HP can manage and execute the process by providing the documentation and the instruction that is required. Any changes would be reviewed by your company to ensure compliance with its standards.
- Help sorting out current SLS environments. If it has been a long time since SLS implementation took place, a qualified HP engineer familiar with both SLS and ABS/MDMS can help you understand all that is being accomplished at your site by SLS. In turn a knowledge transfer can be done to allow you to continue with your migration.
- Auditing the current backup strategy. HP engineers experienced in SLS and ABS/MDMS can look as a third-party to ensure that your backup strategy is as secure as you believe it is.
- Reviewing your plans. HP engineers can review your plans and can be available for you to bounce your thoughts and ideas off of.

For more information

This white paper, complete with templates and relational addendums, is available from the HP Call Center.

To contact the HP Services VAST team, ask one of your HP service engineers or Technical Account Managers, or call (888) 376-4737.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Using VMS_CHECK to Collect OpenVMS Configuration Data

Kostas G. Gavrielidis, Master Technologist HP Services

Overview

You can choose from several layered products and utilities for collecting operating system configuration and performance data, along with the layered products configuration and performance for the HP OpenVMS operating system. Because OpenVMS runs on three different hardware architectures (VAX, Alpha and Itanium), you have to choose the right tool. This article presents the `VMS_Check` utility, which I developed for collecting OpenVMS configuration information. `VMS_Check` is written entirely in the DIGITAL Command Language (DCL). DCL is similar to any of the UNIX shells, such as the Bourne shell (sh), the C shell (csh), and the Korn shell (ksh); it is a command language interpreter that parses commands and passes control to the programs that make up the OpenVMS operating system. While programs developed on any one of the OpenVMS compilers such as, C/C++, Pascal, BLISS, FORTRAN, COBOL, and so forth, they may not run unchanged or without relinking on all the three architectures; DCL procedures work without changes.

How VMS_Check Works

VMS_Check functions like the `sys_check` and `cfg2html` tools, which run on UNIX systems:

- The `sys_check` tool provides configuration and analysis of information gathered on the system. It is useful for debugging or diagnosing system problems. The `sys_check` tool gathers information on over 60 components and subsystems, and performs over 200 analysis operations. It gathers this information into easy to browse and transportable files. These files are sent to support engineering when escalating IPMT cases. It runs on all supported version of the Tru64 UNIX operating system and is included in the Tru64 UNIX operating system and the patch kits.
- The `cfg2html` tool is a UNIX shell script that creates system documentation for HP-UX 10+11, AIX, SCO-UX, SunOS and Linux systems in HTML and ASCII formats. Plugins for SAP, Oracle, Informix, MC/SG, FibreChannel, TIP/ix, Mass Storage like XP48/128/256/512/1024/12000,

© Copyright 2006 Hewlett-Packard Development Company, L.P

EVA3000/EVA5000, Network Node Manager, and OmniBack/DataProtector, and so forth, are included.

VMS_Check is a DCL procedure that runs on all the three OpenVMS architectures and is extendable – you can include in it any series of OpenVMS commands as if you were entering them at the OpenVMS operating system command prompt (the $). The current version of the VMS_Check tool collects data from any system, standalone or in an OpenVMS Cluster, and presents it in both its original form and with HTML wrappers. The main report is an HTML file named of VMS_Check-<*nodename*>-<*ddmonyy-hhmm*>.HTML. For example:

```
VMS_Check-OWL-14MAR2005-1516.HTML
```

This main file is supported by several text and HTML files, which contain the actual data that make up the complete system report.

The Purpose of VMS_Check

The primary goal in devoloping the VMS_Check tool was to collect the data on a customer's configuration. It started out as a small procedure with the goal to collect database related configuration information. Slowly it grew to a large DCL command procedure that now includes operating system and storage configuration information.

All of the VMS_Check report sections include information in tables or plain text which can easily be used elsewhere, such as in any of the Microsoft tools Word, Excel, etc.

Table 1 shows an example of a table generated on an OpenVMS Cluster system, including information about each node, its version, node name, current date and time, and system uptime.

BBCX Cluster Nodes Table			
OpenVMS Version	**Node Name**	**Current Date and Time**	**Uptime**
OpenVMS V7.3-2	BBC200	25-FEB-2005 11:40:47.70	27 10:33:44
OpenVMS V7.3-2	BBC202	25-FEB-2005 11:40:47.73	27 09:51:18
OpenVMS V7.3-2	BBC204	25-FEB-2005 11:40:47.78	9 12:45:43
…	…	…	…
OpenVMS V7.3-2	BBC309	25-FEB-2005 11:40:48.12	19 12:44:58
OpenVMS V7.3-2	BBC311	25-FEB-2005 11:40:48.15	2 12:05:09

Table 1 Cluster Nodes Table

Data Collected by VMS_Check

The VMS_Check tool collects setup and configuration information for databases and associated layered products, such as Oracle, Rdb, ACMS, Ingres, and so forth, on OpenVMS platforms.

© Copyright 2006 Hewlett-Packard Development Company, L.P

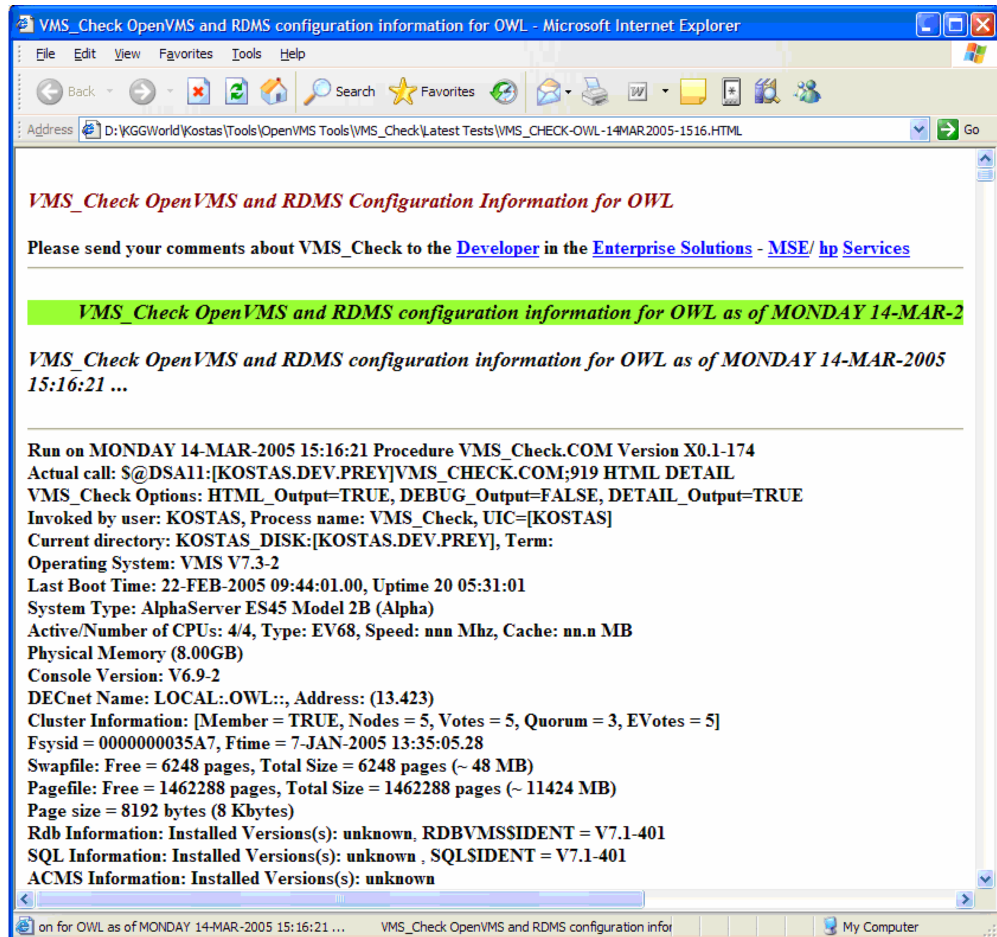


Figure 1- VMS_Check Report in the MS IE Browser

Sections of VMS_Check Reports

As in most generated HTML reports, `VMS_Check` creates a table of contents at the beginning of the report. The following example shows the table of contents generated by the VMS_Check tool:

© Copyright 2006 Hewlett-Packard Development Company, L.P

Table of Contents

OpenVMS Operating System

Console: Variables | RAD Information | Partition Information
Procedures for System: Sylogin | Startup | Shutdown | ModParams
System identification information: GETSYI of this node | GETSYI of all VMScluster nodes |
Show commands for: System | CPU | Memory | Pool | Files | Reserved Memory | RMS | Users | Logicals | Symbolics |
Analyze: System | RAD | SpinLock |
Errolog and Crash Information: Errorlog | DECevent | Crash analysis
SYSGEN Parameters: SYSGEN | Startup | Special | All
System Access and Control: UAF Records | User Rights | Proxies | SYSTEM UAF record | DEFAULT UAF records | Table
Installed Images: Installed images
System tests: RADcheck
VMScluster Configuration Information: VMScluster | Noders Table | GETSYI table for all VMScluster nodes

Storage Subsystem Configurations

Storage: Devices | IO Bus | IO Circuits | IO Devices | Mounted | Devices Table | Devices Charts | Devices Fragmented Files | Stripe | RAID | FDDI | HSC | HSJ

Network and Related Products Information

Network | NCP | NCL | LATCP | LANCP | UCX | MultiNet

Database Information/Configurations

SQL: SQL | SQL Images | UAF Records
Relational Database Operator: RDO
Oracle Rdb: Images | Logicals | Versions | Databases | Schemas | Statistics | UAF Records
Oracle RDBMS: Oracle | Schemas | Statistics | UAF Records
Sybase RDBMS: Sybase | UAF Records
Ingres RDBMS: Ingres | UAF Records

Transaction processing and other layered product information

ACMS: ACMS | Images
TDMS: TDMS
DECforms: DECforms
PathWorks: PathWorks
DECWindows: DECwindows
DECthreads: Images
CMA: Images
Layered Products: Installed | Installation History | Licensed
HyperSort: Images

Performance Data

Performance Solution Advisor (PSA) | Monitor Utility

Tables

VMScluster nodes | GETSYI information for all VMScluster nodes | Devices | SYSUAF

Interactive Sessions

0:Mornitor | 1:Rdb | 2:Rdb | 3:ACMS | 4:PSA | 5:SPL

Goto: Top | Contents | Bottom

© Copyright 2006 Hewlett-Packard Development Company, L.P

Navigating the VMS_Check Report

You can jump to different sections of interest in the report from inside the main HTML report file, and under each section of the report. The following menu appears:

Goto: Top | Contents | Bottom
Section: System | Storage | Network | Database | Layered Products | Performance | Interactive

The **Goto** references jump to the Top, Table of Contents and Bottom sections of the report.

The **Section** references jump to the System, Storage, Network, Database, Layered Products, Performance and Interactive sections of the report.

Internal References to Other Files

The main VMS_Check report file contains internal references to other text and HTML files generated by VMS_Check. These file references are described in **Table 2**. This example was generated for the OpenVMS system named OWL.

Internal References generated for OWL	
Reference	**Description**
Devices_Mounted-OWL.txt	All devices mounted
Devices_DU-OWL.txt	All DU devices
Devices_HSJ-OWL.txt	All HSJ devices
SDA-of-running-OWL.txt	Analyze System
SPL-of-running-OWL.txt	Spinlock Information
…	…
PSA-Brief-OWL.TXT	Performance Analysis (Brief)
PSA-Full-OWL.TXT	Performance Analysis (Full)
PSA-Perf-OWL.TXT	Performance Evaluation
MON-ALL-SUM-OWL.TXT	Monitor all classes (Summary)
MON-ALL-AVE-OWL.TXT	Monitor all classes (Average)

Table 2 - Internal References

Information about Mounted Devices

Table 3 shows the table of all mounted devices that is generated by VMS_Check in the Storage section of the report. It include information about each device, including the device name, the volume name, the device type, the total blocks, the free blocks, percent of blocks free, fragmentation index, and the fragmentation report.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Devices on OWL							
Device	Volume	Type	Total blocks	Free blocks	%Free	Frag Index	Frag Report
_DSA2:	ALP_SITEB	DGX00	68251131	24772725	36	44.1	frpt
_DSA11:	ALPHA_USER	DGX00	71112778	24899409	35	5.2	frpt
_DSA12:	DATABASE	DGX00	213291762	38709535	18	2.4	frpt
_DSA500:	OWL_PAGE	DKX00	35565080	20564320	57	30.2	frpt

Table 3 - Mounted Devices Table

VMS_Check generates a bar graph for the percent of free disk space and the fragmentation index for all devices. **Figure 2** shows the pie chart graph of the total disk capacity, which includes the total MB's used and free disk space.

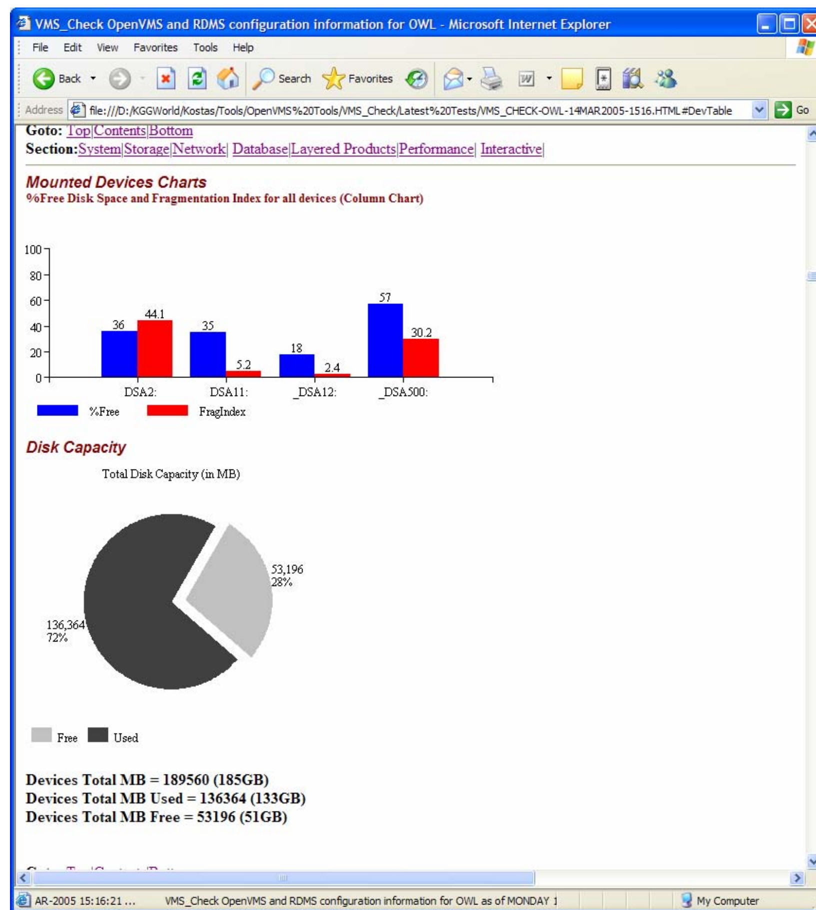


Figure 2 - Mounted Devices Charts

Console Environment Variables

Table 4 describes some of the information that can be collected from the console.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Console Variables for OWL			
Name	**V/N**	**Value**	**Function**
auto_action	N	RESTART	Specifies the action the console will take following an error, halt or power-up. Values are: restart, boot and halt
auto_fault_restart		UNDEFINED	Controls whether the SCM will restart when a fault is encountered.
Boot_dev	N	SCSI3 0 10 0 3 0 0 0 @wwid0,SCSI3 0 10 0 4 0 0 0 @wwid0,SCSI3 0 8 0 1 0 0 0 @wwid0,SCSI3 0 8 0 2 0 0 0 @wwid0	Defines the default device or device list from which booting is attempted when no device name is specified by the boot command.
Bootdef_dev	N	SCSI3 0 10 0 3 0 0 0 @wwid0,SCSI3 0 10 0 4 0 0 0 @wwid0,SCSI3 0 8 0 1 0 0 0 @wwid0,SCSI3 0 8 0 2 0 0 0 @wwid0	Defines the default device or device list from which booting is attempted when no device name is specified by the boot command.
…		…	…

Table 4 - Console Environment Variables

GETSYI Information

The GETSYI informatoin for a single node is shown in **Table 5**.

GETSYI Information for EMPIRE		
Type	**Value**	**Description**
ACTIVECPU_CNT	2	Count of the CPUs actively participating in the current boot of the symmetric multiprocessing (SMP) system.
AVAILCPU_CNT	2	Number of CPUs available in the current boot of the SMP system.
ARCHFLAG	245760	Architecture flags for the system
ARCH_NAME	Alpha	Name of the CPU architecture on which the process is executing
BOOTTIME	24-SEP-2004 14:11:24.00	The time when the node was booted.
…		
VP_MASK	0	Longword mask, the bits of which, when set, indicate which processors in the system have vector coprocessors.
VP_NUMBER	0	Unsigned longword containing the number of vector processors in the system.

Table 5 - GETSYI Information From a Single Node

© Copyright 2006 Hewlett-Packard Development Company, L.P

GETSYI clusterwide information is shown in **Table 6**

GETSYI Information for all VMSCluster Nodes				
Item	**EMPIRE Value**	**AIRTRN Value**	**…**	**STORM Value**
CLUSTER_EVOTES	11	11		11
CLUSTER_FSYSID	0000000034AD	0000000034AD		0000000034AD
CLUSTER_MEMBER	TRUE	TRUE		TRUE
CLUSTER_NODES	11	11		11
DECNET_FULLNAME	LOCAL:.EMPIRE::	LOCAL:.AIRTRN::		LOCAL:.STORM::
HW_MODEL	1976	1962		2030
HW_NAME	AlphaServer ES45 Model 2	AlphaServer DS10L 617 MHz		hp AlphaServer ES47 7/1000
NODENAME	EMPIRE	AIRTRN		STORM
…	…	…	…	…
NODE_SYSTEMID	0000000035A5	0000000034B0		00000000343A
NODE_VOTES	1	0		1

Table 6 - Clusterwide GETSYI Information

How to Use VMS_Check

You can invoke the VMS_Check tool with the `VMS_Check-detail.COM` command procedure, which is included with the software kit. This command procedure invokes the VMS_Check command procedure by setting the appropriate privileges and flags for VMS_Check to collect configuration information about the RDBMSs and the OpenVMS environment. For example:

```
$ @VMS_Check-Detail
```

This DCL command procedure actually performs the following steps:

```
$! Determine output file name
$!
$ nodename = f$getsyi("nodename")
$ systime = f$edit(''f$time()', "TRIM")
$ date = f$element(0, " ", systime)
$ time = f$element(1, " ", systime)
$ day  = f$element(0, "-", date)
$ mon  = f$element(1, "-", date)
$ year = f$element(2, "-", date)
$ hour = f$element(0, ":", time)
$ min  = f$element(1, ":", time)
$ if f$length(day)  .eq. 1 then day = "0"+day
$ filename = "VMS_Check-"+nodename+"-"+day+mon+year+"-
"+hour+min+".HTML"
$! show symbol filename
$ SET PROC/PRIV=(ALL,NOBYPASS)
$ exec := SPAWN/NOWAIT/INPUT=NL:/OUTPUT='filename'
```

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
$! show symbol exec
$ EXEC @VMS_Check.COM HTML DETAIL
```

To invoke VMS_Check in a single system environment, enter the following commands:

```
$ SET PROC/PRIV=(ALL,NOBYPASS)
$ EXEC:= SPAWN/NOWAIT/INPUT=NL: -
 /PROCESS=VMS_Check -
 /OUTPUT=VMS_Check-'F$getsyi("NODENAME")'.HTML
```

To generate HTML output and to get a DETAIL description of the current environment, enter the following command:

```
$ EXEC @VMS_Check.COM HTML DETAIL
```

To debug VMS_Check, enter the following command:

```
$ EXEC @VMS_Check.COM DEBUG NODETAIL
```

The DEBUG option creates the VMS_Check.DEBUG file in the current directory. This file has a record of all actions performed by VMS_Check preceded by a time stamp for the start of each action.

To generate HTML output from all the nodes on the VMS Cluster, enter the following commands:

```
$ MCR SYSMAN
SYSMAN> SET ENV/CLUSTER
SYSMAN> DO -
_SYSMAN> SPAWN
_SYSMAN> /INPUT=NL:/OUTPUT=VMS_Check_'F$getsyi("NODENAME")'.HTML -
_SYSMAN> /PROCESS=VMS_Check -
_SYSMAN> @DSA110:[KOSTAS.DEV]VMS_Check.COM HTML NODETAIL
SYSMAN> EXIT
```

Completing the Report Generation

The completion state of the report is at the end of the generated HTML file. Figure 3 shows an example of successful report generation:

```
*----------------------------------------------------------------------*
*
*  VMS_Check has successfully completed.                              *
*
*----------------------------------------------------------------------*
```

Generated by VMS_Check.COM X0.1-177 on 16-MAY-2005 14:13:31.10

Figure 3 - Successful Report Generation

How to Review the VMS_Check Report

To review the report and associated generated files, transfer them from the OpenVMS environment to another environment, such as a Windows laptop, Use FTP to transfer all the files generated by VMS_Check, in ASCII mode. For example, you can use the following FTP commands to transfer the VMS_Check files to your laptop for review:

```
ftp> open a.b.c.d.com
Connected to a.b.c.d.com.
220 a.b.c.d.com FTP Server (Version 5.4) Ready.
User (a.b.c.d.com:(none)): kostas
331 Username kostas requires a Password
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

```
Password:
230 User logged in.
ftp> cd [kostas.dev.prey]
250-CWD command successful.
250 New default directory is DSA11:[KOSTAS.DEV.PREY]
ftp> hash
Hash mark printing On  ftp: (2048 bytes/hash mark) .
ftp> prompt
Interactive mode Off .
ftp> mget *
...
ftp> quit
```

For more information

For informaton about `cfg2html`, go to http://come.to/cfg2html.

For information about `sys_check`, go to: http://h30097.www3.hp.com/sys_check/.

To get a copy of the VMS_CHECK tool please download:
http://h71000.www7.hp.com/openvms/journal/v7/vms_check.zip

© Copyright 2006 Hewlett-Packard Development Company, L.P

Oracle Rdb Monitoring and Alarming on OpenVMS

Kostas G. Gavrielidis, Master Technologist HP Services

Overview

Several layered products and utilities exist for monitoring Oracle Rdb databases on OpenVMS platforms. In this article, we review products available from other companies and propose an HP internally-developed solution developed and deployed in customer production environments.

There are several Rdb monitoring and alarming utilities and tools available today:

- `RMU/SHOW STATISTICS` Utility

  The `RMU/SHOW STATISTICS` utility, one of the most powerful and useful tools available to database administrators (DBAs), *analyzes* performance characteristics of a database. However, the ability to analyze performance problems is only one aspect of a performance analysis tool. The DBA also needs to *detect* problems in a timely manner, then analyze the problems and immediately perform *corrective* actions.

- PATROL Knowledge Module for Rdb

  PATROL monitors and manages the resources in the environment using information from special files known as Knowledge Modules (KM) that are loaded in the console. If PATROL detects a problem with a particular computer or application that it is monitoring, these modules provide information for PATROL to attempt to fix the problem. If the problem escalates or requires human intervention, PATROL displays a warning or alarm condition for every resource affected by the problem.

- Oracle Rdb Trace and Rdb Expert

  Oracle Trace is a layered product that gathers and reports event-based data from OpenVMS layered products and application programs that contain its service routine calls. Oracle Trace provides Oracle Expert for Rdb along with data for optimizing existing Rdb databases. Event-based data can be collected from products that contain Oracle Trace service routine calls. In addition, Oracle Trace routine calls can be added to other user-developed applications to

© Copyright 2006 Hewlett-Packard Development Company, L.P

collect data from them. This process of adding Oracle Trace service routine calls to an application is called *instrumenting the application*.

- Internally-Developed Solutions

  Internally-developed solutions consist of the `RMU/SHOW STATISTICS` Utility along with alarming and notification features based on rules customized by the database administrator.

This article is written for those who implement enterprise solutions that involve HP customer's Oracle Rdb and OpenVMS production environment configurations.

Database Monitoring Objectives

Database objectives include providing monitoring and alarming capabilities that support overall Enterprise Management (EM) activities. These objectives maximize the availability of applications and support the monitoring and alarming requirements that meet the following objectives:

- Develop software scripts, as necessary, to facilitate the monitoring and alarming of important databases by focusing on specific areas previously identified by the customer. These include (but are not limited to) the following areas:

  a. database status (i.e., up or down)

  b. elapsed transaction processing time

  c. transaction rate

  d. file system fragmentation

  e. record locking

  f. record level fragmentation

  g. critical transaction rate

  h. lock rate

  i. transaction duration

  j. number of fetches

  k. number of stores

  l. number of erases

- Document all developed software and the installation and deployment processes that be performed in the customer production environment.

RMU/SHOW STATISTICS Utility

Introduction

Most database products provide tools to analyze the performance characteristics of the database and the application. These tools often require constant manual attention or extensive post-processing of recorded data in order to detect critical events that might be detrimental to the smooth operation of the database.

In some cases, the DBA is not always available to constantly monitor the utility output, especially for 24*7 production databases. Post-processing recorded data is certainly not timely enough to prevent or resolve potential real-time database downtime situations, particularly with real-world, mission-critical requirements. The DBA needs the ability to be *notified* automatically when time-critical or interesting events occur to the database.

© Copyright 2006 Hewlett-Packard Development Company, L.P

RMU/SHOW STATISTICS Events

The RMU/SHOW STATISTICS utility has been dramatically enhanced for Rdb7 to include many new features that provide DBAs with the information necessary to *detect*, *analyze*, and *correct* performance problems in the database. Even more enhancements have been added to the Rdb7A release, including the *Configuration File* facility, which provides an extremely powerful and easy-to-use *User-Defined Event* facility.

Using the RMU/SHOW STATISTICS configuration file, you can persistently define special *events* that specify the action to be performed when something of interest occurs to the runtime database. An event is an identification of a particular statistic value on which the RMU/SHOW STATISTICS utility will perform some user-defined action. In other words, an event is signaled when a statistic's value exceeds a user-defined set of thresholds.

Using events, the DBA can be automatically notified by an RMU/SHOW STATISTICS utility server running on behalf of a particular database. You define an event by specifying a threshold against a specific statistic and by optionally specifying the attributes the event is to have.

Event Definition Syntax

The RMU/SHOW STATISTICS utility configuration file is a text file that can be maintained using the editor of your choice. The configuration file typically resides in the database directory, although it can reside anywhere that you desire.

Each entry in the configuration file uses the general format variable=value;. The equal-sign (=) separating the variable and value is required.   Note also that each definition is terminated with a semicolon character (;).

User-defined events are specified using the EVENT_DESCRIPTION variable. Events themselves are *not* named; rather, they are defined on behalf of a specific statistic for a given threshold. The EVENT_DESCRIPTION variable's value is a free-format description of the user-specified event. The event definition consists of three required, position-dependent components and an optional component. The following example describes the general format:

```
EVENT_DESCRIPTION="operation \
    statistic_name \
    threshold_name \
    [ attribute_list ]";
```

Event "Operation" Clause

The event *operation* clause identifies the action to be performed for the EVENT_DESCRIPTION operation. The keyword ENABLE is used to enable a new event or change an existing event definition. The keyword DISABLE is used to disable a previously defined event. This keyword is typically used when importing a new configuration file.

Even though an event may be enabled, it may not be active. For an event to be *active*, you must also specify either a program to be invoked or one or more operator classes to be notified.

During runtime, events can be disabled only by the RMU/SHOW STATISTICS utility or by importing a new configuration file that explicitly disables an event.

Event "Statistic Name" Clause

The event *statistic name* clause identifies the particular valid statistic field for which the event will be enabled or disabled. Note that some statistic names are valid only when certain database attributes

© Copyright 2006 Hewlett-Packard Development Company, L.P

are enabled, such as global buffers or record caching. The names of a particular statistic field can be found on the individual screen of interest. When statistic field names contain multiple words, such as *process attaches*, the statistic name must be either single- or double-quoted; failure to quote the statistic name may result in a syntax error.

Certain statistic fields have leading blanks. This white space is considered part of the statistic name and is necessary to identify a unique statistic. However, the use of leading blanks is often difficult to discern in the configuration file. Therefore, the underscore character (_) or dash character (–) can be used in place of spaces in statistic names that have leading spaces. For example, the statistic field name " *file extend*" can also be specified as "_ _ *file_extend*" or "- - *file-extend*". This method improves the readability of difficult statistic field names.

Most general events are defined using the *summary* statistics screens. However, it is sometimes necessary to define an event on a specific table or index, or even a particular partition of a table. The AREA attribute allows you to specify this type of *drill-down event* and indicate the name of a particular storage area. When this clause is specified, the statistic field selected must be from the *IO Statistics (by file)* or *Locking Statistics (by file)* screens. The identified statistic name can be also qualified with the `LAREA` attribute to specify the name of a particular logical area, such as a table, btree index, hash index, or blob. When this clause is specified, the statistic field selected must be from the Logical Area screens. Further, if the selected logical area is partitioned across multiple storage areas, the AREA clause can also be used to identify a specific partition against which to define the event.

The following table explains the semantics for specifying the `AREA` and `LAREA` clauses to qualify a statistic field name:

AREA	LAREA	Description
No	No	General Statistic Field
Yes	No	Storage Area Statistic Field
No	Yes	Logical Area Statistic Field, all partitions
Yes	Yes	Logical Area Statistic Field, single partition

The `AREA` and `LAREA` clauses are attributes and must follow the *Threshold Name* component of the event definition.

Event "Threshold Name" Clause

The event *threshold name* clause identifies the particular event threshold for which the specified statistic will be enabled or disabled. The thresholds are essentially the columns in the numeric version of the statistics screens.

Up to eight different thresholds can be specified for a particular statistic field, although each individual event name must be specified in its own `EVENT_DESCRIPTION` variable definition. The *threshold name* clauses are as follows:

- `MAX_RATE` – The maximum "current" occurrence-per-second rate collected. This threshold only *increases* as each event is signaled.
- `MAX_CUR_TOTAL` – The maximum "total" value collected since the database was opened. This threshold only *increases* as each event is signaled.
- `MIN_CUR_RATE` – The lowest rate currently being sustained. This threshold remains constant.
- `MAX_CUR_RATE` – The highest rate currently being sustained. This threshold remains constant.

© Copyright 2006 Hewlett-Packard Development Company, L.P

- MIN_AVG_RATE – The lowest average rate. This threshold only *decreases* as each event is signaled.
- MAX_AVG_RATE – The highest average rate. This threshold only *increases* as each event is signaled.
- MIN_PER_TX – The lowest per-transaction rate. This threshold only *decreases* as each event is signaled.
- MAX_PER_TX – The highest per-transaction rate. This threshold only *increases* as each event is signaled.

Event "Attribute List" Clause

The optional event *attribute list* clause provides additional characteristics for *enabled* event thresholds. In general, these attributes are ignored when disabling an event. Any or all of the event attributes can be specified for each event name within the same EVENT_DESCRIPTION variable definition. The *attribute list* clauses are as follows:

- AREA *storage_area_name* – Defines the name of a particular storage area. When this clause is specified, the statistic field selected must be from the "IO Statistics (by file)" or "Locking Statistics (by file)" screens, unless the LAREA clause is also specified. This clause is not ignored when disabling the event.
- LAREA *logical_area_name* – Defines the name of a particular logical area, such as a table, btree index, hash index, or blob. When this clause is specified, the statistic field selected must be from the "Logical Area" screens. This clause is not ignored when disabling the event.
- INITIAL *value* – Defines the initial value of the "current" event threshold. The default value is zero (0) for MAX_XXX thresholds and "very big number" for MIN_XXX thresholds. The default value guarantees that at least one event will be signaled, thereby initializing the new "current" threshold value.
- EVERY *value* – Defines the value by which the initial threshold will be incremented or decremented when an event is signaled. If this value is the default value zero (0) for any event except the MIN_CUR_RATE and MAX_CUR_RATE events, then the event will be signaled only once.
- LIMIT *value* – Defines the maximum number of times the event can be signaled. If the value is the default value zero (0), events can be signaled indefinitely providing that the EVERY clause is specified with a non-zero value.
- SKIP *value* – Defines the number of event notifications to *ignore* before performing an actual notification. This clause is extremely useful for the MIN_CUR_RATE and MAX_CUR_RATE events, as the thresholds for these events are not reset upon being signaled. The default value zero (0) ensures that all events are notified.
- NOTIFY *oper_class_list* – Defines the *quoted* comma-separated list of operators to be notified for all events defined on the specified statistic. Valid operator keywords are CENTRAL, DISKS, CLUSTER, SECURITY and OPER1 through OPER12.
- INVOKE *program_name* – Defines the user-supplied program to be invoked for all events defined on the specified statistic. On OpenVMS, the program name is specified as a DCL process global symbol known to the RMU/SHOW STATISTICS utility.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Event Description Readability

Very long configuration file lines can be *continued* on the next line by terminating the line with a back-slash (\). As the last character of a line, this continuation character is used to indicate that the configuration entry is continued on the next line exactly as if it were entered as a single line. Lines can be continued practically indefinitely, up to 2048 characters, even within quoted values. The following example demonstrates how to define a multi-line event description:

```
EVENT_DESCRIPTION="ENABLE 'pages checked' \
    MAX_CUR_TOTAL \
    INITIAL 7 \
    EVERY 11 \
    LIMIT 100 \
    INVOKE DB_ALERT";
```

The continuation character is not limited to the `EVENT_DESCRIPTION` variable; it can be used for any configuration variable.

Comments can be embedded in continued lines if they start at the beginning of the next line. The following example demonstrates two event descriptions containing embedded comments. The comment in the second event description takes precedence over the line continuation character.

```
EVENT_DESCRIPTION="ENABLE ' (Asynch. reads)' \
    MAX_CUR_TOTAL \
    AREA EMPIDS_OVER \
! this will work as expected
    INITIAL 6 EVERY 10 LIMIT 100 \
    INVOKE DB_ALERT";
EVENT_DESCRIPTION="ENABLE ' (Asynch. reads)' \
    MAX_CUR_TOTAL ! this will NOT work as expected \
    AREA EMPIDS_OVER \
    INITIAL 6 EVERY 10 LIMIT 100 \
    INVOKE DB_ALERT";
```

Event Semantics

For an event to be active, you must specify either one or both of the `NOTIFY` or `INVOKE` attribute clauses. When using the `INVOKE` attribute clause, the program must be specified by defining a process-global symbol pointing to the DCL command procedure or image to be invoked. The `INVOKE` program and `NOTIFY` operator classes apply to all events defined for the statistic field. Therefore, these clauses need to be defined only *once* per statistic field, no matter how many events thresholds are defined for that statistic. By specifying multiple programs or operator classes, only the last-specified attribute is used.

Once an event has been signaled, it will only be re-signaled if the `EVERY` attribute clause was specified with a non-zero value. The current threshold value, originally initialized to the `INITIAL` value, will be advanced for `MAX_XXX` thresholds and declined for `MIN_XXX` thresholds. The exceptions to this rule are the *current rate* thresholds `MIN_CUR_RATE` and `MAX_CUR_RATE`, which are never advanced nor declined. The `MIN_XXX` thresholds disable themselves once the `INITIAL` value reaches zero (0), while the `MAX_XXX` thresholds never disable themselves.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Once an event has been disabled, it can be re-enabled only by importing a new configuration file or by manually using the *Statistics Event Information* screen, *Re-enable all disabled events* configuration sub-menu option. Individual events cannot be re-enabled on line.

How User-Defined Events Work

The user-defined events are analyzed by the RMU/SHOW STATISTICS utility at the specified screen refresh rate. The default screen refresh rate of 3-seconds is ideal for most databases. However, using a 1-second refresh rate will produce a finer granularity event signaling mechanism. Multiple events defined for the same statistic field may cause the specified program to be invoked multiple times (once for each affected event).

As the RMU/SHOW STATISTICS utility identifies a statistic field whose current value or average value is changing, it examines any defined event thresholds established for that statistic field. This manner of examination minimizes the impact of event analysis, since the analysis is performed as part of the normal statistics collection process.

When the utility determines that a specified event threshold has been exceeded, an event is signaled. The signaling of the event means that any specified programs will be invoked and any specified operators will be notified. The event notification occurs immediately.

If you defined a program that will be invoked when an event is signaled, the program will be invoked with eight parameters. Some of the parameters contain multiple words that must be quoted if the parameters are passed to other utilities.

The parameters passed to invoked programs are as follows:

- P1 – This parameter is the date and time the event occurred. This parameter contains embedded blanks.
- P2 – This parameter is the statistic field name. This parameter may contain embedded blanks.
- P3 – This parameter is the event name.
- P4 – This parameter is the current event numeric value, expressed to the nearest tenth.
- P5 – This parameter is the word *above* or *below*.
- P6 – This parameter is the current event threshold value.
- P7 – This parameter is the event occurrence count.
- P8 – This parameter is the optional physical area and/or logical area name for the statistic field.

The P8 parameter is either null (blank) or contains the name of the affected storage area and/or logical area. The following example contains a log file sample output where an event for a partitioned logical area was signaled. Note that when both the storage area and logical area names are specified, they are separated by a period (.).

```
pages checked MAX_CUR_TOTAL 6.0 above 4.0 count is 1
area is EMPIDS_MID.EMPLOYEES
pages checked MAX_CUR_TOTAL 32820.0 above 5.0 count is 1
area is EMPIDS_OVER.EMPLOYEES
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

Runtime Event Status Information

The current runtime status of the user-defined events can be examined using the new *Statistics Event Information* screen, located in the *Database Parameters* sub-menu. Note that you do *not* have to be viewing this screen to signal events. Note also that the physical area and logical area identifiers are only displayed in *Full* mode.

Real-Life User-Defined Event Example

Nothing demonstrates a new feature better than a real-life example explained in step-by-step detail. For the purposes of this example, suppose that the DBA wants to be sent email whenever a database *freeze* occurs. A database *freeze* occurs when an application process on the database prematurely terminates (i.e., “dies”). Such an event results in all application activity being temporarily suspended until the recovery operation for the terminated process has been completed. This is a very significant and serious runtime event that should be immediately detected.

Using events to notify the DBA when a process terminates prematurely is very easy to accomplish. The following steps describe how this can be achieved using the `RMU/SHOW STATISTICS` utility *User-Defined Events*:

1. Identify the operation. Because you are going to define a new event, specify the `ENABLE` operation keyword.
2. Identify the statistic name to which the event will be assigned. Use the "process failures" statistic from the "Recovery Statistics" screen, which is located in the "AIJ Information" sub-menu. This statistic is available even if you are not using after-image journaling.
3. Identify the threshold name to use. Use the `MAX_CUR_TOTAL` threshold, since this represents the current number of processes that have failed.
4. Identify the event attributes to use. This is probably the hardest part of defining an event. You want to be alerted to *any* process failure, so you must set the INITIAL attribute to zero (0). Since you want to be notified on each and every process failure, set the `EVERY` attribute to one (1) and the `LIMIT` attribute to zero (0).
5. Define how you will be alerted about the event. Since you want to be sent mail, use the `INVOKE` clause. Invoking a program on OpenVMS requires that you define a "DCL process-global symbol" to identify the actual DCL script, as is demonstrated by the following example:

```
$ DBR_LOGGER :==@SYS$SYSTEM:DBR_LOGGER.COM
```

6. Write the program to be invoked. Since you want to be sent mail with a clear description of the event actually signaled, use the simple DCL script in the following example:

```
$ set noon
$ create /nolog sys$scratch:dbr_logger.tmp
EOD
$ open /write dbr_logger sys$scratch:dbr_logger.tmp
$ write dbr_logger " '''p1' ''p2' ''p3' ''p4' ", -
" ''p5' ''p6' (count is ''p7') area is ''p8' "
$ close dbr_logger
$ mail sys$scratch:dbr_logger.tmp -
         RDB_DBA_USER /subject="DBR notification"
```

7. Combine all of this information into the configuration file entry. The following example contains the final event description as you would enter it in the configuration file:

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
EVENT_DESCRIPTION="ENABLE 'process failures'\
 MAX_CUR_TOTAL \
 INITIAL 0 EVERY 1 LIMIT 0 \
 INVOKE DBR_LOGGER";
```

8. Invoke the RMU/SHOW STATISTICS utility as a server, using the configuration file. Be sure to use the /CLUSTER command qualifier if you want to be notified of cluster-wide events. The following example demonstrates the command-line to perform this operation:

```
$ RMU/SHOW STATISTIC -
    /CONFIG=CONFIG.CFG -
    /NOINTERACTIVE /UNTIL=HH:MM:SS -
    MF_PERSONNEL
```

Because applications increasingly require 24*7 availability, the rate at which DBAs are expected to react to potential downtime increases accordingly. The RMU/SHOW STATISTICS utility *User-Defined Events* provides the means by which DBAs can be automatically alerted when such critical situations arise, therefore enabling timely corrective actions.

Glossary of Terms for this Section

Configuration Variable – A symbolic name that defines a value that can be used to define other variables' values, or can be used by the RMU/SHOW STATISTICS utility.

Drill-Down Event – An event defined on a specific storage area, logical area, or logical area partition statistic.

Event – The identification of a particular statistic value on which the RMU/SHOW STATISTICS utility is to perform some user-defined action.

Oracle Rdb – A high-end client/server relational database for Alpha AXP and VAX.

Statistic Name – The valid statistic field for which the event is to be enabled or disabled.

Summary Event – A general-purpose event defined on a "summary" statistic field.

Threshold Name – The columns in the numeric version of the statistics screens for the specified statistic to be enabled or disabled.

PATROL Knowledge Module for Rdb

PATROL is a systems, applications, and event management tool for database and system administrators. It provides an object-oriented graphical workspace where you can view the status of every vital resource in the distributed environment that it is managing. PATROL monitors and manages the resources in the environment using information obtained from special files called Knowledge Modules (KM) that are loaded in the console. If PATROL detects a problem with a particular computer or application that it is monitoring, then these modules provide "knowledge" information that PATROL will use to attempt to fix the problem. If the problem escalates or requires human intervention, PATROL displays every resource affected by the problem in a warning or alarm condition. **Table 1** (see Appendix) includes a list of all the Rdb KM parameters that can be monitored. PATROL is made up of three major components: the PATROL Console, the PATROL Agent, and the Knowledge Modules.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Rdb Trace and Rdb Expert

Oracle Trace is a layered product that gathers and reports event-based data from any combination of OpenVMS layered products and application programs containing Oracle Trace service routine calls. Currently, the only way to perform gathering and reporting on Rdb databases on OpenVMS is through the Oracle Trace for OpenVMS layered product. Application programs that can contain Oracle Trace service routines are considered to be facilities that include the following products: DEC ACMS, DEC ALL-IN-1, DECforms, Oracle CODASYL DBMS, Oracle RALLY, and Oracle Rdb.

Oracle Trace provides Oracle Expert for Rdb along with data that it uses to optimize existing Rdb databases. Event-based data can be collected from products that contain Oracle Trace service routine calls. In addition, Oracle Trace routine calls can be added to other user-developed applications to collect data from them. The process of adding Oracle Trace service routine calls to an application is called *instrumenting the application*.

The Oracle Trace software operates with minimum performance impact on the system. It can run with both the development and production versions of your application to give you information about the behavior of your application.

Features of the Oracle Trace

Using Oracle Trace, the event data collected from applications can be used for different purposes, including the following:

- Tuning and performance improvements of applications
- Planning for hardware resources, (i.e., capacity planning, and so on)
- Tuning the performance of the databases
- Debugging applications
- Logging errors

Tuning the Performance of Databases

Oracle Trace provides request and transactional-level information from Oracle Rdb. This information allows a database administrator to examine a wide variety of performance statistics and usage information related to actual transactions and DML requests.

Oracle Trace provides RdbExpert information that RdbExpert uses to produce more efficient database designs. Database administrators no longer have to guess about the database workload because Oracle Trace collects actual workload information.

Transaction Processing

Oracle Trace tabular reports identify occurrences such as transaction with the higher virtual memory usage or the 95th percentile disk I/O for each transaction. For transaction processing, Oracle trace gathers information for DEC ACMS events to provide task-level performance information.

Relating Events Among Facilities

Oracle Trace allows the instrumentation of routines that use the cross-facility capability. For example, using the cross-facility, Oracle Trace associates the ACMS Procedure Call event with its related Oracle Rdb transactions and requests in order to provide a greater understanding of the total resources the ACMS Procedure Call uses.

Internally Developed Tool

The MSE `Rdb_Mon` utility is an nternally-developed application that makes use of the Oracle Rdb `RMU/SHOW STATISTICS` utility and adds alarming and notifications. Developed by MSE, this tool

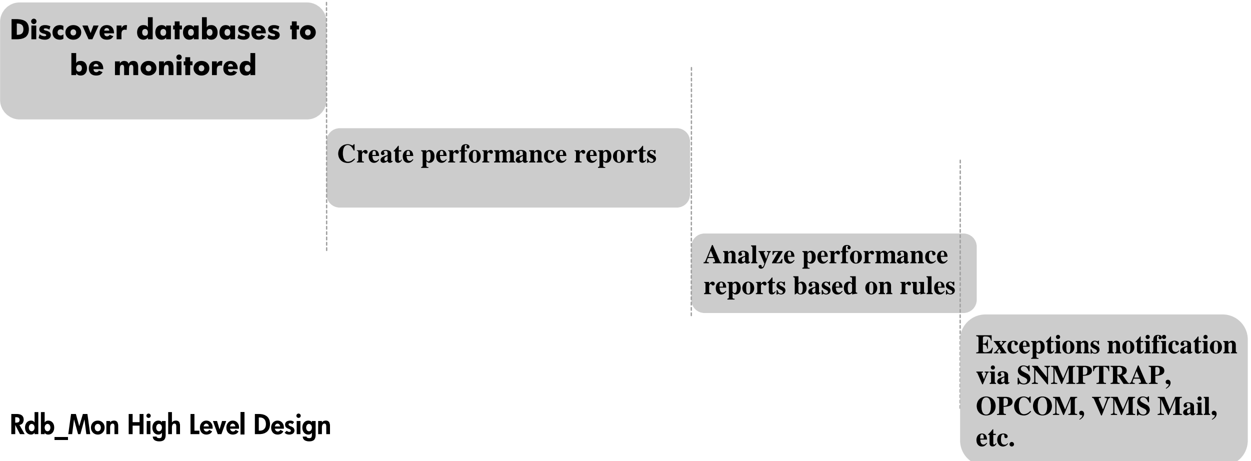
© Copyright 2006 Hewlett-Packard Development Company, L.P

contains a number of DCL procedures that can be adapted to the needs and requirements of the Oracle Rdb DBA.

Several forms of the `Rdb_Mon` utility currently exist. One form is a standalone configuration, comprising a number of DCL command procedures, where the tool is not integrated with any other enterprise management system. Another form integrates with an enterprise-wide management system. Currently, the latter from has been integrated with the Heroix Enterprise Management products RoboMon, RoboEDA, and so on. `Rdb_Mon` can be customized to be included and integrated under any other suite of enterprise management products.

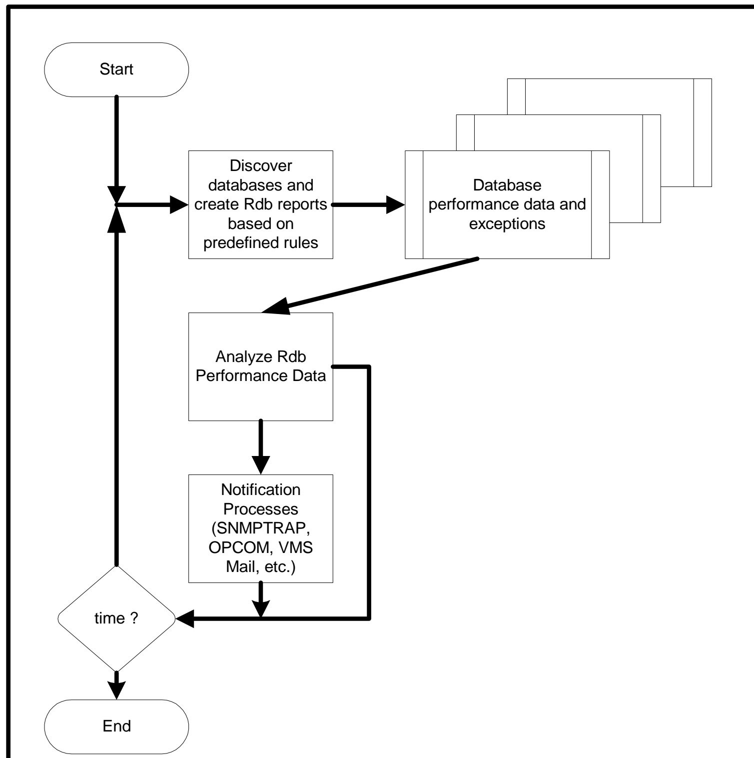
Database Monitoring and Alarming Basic Sequence

The following figure illustrates the sequence for monitoring databases:



Rdb_Mon High Level Design

The following figure shows the logic flow of the MSE-developed tools for the Vodafone Rdb Database Monitoring and Alarming project:



© Copyright 2006 Hewlett-Packard Development Company, L.P

Summary

This article described the monitoring and alarming capabilities of existing utilities and products for the Oracle Rdb database engine on OpenVMS platforms. It presented the MSE `Rdb_Mon` utility as one of the internally-developed alternatives for this technology sector.

References

http://www.oracle.com/technology/products/rdb/index.html - Oracle Rdb

http://documents.bmc.com/supportu/documents/55/16/5516/5516.pdf - PATROL Knowledge

Module for Rdb User Guide

Kostas G. Gavrielidis works in HP Services Customer Support and has been with HP for more than 20 years. Currently, and for the last 10 years, he is involved with the MSE proactive consulting projects for our customer production Database Management systems, and works on the analysis and performance improvements for SAP R/3, Oracle, Rdb, Ingres, SYBASE, SQL Server on UNIX, OpenVMS, and Windows platforms.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Appendix - Rdb KM parameters that can be monitored

Table 1: Rdb KM parameters that can be monitored

Parameter	Description
RDB_aij_reads	Displays the number of read QIOs issued to the database .AIJ file (if after-image journaling is enabled).
RDB_aij_writes	Displays the total number of QIOs issued to the database after-image journal file (if after-image journaling is enabled).
RDB_attaches	Displays the number of current attaches to the database.
RDB_blasts	Monitors the number of blocking AST's delivered to Rdb by the OpenVMS lock manager.
RDB_buf_unmark	This parameter is incremented each time a modified buffer is written back to disk. Its value is equal to the sum of the 14 fields: transaction, pool overflow, blocking AST, lock quota, lock conflict, user unbind, batch rollback, new area mode, larea change, incr backup, no aij access, truncate snaps, checkpoint, and aij_backup.
RDB_check_pts	Displays the current number of checkpoints per minute.
RDB_df_reads	Displays the number of read QIOs issued to the database storage area for a single-file and multifile databases and snapshot files.
RDB_df_writes	Displays the number of write QIOs issued to the database storage area for a single-file and multifile databases and snapshot files.
RDB_dup_nd_ins	Displays the number of duplicate index keys inserted into the database's indexes. There should be a one-to-one correspondence to the number of duplicate records being stored in the table.
RDB_fetch_read	Displays the number of synchronous data page requests to the PIO subsystem where only read privileges are being requested for the page.
RDB_fetch_upd	Displays the number of data page requests to the PIO subsystem where update and read privileges are being requested for the page.
RDB_free_global	This parameter displays the current percentage of free global buffers.
RDB_hash_del	Displays the number of hash key deletions from the database's hashed indexes. It includes unique key deletions and duplicate key deletions.
RDB_hash_dup_ins	Displays the number of duplicate hash key insertions in the database's hashed indexes.
RDB_hash_ins	Displays the number of hash key insertions in the database's hashed indexes. It includes unique key insertions and duplicate key insertions.
RDB_lck_conf_unmask	This parameter is incremented by 1 for each modified buffer that is written back to the disk to reduce the possibility of a deadlock when Rdb discovers a lock conflict.
RDB_lock_dem	Displays the number of $ENQ lock requests to demote an existing lock to a lower lock mode. These requests always succeed.
RDB_lock_req	Displays the number of lock requests to new locks. Whether the lock request succeeds or fails, it is included in this count.
RDB_overflow_unmark	This parameter is incremented by 1 for each modified buffer that is written back to disk as a result of a request to read in a new page from disk.
RDB_recoveries	Displays the current number of detached recovery (DBR) processes acting on this database.
RDB_rt_nd_rem	Displays the number of index entries removed from a root node because of deletion of entries within lower-level nodes. If an index consists of only one node, removals from this node are not included in this field; but are included in the leaf removals field.
RDB_rt_nd_ins	Displays the number of index entries inserted into the root index node. The

© Copyright 2006 Hewlett-Packard Development Company, L.P

	number of insertions should be small except when you load a database. If an index consists of only one node, insertions into this node are not included in this field; but are included in the leaf insertion field.
RDB_rt_reads	Displays the number of read QIOs issued to the database root (.RDB) file. Rdb reads the .RDB file when a new user attaches to the database and when an .RDB file control block needs to be updated because of database activity on another OpenVMS cluster node.
RDB_rt_writes	Displays the number of write QIOs issued to the database root (.RDB) file. Rdb writes to the .RDB file when a user issues a COMMIT or ROLLBACK statements. Other events also cause updates to the .RDB file.
RDB_ruj_reads	Displays the number of read QIOs issued to the database recovery unit journal (.RUJ) file. This operation reads before-image records from the .RUJ file to roll back a verb or a transaction.
RDB_ruj_writes	Displays the number of write QIOs issued to the database recovery unit journal (.RUJ) file. This operation writes before-image records to the .RUJ file in case a verb or a transaction must be rolled back. Before-image must be written to the RUJ file before the corresponding database page can be written back to the database.
RDB_trans_cnt	Displays the number of completed database transactions. It is the count of the COMMIT and ROLLBACK statements that have executed.
RDB_txn_unmark	This parameter is incremented by 1 for each modified buffer that is written back to disk as a result of a COMMIT or ROLLBACK statement.
RDBMON_attaches	Displays the number of current attaches to all databases on this system.
RDBMON_databases	Displays the number of open databases on this system.
RDBMON_recoveries	Displays the current number of detached database recovery (DBR) processes on this system.
RMU_stats	This parameter is the collector for all Rdb parameters.

For more information

For more information on this article and to make suggestions and comments for improvements, please email the author.

© Copyright 2006 Hewlett-Packard Development Company, L.P

OpenVMS Technical Journal V7

Faking it with OpenVMS Shareable Images

John Gillings, HP Customer Support Centre, Sydney Australia

Overview

Shareable images play a key role in OpenVMS. They are largely responsible for the legendary upwards compatibility across all versions by allowing run-time libraries (RTLs) to be updated while remaining binary compatible with existing program images. This mechanism can be exploited to provide a means for intercepting calls into shareable images, allowing black box diagnosis and debugging, selective modification of function, and a variety of other interesting applications.

This article discusses the theory and presents some DCL command procedures for analyzing and manipulating shareable images, and also for generating *fake* shareable image interfaces.

Caveats

Note that the techniques presented are NOT universal and may not work correctly for all shareable images. They are intended to be used for education, diagnosis and debugging, NOT for production use or in critical applications.

For the purposes of this article, only standard call interfaces on OpenVMS Alpha are considered. See the postscript for an enhancement that supports standard call interfaces on OpenVMS I64.

The procedure is user mode and requires no privilege. System shareable images can be faked by unprivileged users; however, the mechanism **cannot** be used to subvert the function of privileged images.

What is a Shareable Image?

Shareable images are collections of data and code that can be treated as a black box. Among other things, linking a program image involves resolving references to symbols (routines and data). Some references are resolved by including object code directly in the output image. Others are resolved from external libraries, ***shareable images***, which are activated at run time. Thus most images contain a list of shareable images against which it was linked. Those images, in turn may reference other shareable images. A program image may therefore be seen as a hierarchy of shareable images. Images linked against a shareable image know only about the routines and data exported from the shareable image. They have no knowledge of how the routines are implemented.

© Copyright 2006 Hewlett-Packard Development Company, L.P.

How Shareable Images Work

The shareable image interface is controlled by an entity called a *symbol vector* and a GSMATCH value and condition. These two, plus the image name, completely define the shareable image and control how a calling image can identify the correct shareable image. The symbol vector acts like a plug and socket. Provided the plug in the calling image fits the socket in the shareable image, the shareable image is activated.

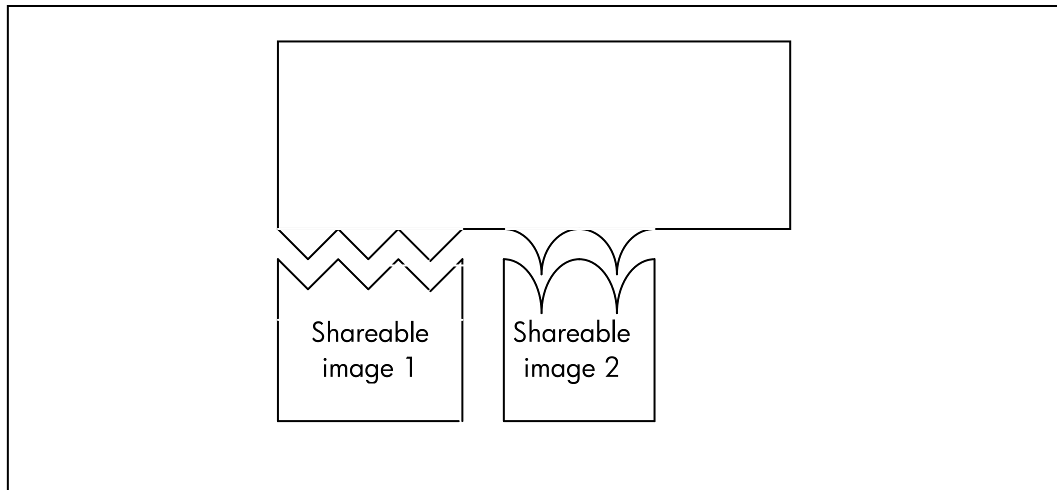


Figure 1 – Matching Interfaces

The calling image establishes the shapes of the interfaces of referenced shareable images at link time and expects the run-time shareable images to match.

Symbol Vectors and GSMATCH

The symbol vector consists of an ordered list of entries representing objects exported from the shareable image. Entry types considered in this article are PROCEDURE, CONSTANT, DATA CELL and SPARE. There are other types, but they are very rare and beyond the scope of this article. The Global Section MATCH (GSMATCH) is a pair of values and a matching rule that are considered when activating an image. Rules include ALWAYS, EQUAL, LEQUAL and NEVER. The pair of values is major ID and minor ID. When an image is linked, the GSMATCH values and rule of the target image are saved for comparison against a candidate image to be activated at run time. Rule ALWAYS causes the image activator to ignore the GSMATCH and always activate the image. Rule NEVER causes it to never activate an image (naturally this is rarely used in practice!). EQUAL requires both major and minor IDs to match exactly. The LEQUAL rule requires the major ID to match, but the minor ID of the candidate image may be greater than the expected value. The LEQUAL rule is the most commonly used match criterion, because it allows the implementation of upward compatible shareable images.

Image Activation

When an image is activated, the image activator extracts the list of referenced shareable images from the image header. The list gives the image name, the GSMATCH values and match criteria, and the symbol vector size. The name is used to locate the image. By default it is expected to be in directory SYS$SHARE with file type .EXE. The default can be overridden by defining the image name as a logical name with a full file specification. Once the shareable image file is found, the GSMATCH values are checked. If they pass the match criteria, the symbol vector size is compared. The symbol vector of the candidate image must be at least as large as the expected symbol vector size. If all of these tests pass, the shareable image is activated. If the newly activated image itself references shareable images, these are added to the image activator's list and the process of image activation continues.

Upwards Compatibility

Using GSMATCH=LEQUAL, it is easy to create shareable images that support upwards compatibility. If changes are made to the implementation of a shareable image, the new image will be compatible (plug interchangeable) with the old image provided there are no changes to the symbol vector. However if new entries are added to the end of the symbol vector, the new symbol vector will be

© Copyright 2006 Hewlett-Packard Development Company, L.P.

longer. So images linked against the old image can execute against old or new shareable images, but images linked against the new shareable image cannot execute against the old version (because they might reference the new symbol vector entries, which do not exist in the older image's symbol vector). By convention, the minor ID of the GSMATCH values for the new image should be incremented, so that the GSMATCH=LEQUAL match criteria detects the difference in the images. For example:

```
MYSHARE
GSMATCH=LEQUAL,100,5
SYMBOL_VECTOR=(F1=PROCEDURE,F2=PROCEDURE)
```

```
CALLING PROGRAM
Shareable image list
Name: MYSHARE, match: LEQUAL, Major:100, Minor: 5, vector size:2
```

Create a new version of the shareable image MYSHARE, adding procedure F3:

```
MYSHARE
GSMATCH=LEQUAL,100,6
SYMBOL_VECTOR=(F1=PROCEDURE,F2=PROCEDURE,F3=PROCEDURE)
```

A caller linked against the older version of MYSHARE passes the GSMATCH and vector size tests, and therefore, activates the image. A program linked against the new image has a different entry in the shareable image list:

```
CALLING PROGRAM
Shareable image list
Name: MYSHARE, match: LEQUAL, Major:100, Minor: 6, vector size:3
```

If this image attempts to activate the older version of MYSHARE, both the GSMATCH and symbol vector size tests fail, with an error similar to the following:

```
%DCL-W-ACTIMAGE, error activating image MYSHARE
-CLI-E-IMGNAME, image file DKA100:[SHARE]MYSHARE.EXE;1
-SYSTEM-F-SHRIDMISMAT, ident mismatch with shareable image
```

If for some reason the GSMATCH comparison succeeds, but the symbol vector is too short, the image fails to activate with an error message similar to the following:

```
%IMGACT-F-SYMVECMIS, shareable image's symbol vector table mismatch
-IMGACT-F-FIXUPERR, error when CALLPROG referenced MYSHARE
```

Dynamic Activation

Shareable images can also be activated dynamically under program control. The LIBRTL routine LIB$FIND_IMAGE_SYMBOL (LIB$FIS) accepts an image name, symbol name pair. It activates the image (if necessary) and returns the address of the object represented by the symbol. There are no GSMATCH or symbol vector checks.

Given the above, it is possible to *clone* the symbol vector and declare the same GSMATCH criteria and values of a given shareable image to create a fake image that can be substituted for the real image at run time. Further, we can implement the routines in the cloned image so they call the real routines in the real image, but we can also insert code at the beginning, at the end, or both to do whatever we like.

© Copyright 2006 Hewlett-Packard Development Company, L.P.

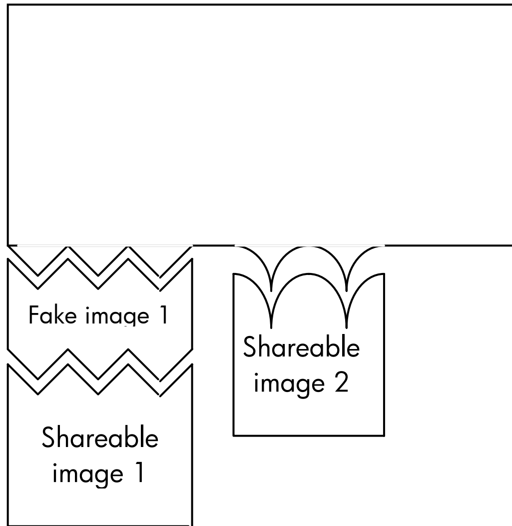


Figure 2 - Call Through a Fake Image

The calling image activates a compatible shareable image that intercepts all calls to the real shareable image.

Will the Real Image Please Stand Up?

If the fake image is activated, finding the real image is a problem. To distinguish the images, we need to give the real image a different name, so that the image activator thinks it's a different image. The simplest approach is to take a copy of the image and give it a prefix of REAL_. At run time, a logical name with the REAL_ prefix can be defined to locate the real image.

Because we'd like to be able to activate the fake image either with normal activation or dynamic activation, the symbols in the symbol vector must match those of the real image. Therefore, the fake image cannot call the real image directly, because the linker would detect duplicate symbols. Instead, we can dynamically activate the real image and use LIB$FIND_IMAGE_SYMBOL to locate the real routines. But LIB$FIND_IMAGE_SYMBOL is inside LIBRTL, and it's sometimes desirable to be able to create a fake LIBRTL, so we can't call LIB$FIND_IMAGE_SYMBOL directly from the fake image. Instead, we can create a FAKE_RTL shareable image in which the calls to other RTLs can be hidden. FAKE_RTL also contains other support routines for tracing argument lists and other functions. It is linked against REAL_LIBRTL to ensure it always calls the real LIB$FIND_IMAGE_SYMBOL.

© Copyright 2006 Hewlett-Packard Development Company, L.P.

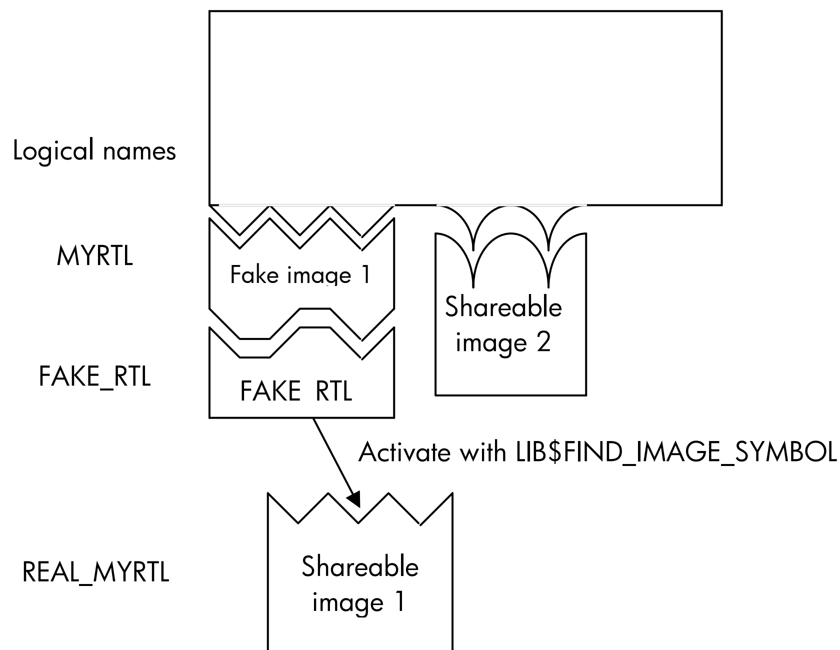


Figure 3 – FAKE_RTL Support Routines

Use FAKE_RTL to mediate between the fake and real shareable images. Images are located using the logical names as shown.

Resolving Self References

Consider a program calling routine LIB$FIND_IMAGE_SYMBOL in a fake LIBRTL. The image activator follows the logical name LIBRTL finding and activating FAKE_LIBRTL. FAKE_LIBRTL calls FAKE_RTL to find the address of LIB$FIND_IMAGE_SYMBOL in REAL_LIBRTL, using LIB$FIND_IMAGE_SYMBOL itself. FAKE_RTL then calls the real LIB$FIND_IMAGE_SYMBOL on behalf of FAKE_LIBRTL and returns the results to the caller.

But Why Would I Want to Do That?

OK, so it's possible to do this, but why would you want to? The mechanism is general. There are many possibilities. Here are some real world examples:

- A customer wanted to run COBOL programs with the clock offset arbitrarily forwards or backwards. A FAKE_COBRTL was generated that passed through all COBRTL calls except those dealing with dates and times. These were caught and the dates and times returned were offset according to a logical name.
- A customer program was apparently triggering a bug in SMGSHR, but attempts to reduce the customer's large program to a manageable reproducer were failing. By tracing the calls and argument lists to SMG$ routines through a FAKE_SMGSHR, it was possible to create a simple program, consisting of a sequence of SMG calls that demonstrated the bug.
- A suspected AST re-entrancy problem in a customer program was proven by creating a fake RTL for a key (3rd party) shareable image and inserting $SETAST calls before and after each call to block and restore AST interrupts while inside the shareable image. As well as proving the suspicions, this also provided a short term workaround while the shareable image was corrected.
- When debugging, it's sometimes desirable to set a break point at an RTL routine. If the shareable image containing the routine was not linked with /DEBUG, the symbol is not accessible. A fake image can be compiled and linked with /DEBUG and thereby make symbols available as breakpoints. Moreover, it is possible to break at calls from other code, including OpenVMS RTLs.

© Copyright 2006 Hewlett-Packard Development Company, L.P.

This mechanism is applicable for any case where you want to see what's going on inside a shareable image, or want to modify the function of some routines within the image, without having to re-implement the whole thing.

How it's Done

The command procedure FAKE_RTL.COM contains code to create all the necessary components for the FAKE_RTL environment and then analyze a given sharable image, generate a symbol vector and MACRO32 code to implement a template for the fake RTL.

Because the intention is for the user to be able to modify the fake RTL template for specific uses, the procedure is written as a sequence of phases, any of which can be executed independently.

Why MACRO?

Code is generated in MACRO32, because it's necessary to preserve all register values at all times. It's also much easier to manipulate argument lists without having access to the original definitions. MACRO32 doesn't have an RTL of its own, so there's no need to preclude specific RTLs from being candidates of FAKE_RTL (not entirely true, as the MOVC3 and CALLG instructions, both used in FAKE_RTL, are implemented by routines in LIBOTS – FAKE_RTL is therefore linked against REAL_LIBOTS to ensure the "real" routines are always used). The other advantage of MACRO is it is available on all OpenVMS systems without additional licenses.

Vector Entries

As mentioned previously, the four types of vector entries with which we're concerned are SPARE, CONSTANT, PROCEDURE and DATA CELL. For each object, we need to generate a symbol vector reference and some code to implement the object. As a quick overview:

- SPARE

  Shareable images may declare *empty* symbol vector slots for a variety of reasons. The existence of spare entries must be inferred from the relative spacing of other entries:

```
SYMBOL_VECTOR=(SPARE)
```

  No MACRO32 code is necessary:

- CONSTANT

  This is the simplest case of a real object, the symbol vector entry is:

```
SYMBOL_VECTOR=(<symbol-name>=DATA)
```

  MACRO32 code is a global symbol definition:

```
<symbol-name>==<value>
```

  Note that the fake image could export a constant value different from the value exported from the real image. The FAKE_RTL procedure generates the real value in the template source code, but it can easily be modified.

- PROCEDURE

  The majority of entries are procedure calls. The symbol vector entry is:

```
SYMBOL_VECTOR=(<symbol-name>=PROCEDURE)
```

  MACRO32 code is a call to a macro:

```
CallRoutine <symbol-name>
```

  Details of this macro and alternatives are discussed later. In practice, what the macro does depends on why you want to create a fake shareable image. By default FAKE_RTL.COM generates a macro that writes a log file, tracing each call into the shareable image, including argument list and register values.

- DATA CELL

© Copyright 2006 Hewlett-Packard Development Company, L.P.

Data is hard to deal with, and in some cases intractable. That's because there is no mechanism in OpenVMS to *jacket* a data reference. Exactly how to implement data depends on how it is used in both the caller and the real shareable image. The symbol vector entry is:

```
SYMBOL_VECTOR=(<symbol-name>=DATA)
```

By default, the MACRO32 code generated attempts to declare data cells that match the original in size and location relative to other data cells:

```
<symbol-name>:: .BLKB <object size>
```

At run time, the data region containing these declarations is mapped to the corresponding locations in the real shareable image. Although this works for many uses of exported data, it's not universal. Examples of where this might not work, and some potential workarounds are discussed later.

Walkthrough FAKE_RTL.COM

There are seven phases to creating a fake shareable image. They are:

1. DEF - Define fake RTL environment
2. COPY - Make a copy of the original shareable image
3. VECTOR - Analyze the symbol vector
4. GEN - Generate MACRO code
5. COMPILE - Compile MACRO code
6. LINK - Link fake RTL
7. USE - Define logical names to use image

The procedure is invoked, optionally giving the name of a shareable image, a start phase and options. By default, the named phase and all subsequent phases are executed. If no phase is given, DEF is assumed, but only those phases that appear to have not been executed are performed. If no parameters are given, the environment is defined. For example:

```
$ @FAKE_RTL
```

- Defines logical names necessary for running a fake shareable image, including any logical names and symbols for using fake images already created. It also creates FAKE_RTL.EXE if it does not exist.

```
$ @FAKE_RTL SMGSHR
```

- Creates a fake version of SMGSHR if it does not already exist. Defines logical names and symbols for using FAKE_SMGSHR if it already exists.

```
$ @FAKE_RTL SMGSHR COMPILE
```

- Compiles and links FAKE_SMGSHR, then defines logical names and symbols to use it.

```
$ @FAKE_RTL SMGSHR GENERATE DEBUG
```

- Generates MACRO32 code for FAKE_SMGSHR, then compiles and links FAKE_SMGSHR with DEBUG.

```
$ @FAKE_RTL LIBRTL VECTOR FORCE
```

- Analyzes LIBRTL.EXE, generates MACRO32 code, compiles and links FAKE_LIBRTL. The FORCE option causes any existing versions of FAKE_LIBRTL to be overwritten.

FAKE_RTL.COM Phases in Detail

The following list describes FAKE_RTL.COM phases:

- **DEF - Define fake RTL environment**
  This phase defines the logical name FAKE_DIR to point to the directory containing FAKE_RTL routines and procedures. If the shareable image FAKE_RTL.EXE does not exist, it is created. FAKE_DIR is then scanned for files called REAL_<name>.EXE. A logical name is defined for each one found.
- **COPY - Make a copy of the original shareable image**

© Copyright 2006 Hewlett-Packard Development Company, L.P.

The copy is called REAL_<imagename> and placed in the directory containing FAKE_RTL

- **VECTOR - Analyze the symbol vector**
  The input image is analyzed using ANALYZE/IMAGE and then filtered using SEARCH to extract the GSMATCH values and symbol vector entries. The GSMATCH information looks like this.

```
image type: shareable (EIHD$K_LIM)
  global section major id: %X'42', minor id: %X'000017'
  match control: ISD$K_MATLEQ
```

It would be transformed into a Linker options definition:

```
GSMATCH=LEQUAL,%X42,%X17
```

The three symbol vector entry types look like this:

```
value: 16 (%X'00000010')
symbol vector entry (constant)
      %X'00000000 00000000'
      %X'00000000 00358014' (3506196)
symbol: "C$_ENOENT"

value: 1248 (%X'000004E0')
symbol vector entry (procedure)
      %X'00000000 0009BD40'
      %X'00000000 000D5038'
symbol: "LIB$WAIT"

value: 30272 (%X'00007640')
symbol vector entry (data cell)
      %X'00000000 00000000'
      %X'00000000 002502C4'
symbol: "DECC$GA_TZNAME"
```

The *value* field is the offset to the entry in the symbol vector. For constants, the second symbol vector entry value is the value of the constant. For procedures, the symbol vector entry is the procedure descriptor. For data cells, the second symbol vector entry value is the offset to the data cell from the start of the shareable image. These entries are parsed and written to a file one line per entry as:

```
<value> <vector value 1> <vector value 2> <symbol> <type>
```

Data cell values are written to a second output file as well:

```
<vector value 2> <symbol>
```

The output files are then sorted. The vector list is then in symbol vector order. The data list is in allocation order.

© Copyright 2006 Hewlett-Packard Development Company, L.P.

- **GEN - Generate MACRO code**
  This phase uses the ordered vector and data lists from the VECTOR phase to generate the symbol vector (as a linker options file) and MACRO32 code. For each vector entry we generate a symbol vector declaration and some code. Before writing the symbol vector declaration, we first check for holes. Because all entries are 16 bytes, this is a simple matter of keeping track of the next expected entry value. Any missing entries are filled with SPARE entries. Constants are dealt with as described previously.

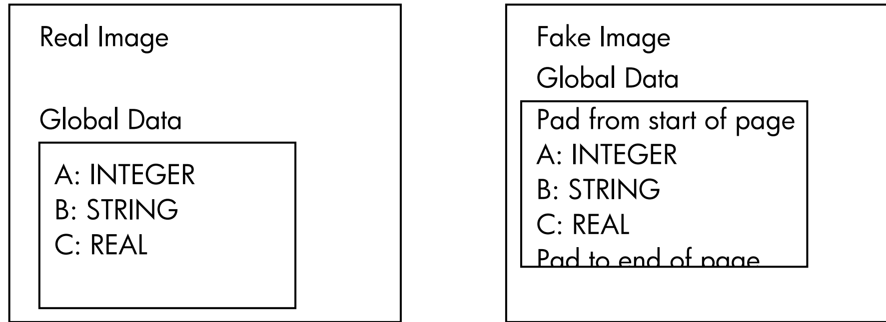
  Procedures generate a call to a macro. The default macro generates a memory location to store the symbol name and another to store the symbol address (initialized to 0). These are used in a call to routine FAKE_LOGCALL in FAKE_RTL, passing the name of the real RTL, the name of the symbol, the symbol address and a pointer to the homed argument list.
  FAKE_LOGCALL will check the symbol address. If zero, it uses LIB$FIND_IMAGE_SYMBOL to locate the routine. The routine is then called, passing the argument list. If the logical name FAKE_DUMPARGS is defined, FAKE_LOGCALL logs the routine name, argument list, and register values to a file called ARGDUMP.LOG.

  In some cases, the shareable image routine should be called using an OpenVMS VAX style JSB mechanism. In these cases, the FAKE_RTL routine FAKE_LOGJSB should be called instead. The OpenVMS naming convention is to name JSB routines with a trailing _Rn, where n is the highest register number used by the routine. FAKE_RTL.COM recognizes this convention and uses a JSB macro instead of a CALL macro. However, this is NOT universal. If your target shareable image has JSB routines, you need to manually modify the generated code to use the correct call semantics.

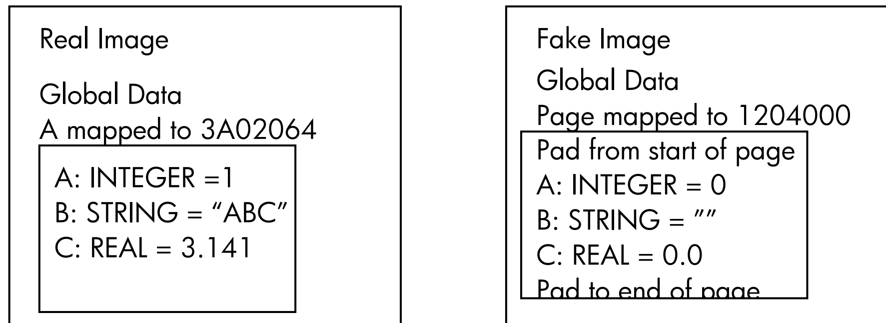
  When processing vector entries, data cell entries generate a symbol vector declaration, but no code. Once the vector is complete, the data list is processed (if it exists). Data declarations are processed in allocation order. They are placed in a page-aligned PSECT, with the first entry aligned to the same page offset as the entry in the real shareable image. Objects are assumed to be the inferred size of the difference between adjacent declarations.
  However, if a gap between declarations exceeds an Alpha page, a new block of data is started. For each block, the name of the first symbol is saved, and a private routine MapData is generated to be called when the image is activated.

  At run time, MapData calls the FAKE_RTL routine FAKE_MAP_DATA for each data block, given the name of the real shareable image, a generated global section name, the name of the first symbol in the block and the size, and start and end addresses of the block. FAKE_MAP_DATA first deletes the virtual memory for the block, then creates and maps a global section. It then copies the contents of the corresponding memory locations from the real shareable image, deletes the virtual memory in the real shareable image, and maps the locations to the global section. The result is two address ranges that map the same physical data. See Figure 4 for an illustration of this process. This is an imperfect solution. Potential problems are discussed later.

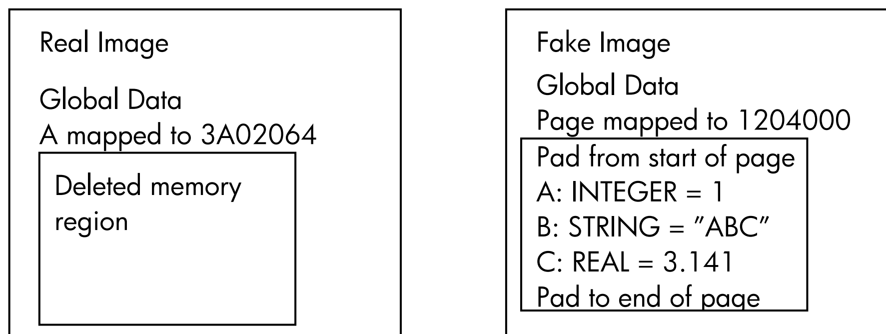
© Copyright 2006 Hewlett-Packard Development Company, L.P.



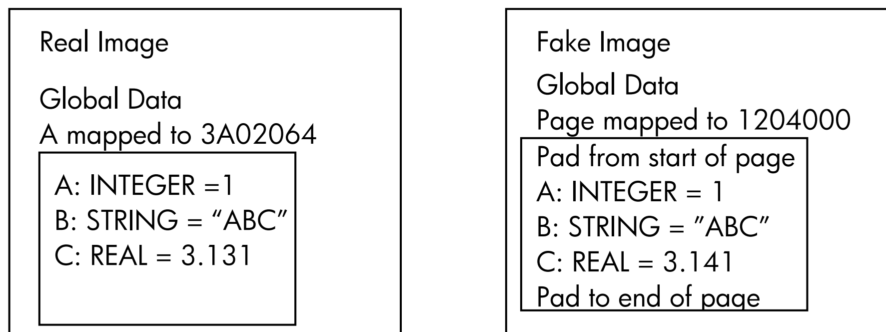
Creation time – Fake image has data declared in whole pages, aligned and allocated to match the data in the real image.



Initial run time – Real image has data values and both images are mapped to real addresses. The page in the fake image at 1204000 is deleted ($DELTVA) and a global section created from 1204000:1205FFF



Data from 3A02000:3A03FFF (entire page surrounding the real data) is copied to the global section at 1204000. The page at 3A02000 is then deleted ($DELTVA). The fake image now has a replica of the real data



Address range 3A02000:3A03FFF is now mapped to the global section. Variable A is now accessible for read/write access from both its "real" address, 3A02064, and the "fake" address, 1204064

Figure 4 – Mapping Shared Data

© Copyright 2006 Hewlett-Packard Development Company, L.P.

COMPILE - Compile MACRO code

If you make changes to the generated template, you must recompile the MACRO32 source code. The DEBUG option causes the compilation to be /NOOPTIMIZE/DEBUG.

LINK - Link fake RTL

The FAKE_imagename image is linked using the options file generated in the GEN phase. The DEBUG option causes /DEBUG to be added to the LINK command.

USE - Define logical names to use image

A logical name REAL_imagename is defined to point to the image created in the COPY step:

```
$ DEFINE REAL_LIBRTL DISK:[DIR]LIBRTL
```

You can redefine this point to the original image:

```
$ DEFINE REAL_LIBRTL SYS$SHARE:LIBRTL
```

However, in the case of installed resident images, especially those with shared address data, this might result in bypassing some defined "fake" images. For most cases, it's best to use the renamed copy created in the COPY phase. Two global symbols are also defined:

```
$ FAKE_imagename=="$DEFINE/USER/NAME=CONFINE imagename
FAKE_DIR:FAKE_imagename"
```

and:

```
$ REAL_imagename=="$DEASSIGN imagename"
```

These symbols are intended to simplify enabling a fake shareable image. Note that /USER is specified so the definition is automatically cancelled after the next image activation. This is a precaution to prevent surprises. /NAME=CONFINE prevents the logical name from being propagated to subprocesses. This may be necessary, for example, when running an image under DEBUG, as you probably don't want the DEBUG subprocess to run through your fake shareable image.

Does it work?

Well, it does mostly. For shareable images that export only constants and procedures, the only potential problem is JSB routines, which are comparatively rare. The most likely symptom of a JSB routine called incorrectly is when logging argument lists, seeing an obviously incorrect (very high) argument count. It seems that sometimes these high values can upset the argument homing mechanism, resulting in ACCVIOs.

However, images that export data don't always work. In some cases, the data is constant. For example, LIBRTL exports a number of character conversion tables. These work correctly with the remapping mechanism. In other cases the data is used in ways that are incompatible with the mapped section solution implemented by default.

For example, PAS$RTL exports 3 file variables: PASFV_INPUT, PASFV_OUTPUT and PAS$FV_ERR, being standard input, standard output and standard error. Although the double map mechanism correctly references the variables, PAS$RTL code compares addresses of file variables passed to I/O routines to detect use of the standard files. Because the caller is passing via the fake addresses, the variables aren't recognized.

One way to work around this type of usage is to scan the argument lists of potentially affected routines, looking for addresses in the range of the fake data blocks. If any are found, calculate the offset and adjust it to be in the real address range. This is very ugly, but it works for PAS$RTL (the task is made a bit easier because there's only one potential argument affected, and the I/O routines are easily identified).

Case-sensitive symbol names can't be defined in MACRO32, so any shareable images that export lowercase symbols can cause trouble. For example, DECC$SHR exports each symbol in both

© Copyright 2006 Hewlett-Packard Development Company, L.P.

uppercase and lowercase forms. Without some modification, this would generate duplicate symbols in MACRO32. The GEN phase identifies lowercase symbols and defines them using a prefix L_, truncating the symbol name if necessary. Because there is no reason to LINK against the FAKE_imagename image, this does not cause any trouble for a caller activating the image normally. But an attempt to activate the image with LIB$FIND_IMAGE_SYMBOL using a lowercase, case-sensitive symbol name fails because the expected symbol does not exist.

Examples

Under OpenVMS V7.3-2 and V8.2, the following OpenVMS RTLs and shareable images can be converted automatically and appear to work as expected:

- DEC$BASRTL
- DEC$COBRTL
- DEC$FORRTL
- FDLSHR
- LIBRTL
- LBRSHR
- MAILSHR
- SMGSHR
- SORTSHR
- TPUSHR

DECC$SHR seems to work for some simple programs, but fails with more complex programs. This may be a JSB routine issue or a data usage issue. Debugging is rather difficult. Use with extreme care. LIBOTS doesn't work at all. This appears to be some form of recursion trying to call OTS$CALL_PROC using itself, which is even more difficult to debug.

In testing while writing this article, a small bug was found in the MAIL utility. MAIL attempts to find the symbol TPU$_NONANSICRT in TPUSHR to correctly report an attempt to edit from a non-ANSI terminal. A typo in the source code defined the symbol name incorrectly as TPU$_NOANSICRT. As a result, when an attempt was made to use the editor from a non-ANSI terminal, MAIL did not report the error, it just failed to send the message:

```
MAIL> send/edit
To:     _gillings
Subj:   test
%MAIL-E-SENDABORT, no message sent
MAIL> *EXIT*
```

To confirm the diagnosis, a FAKE_TPUSHR was created, and the symbol name and definition TPU$_NONANSICRT was changed to TPU$_NOANSICRT so that the call from MAIL would be correct. When running with this modified FAKE_TPUSHR, the behaviour of MAIL changed to the following, as intended:

```
$ mail


You have 1 new message.


MAIL> send/edit
To:     _gillings
Subj:   test
%TPU-E-NONANSICRT, SYS$INPUT must be supported CRT
%MAIL-E-SENDABORT, no message sent
MAIL> *EXIT*
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

This demonstrates the ability to quickly test and workaround some classes of bug, even when the incorrect source code is unavailable. It also proves that the proposed solution (fixing the symbol name) corrects the problem. (This bug has been reported to OpenVMS engineering, and is now fixed).

Postscript – Port to OpenVMS Integrity Server

Because the routines and mechanisms involved are deeply involved in the calling standard, which has been changed to accommodate the Integrity server platform, it was assumed that porting this utility to OpenVMS Integrity server (I64) would be difficult, hence the caveat about Alpha Only.

Preliminary investigation showed that the FAKE_RTL shareable image worked correctly with no changes. A straight forward "compile and go". All that was necessary was to teach the vector generation phase of FAKE_RTL.COM how to interpret an I64 image analysis. The output of `ANALYZE/IMAGE/SECTION=SYMBOL_VECTOR` was already much closer to the required format than the Alpha equivalent. Also the I64 qualifier `/NOPAGE_BREAK` simplified parsing by eliminating page breaks. The I64 implementation lists the symbol vector completely, whereas it needs to be reconstructed by inference from an Alpha image analysis. One consequence of this was the vector listing now contains explicit "SPARE" vector entries, so the GEN phase required extra logic to recognize the entries, rather than inferring them. The only other change required was to change the sanity-check logic to know that I64 vector entries are 8 bytes, and Alpha vector entries are 16 bytes.

In hindsight, the VECTOR phase is almost superfluous given the output of ANALYZE/IMAGE on I64. A better design might be to change the Alpha VECTOR phase to match the I64 format, rather than processing the I64 output to match what was implemented for Alpha.

The port was completed in less than one hour, yielding working FAKE_LIBRTL, FAKE_DECC$SHR, FAKE_MAILSHR, FAKE_SMGSHR and FAKE_UTIL$SHARE images. The command procedure published with this article in Appendix F is the updated Alpha and I64 version. As a minor addition, it also outputs an *unsupported* error message if executed on any other platform.

The FAKE_RTL command procedure is located at the following URL:

http://h71000.www7.hp.com/openvms/journal/v7/fake_rtl_com.txt

© Copyright 2006 Hewlett-Packard Development Company, L.P.

Appendix A: FAKE_RTL Routines

The following are routines exported by FAKE_RTL:

FAKE_DOCALL(! dispatch to routine by CALL
imagename readonly string descriptor,
symbolname readonly string descriptor,
routineaddress modify longword reference
arglist readonly argument vector reference):returns longword

FAKE_LOGCALL(! log argument list and dispatch by CALL
imagename readonly string descriptor,
symbolname readonly string descriptor,
routineaddress modify longword reference
arglist readonly argument vector reference):returns longword

FAKE_DOJSB(! dispatch to routine by JSB
imagename readonly string descriptor,
symbolname readonly string descriptor,
routineaddress modify longword reference
r16,r17,r18,r19,r20,r21 readonly quadword immediate value):returns longword

FAKE_LOGJSB(! log argument list and dispatch by JSB
imagename readonly string descriptor,
symbolname readonly string descriptor,
routineaddress modify longword reference
r16,r17,r18,r19,r20,r21 readonly quadword immediate value):returns longword

FAKE_PUT(! write a string to log file (unconditional)
string readonly string descriptor)

FAKE_FIS(! jacket for LIB$FIND_IMAGE_SYMBOL
imagename readonly string descriptor,
symbolname readonly string descriptor,
routineaddress modify longword reference)

FAKE_CALL(! jacket for CALLG (OTS$EMUL_CALL)
routineaddress readonly longword reference
arglist readonly argument vector reference):returns longword

© Copyright 2006 Hewlett-Packard Development Company, L.P.

FAKE_MOVE(! jacket for MOVC3
bytes readonly longword immediate value
source readonly bytearray reference
destination writeonly bytearracy reference)

FAKE_OUT(! jacket for LIB$PUT_OUTPUT
string readonly string descriptor)

FAKE_LOG(! write to log file if logging enabled
string readonly string descriptor)

FAKE_MAP_DATA(! map data area
imagename readonly string descriptor,
sectionname readonly string descriptor,
symbolname readonly string descriptor,
pages readonly longword immediate value
startaddress readonly address immediate value
endaddress readonly address immediate value):returns address

© Copyright 2006 Hewlett-Packard Development Company, L.P.

Appendix B: Logging Argument Lists

Logging calls and argument lists is just one application of this mechanism. As it's of general use, logging is implemented as the default function of a fake RTL. Because there is no information about argument lists available, this implementation of logging is *best guess* based on the actual arguments. RMS services are used to avoid dependence on language RTLs.

Logging starts with the routine name and a time stamp, followed by a dump of general registers R0 through R11.

The first test is to see if the argument list is readable at all. If not readable, it's considered a zero list. If readable, the argument count is determined and logged.

For each argument, the value is examined. If it's not a readable address, the argument is assumed to have been passed by immediate value and is displayed like arguments 3 and 6 through 10 in Figure 5.

```
LIB$CREATE_VM_ZONE at 15:33:46.66
R0:00000000 R1:010E0009 R2:00B9AA00 R3:00000007 R4 :00124B60 R5: 00090020
R6:00000000 R7:00BC8024 R8:00000000 R9:00090020 R10:7FFA4F28 R11:7FFCDBE8
LIB$CREATE_VM_ZONE called with 11 args
   1 00BC8020 => 00000000
   2 7AD496B0 => 00000001
   3 00000000
   4 7AD496B8 => 000000A1
   5 00000000
   6 00000000
   7 00000000
   8 00000000
   9 00000000
  10 00000000
  11 7AD49630 => 010E0009
                 00BA8750 => SMG$_ZONE
LIB$CREATE_VM_ZONE returning 11 args
   1 00BC8020 => 007D0800
   2 7AD496B0 => 00000001
   3 00000000
   4 7AD496B8 => 000000A1
   5 00000000
   6 00000000
   7 00000000
   8 00000000
   9 00000000
  10 00000000
  11 7AD49630 => 010E0009
                 00BA8750 => SMG$_ZONE
LIB$CREATE_VM_ZONE returned: 00000001 at 15:33:46.66
R0:00000001 R1:0000C3A5 R2:00B9AA00 R3:00000007 R4 :00124B60 R5: 00090020
R6:00000000 R7:00BC8024 R8:00000000 R9:00090020 R10:7FFA4F28 R11:7FFCDBE8
```

Figure 5 – Sample Logged Call and Argument List

If the argument is a readable address, the value referenced is displayed, like arguments 1, 2 and 4 in Figure 5. The address is also passed to LIB$CVT_DX_DX with a valid output string descriptor. If the argument is a valid scalar descriptor, CVT_DX_DX converts it to a string, which can then be displayed like argument 11 in Figure 4.

If the descriptor test fails, the address is then scanned for printable characters. If any are found, the argument is assumed to be a string by reference or null terminated string, which is displayed, like argument 1 to DECC$GETENV in Figure 6. Obviously, this test has the potential to "run away" on a

© Copyright 2006 Hewlett-Packard Development Company, L.P.

very long string, so the string is limited to a displayable length (see symbol *maxstr* in the source code FAKE_RTL.MAR).

When the routine returns, the argument list is logged a second time, the routine name, its return value and a time stamp are logged, and the general registers are dumped again. This shows any changes to arguments resulting from the call, for example, argument 1 in Figure 5.

```
DECC$GETENV called with 1 arg
   1 00010790 => 24554644
               =/DFU$NOSMG/

  LIB$GET_SYMBOL at 08:13:44.47
  R0:00000000 R1:00000001 R2:00A27A18 R3:00010790 R4 :00000009 R5: 00000001
  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:7FFA4F28 R11:7FFCDBE8
  LIB$GET_SYMBOL called with 4 args
     1 7AD492F8 => 010E0009
                   00010790 => DFU$NOSMG
     2 7AD492C8 => 010E0401
                   7AD49310 =>        /      q              °€  ÿÿÿÿ°Ð<
     3 7AD49300 => 00000000
     4 00000000
  LIB$GET_SYMBOL returning 4 args
     1 7AD492F8 => 010E0009
                   00010790 => DFU$NOSMG
     2 7AD492C8 => 010E0401
                   7AD49310 => TRUE
     3 7AD49300 => 00000004
     4 00000000
  LIB$GET_SYMBOL returned: 00000001 at 08:13:44.47
  R0:00000001 R1:00000004 R2:00A27A18 R3:00010790 R4 :00000009 R5: 00000001
  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:7FFA4F28 R11:7FFCDBE8

  (Calls to other LIB$ routines omitted)

DECC$GETENV returning 1 arg
   1 00010790 => 24554644
               =/DFU$NOSMG/
DECC$GETENV returned: 00B2CC40 at 08:13:44.47
R0:00B2CC40 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:7FFA4F28 R11:7FFCDBE8
```

Figure 6 – Sample Logged Call Showing Nesting

Because logged routines may call other fake RTL routines, logging keeps track of the nesting level and indents nested calls. For example, Figure 6 shows the CRTL routine *getenv* calls LIB$GET_SYMBOL to obtain the symbol value from DCL. Note that argument 2 to LIB$GET_SYMBOL (the output string) contains junk on input. The descriptor is for a string 1025 characters long, but only the first 48 characters are displayed.

Because this algorithm knows nothing of the actual argument declaration and usage, it's subject to error. It frequently displays an argument as a string, just because the bytes look like printable ASCII characters. The philosophy is to try to show as much as possible. Obviously, the code implemented in FAKE_RTL can be replaced by a more sophisticated algorithm, or one with more direct knowledge of the target shareable image.

© Copyright 2006 Hewlett-Packard Development Company, L.P.

Appendix C: Sample Run

The following is a transcript from a terminal session showing initial execution of FAKE_RTL.COM to generate FAKE_RTL.EXE. The procedure is then used to create fake RTLs for SMGSHR, LIBRTL and DECC$SHR.

Note that the warnings listed for LIBRTL and DECC$SHR are normal. They show holes in the symbol vector that are filled with SPARE entries. LIBRTL also contains several JSB routines, identified by the naming convention of suffixing the routine name with _Rn.

```
$ @fake_rtl
Defining FAKE_RTL environment
%CREATE-I-CREATED, USER$TSC:[GILLINGS.FAKE_RTL]FAKE_RTL.MAR;1 created
%CREATE-I-CREATED, USER$TSC:[GILLINGS.FAKE_RTL]FAKE_RTL.OPT;1 created
Compiling FAKE_RTL.MAR
Linking FAKE_RTL.OBJ
$
$ @fake_rtl smgshr
%COPY-S-COPIED, SYS$COMMON:[SYSLIB]SMGSHR.EXE;1 copied to
USER$TSC:[GILLINGS.FAKE_RTL]REAL_SMGSHR.EXE;1 (593 blocks)
Generating symbol vector for SMGSHR
%ANALYZE-I-ERRORS, USER$TSC:[GILLINGS.FAKE_RTL]REAL_SMGSHR.EXE;1 0
errors
Generating MACRO code for SMGSHR
Compiling FAKE_SMGSHR
Linking FAKE_SMGSHR
$
$ @fake_rtl librtl
Generating symbol vector for LIBRTL
%ANALYZE-I-ERRORS, USER$TSC:[GILLINGS.FAKE_RTL]REAL_LIBRTL.EXE;1 0
errors
Generating MACRO code for LIBRTL
Assumed JSB routine LIB$ANALYZE_SDESC_R2
Warning - vector hole before LIB$CVT_DTB. Expecting 272, got 288
Warning - vector hole before LIB$DELETE_FILE. Expecting 336, got 352
Warning - vector hole before LIB$EXTV. Expecting 432, got 448
Warning - vector hole before LIB$GET_COMMAND. Expecting 544, got 576
Warning - vector hole before LIB$INDEX. Expecting 608, got 624
Warning - vector hole before LIB$LOCC. Expecting 656, got 672
Warning - vector hole before LIB$SCANC. Expecting 848, got 864
Assumed JSB routine LIB$SGET1_DD_R6
Warning - vector hole before LIB$TRA_ASC_EBC. Expecting 1168, got 1184
Assumed JSB routine OTS$$CVT_D_T_R8
Assumed JSB routine OTS$$CVT_F_T_R8
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
Assumed JSB routine OTS$$CVT_G_T_R8
Assumed JSB routine OTS$$CVT_H_T_R8
Assumed JSB routine OTS$MOVE3_R5
Assumed JSB routine OTS$MOVE5_R5
Assumed JSB routine OTS$SGET1_DD_R6
Assumed JSB routine STR$ANALYZE_SDESC_R1
Assumed JSB routine STR$COPY_DX_R8
Assumed JSB routine STR$COPY_R_R8
Assumed JSB routine STR$FREE1_DX_R4
Assumed JSB routine STR$GET1_DX_R4
Assumed JSB routine STR$LEFT_R8
Assumed JSB routine STR$LEN_EXTR_R8
Assumed JSB routine STR$POSITION_R6
Assumed JSB routine STR$POS_EXTR_R8
Assumed JSB routine STR$REPLACE_R8
Assumed JSB routine STR$RIGHT_R8
Assumed JSB routine OTS$$RET_A_CVT_TAB_R1
Warning - vector hole before LIB$EMUL. Expecting 2896, got 2912
Warning - vector hole before LIB$REMQHI. Expecting 3104, got 3120
Warning - vector hole before STR$ADD. Expecting 3392, got 3680
Assumed JSB routine STR$ELEMENT_R8
Warning - vector hole before LIB$TABLE_PARSE. Expecting 4496, got 4576
Warning - vector hole before LIB$FIND_VM_ZONE. Expecting 4608, got
4640
Start of data block 1 00000000000D6EB0 LIB$AB_ASC_EBC
Compiling FAKE_LIBRTL
Linking FAKE_LIBRTL
$
$ @fake_rtl decc$shr
%COPY-S-COPIED, SYS$COMMON:[SYSLIB]DECC$SHR.EXE;1 copied to
USER$TSC:[GILLINGS.FAKE_RTL]REAL_DECC$SHR.EXE;1 (4483 blocks)
Generating symbol vector for DECC$SHR
%ANALYZE-I-ERRORS, USER$TSC:[GILLINGS.FAKE_RTL]REAL_DECC$SHR.EXE;1 0
errors
Generating MACRO code for DECC$SHR
Warning - vector hole before decc$bsd__cputchar. Expecting 22784, got
22800
Warning - vector hole before DECC$CONFSTR. Expecting 34736, got 34800
Warning - vector hole before decc$readdir_r. Expecting 46128, got
46224
Warning - vector hole before decc$poll. Expecting 49776, got 50000
Start of data block 1 00000000001F0000 DECC$GA_BSD_AE
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
Hole in data exceeded threshold at 00000000001F4E18 DECC$$GA_IO_BLOCK
Start of data block 2 00000000001F4E18 DECC$$GA_IO_BLOCK
Compiling FAKE_DECC$SHR
Linking FAKE_DECC$SHR
$
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

Appendix D: Sample Use

Enable argument logging by defining the logical name fake_dumpargs, then enable the use of the fake DECC$SHR, LIBRTL and SMGSHR and activate the freeware utility DFU.

```
$
$ define fake_dumpargs "TRUE"
$ fake_decc$shr
$ fake_librtl
$ fake_smgshr
$ mcr dfu
```

```
+------------------------< DFU V2.7 >------------------------------+
|                                                                   |
|     Disk and File Utilities for OpenVMS DFU V2.7                  |
|     Internal Use Only!                                            |
|     Copyright © 2000 COMPAQ Computer Corporation                  |
|                                                                   |
|     DFU functions are :                                           |
|                                                                   |
|     DEFRAGMENT : Defragment files or disks                        |
|     DELETE     : Delete files by File-ID; delete directory (trees)|
|     DIRECTORY  : Manipulate directories                           |
|     INDEXF     : Modify /View INDEXF.SYS                          |
|     REPORT     : Generate a complete disk report                  |
|     SEARCH     : Fast file search                                 |
|     SET        : Modify file attributes                           |
|     UNDELETE   : Recover deleted files                            |
|     VERIFY     : Check and repair disk structure                  |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|
|| -----------------------Statistics-------------------------------|
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
|                                                                     |
|
|
||
+-------------------------------------------------------------+

DFU>  Exit
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

Appendix E: Sample Output

This (long) listing shows the beginning and end of the argument dump from the activation of DFU. A large chunk is omitted from the middle.

```
$ type/page argdump.log

Arg tracing started at 17-MAY-2005 15:33:46.62

Mapped Data LIBRTL1_DATA real:004EC000:004EFFFF =>
fake:00332000:00335FFF

Mapped Data DECC$SHR1_DATA real:00A5E000:00A5FFFF =>
fake:00604000:00605FFF

Mapped Data DECC$SHR2_DATA real:00A62000:00A63FFF =>
fake:00606000:00607FFF


DECC$MAIN at 15:33:46.65

R0:7FFCF87C R1:00002000 R2:00010830 R3:7AF08EB2 R4 :7FFCF814 R5:
7FFCF934

R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

DECC$MAIN called with 9 args

   1 7FFCF884 => 00010830

                =/0/

   2 7AE63EC8 => 00303089

   3 7FFCF814 => 00000003

   4 7FFCF934 => 001C0048

                =/H/

   5 00000028

   6 00000000

   7 7AD49B78 => 7AEDB914

   8 7AD49B74 => 00000000

   9 7AD49B70 => 00000000


  LIB$GET_SYMBOL at 15:33:46.65

  R0:00000000 R1:00000001 R2:00A27A18 R3:00A13990 R4 :00000004 R5:
00000001

  R6:00000000 R7:00AFC008 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_SYMBOL called with 4 args

     1 7AD48F58 => 010E0004

                   00A13990 => PATH

     2 7AD48F28 => 010E0401

                   7AD48F70 =>
ÐeM     8       T•G     ÐeM     8’Ôz     ¸ L
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      3 7AD48F60 => 00000000
      4 00000000
   LIB$GET_SYMBOL returning 4 args
      1 7AD48F58 => 010E0004
                    00A13990 => PATH
      2 7AD48F28 => 010E0401
                    7AD48F70 => sys$login
      3 7AD48F60 => 00000009
      4 00000000
   LIB$GET_SYMBOL returned: 00000001 at 15:33:46.66
   R0:00000001 R1:00000009 R2:00A27A18 R3:00A13990 R4 :00000004 R5:
00000001
   R6:00000000 R7:00AFC008 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


   LIB$INSERT_TREE at 15:33:46.66
   R0:00000000 R1:00000001 R2:00A3F2D8 R3:00AFC008 R4 :00000001 R5:
00AA0280
   R6:7AD48F70 R7:00AA0280 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
   LIB$INSERT_TREE called with 7 args
      1 00A67B40 => 00000000
      2 7AD48EC8 => 00A13990
      3 7AD48ED8 => 00000000
      4 00A3F170 => 0116300A
      5 00A3F288 => 00083089
      6 7AD48EE0 => 00A0FF10
      7 00000000


     LIB$VM_MALLOC at 15:33:46.66
     R0:0045C834 R1:7AD48BC8 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00000000
     R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000
     LIB$VM_MALLOC called with 1 arg
        1 0000001C
     LIB$VM_MALLOC returning 1 arg
        1 0000001C
     LIB$VM_MALLOC returned: 00B2C008 at 15:33:46.66
     R0:00B2C008 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00000000
     R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    LIB$VM_MALLOC at 15:33:46.66

    R0:00B2C008 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008

    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000

    LIB$VM_MALLOC called with 1 arg

       1 00000005

    LIB$VM_MALLOC returning 1 arg

       1 00000005

    LIB$VM_MALLOC returned: 00B2C030 at 15:33:46.66

    R0:00B2C030 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008

    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000


    LIB$VM_MALLOC at 15:33:46.66

    R0:00B2C030 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008

    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000

    LIB$VM_MALLOC called with 1 arg

       1 0000000A

    LIB$VM_MALLOC returning 1 arg

       1 0000000A

    LIB$VM_MALLOC returned: 00B2C040 at 15:33:46.66

    R0:00B2C040 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008

    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000


  LIB$INSERT_TREE returning 7 args

     1 00A67B40 => 00B2C008

     2 7AD48EC8 => 00A13990

     3 7AD48ED8 => 00000000

     4 00A3F170 => 0116300A

     5 00A3F288 => 00083089

     6 7AD48EE0 => 00B2C008

     7 00000000

  LIB$INSERT_TREE returned: 00158001 at 15:33:46.66

  R0:00158001 R1:00158214 R2:00A3F2D8 R3:00AFC008 R4 :00000001 R5:
00AA0280

  R6:7AD48F70 R7:00AA0280 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  LIB$GET_SYMBOL at 15:33:46.66

  R0:00000000 R1:00000001 R2:00A27A18 R3:00A139F0 R4 :00000009 R5:
00000001

  R6:00000000 R7:00AFC008 R8:00A9E2D7 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_SYMBOL called with 4 args

     1 7AD48F58 => 010E0009

                   00A139F0 => VAXC$PATH

     2 7AD48F28 => 010E0401

                   7AD48F70 => sys$login

     3 7AD48F60 => 00000000

     4 00000000

  LIB$GET_SYMBOL returning 4 args

     1 7AD48F58 => 010E0009

                   00A139F0 => VAXC$PATH

     2 7AD48F28 => 010E0401

                   7AD48F70 => sys$login

     3 7AD48F60 => 00000000

     4 00000000

  LIB$GET_SYMBOL returned: 00158364 at 15:33:46.66

  R0:00158364 R1:FFFFFFFF R2:00A27A18 R3:00A139F0 R4 :00000009 R5:
00000001

  R6:00000000 R7:00AFC008 R8:00A9E2D7 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


  LIB$INSERT_TREE at 15:33:46.66

  R0:00000000 R1:00000001 R2:00A3F2D8 R3:00AFC008 R4 :00000001 R5:
0000000E

  R6:7AD49030 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$INSERT_TREE called with 7 args

     1 00A67B40 => 00B2C008

     2 7AD48F88 => 00A13CF0

     3 7AD48F98 => 00000000

     4 00A3F170 => 0116300A

     5 00A3F288 => 00083089

     6 7AD48FA0 => 20202020

                 =/        ti/

     7 00000000


    LIB$VM_MALLOC at 15:33:46.66
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    R0:00000001 R1:0045C520 R2:00A1A260 R3:7AD48F88 R4 :7AD48BD0 R5:
00000000
    R6:00158001 R7:7AD48FA0 R8:00A3F288 R9:00A3F170 R10:7AD48F88
R11:00000000
    LIB$VM_MALLOC called with 1 arg
       1 0000001C
    LIB$VM_MALLOC returning 1 arg
       1 0000001C
    LIB$VM_MALLOC returned: 00B2C058 at 15:33:46.66
    R0:00B2C058 R1:00000001 R2:00A1A260 R3:7AD48F88 R4 :7AD48BD0 R5:
00000000
    R6:00158001 R7:7AD48FA0 R8:00A3F288 R9:00A3F170 R10:7AD48F88
R11:00000000

    LIB$VM_MALLOC at 15:33:46.66
    R0:00B2C058 R1:00000001 R2:00A1A260 R3:7AD48F88 R4 :7AD48BD0 R5:
00B2C058
    R6:00158001 R7:7AD48FA0 R8:00A3F288 R9:00A3F170 R10:7AD48F88
R11:00000000
    LIB$VM_MALLOC called with 1 arg
       1 0000000A
    LIB$VM_MALLOC returning 1 arg
       1 0000000A
    LIB$VM_MALLOC returned: 00B2C080 at 15:33:46.66
    R0:00B2C080 R1:00000001 R2:00A1A260 R3:7AD48F88 R4 :7AD48BD0 R5:
00B2C058
    R6:00158001 R7:7AD48FA0 R8:00A3F288 R9:00A3F170 R10:7AD48F88
R11:00000000

    LIB$VM_MALLOC at 15:33:46.66
    R0:00B2C080 R1:00000001 R2:00A1A260 R3:7AD48F88 R4 :7AD48BD0 R5:
00B2C058
    R6:00158001 R7:7AD48FA0 R8:00A3F288 R9:00A3F170 R10:7AD48F88
R11:00000000
    LIB$VM_MALLOC called with 1 arg
       1 00000020
    LIB$VM_MALLOC returning 1 arg
       1 00000020
    LIB$VM_MALLOC returned: 00B2C098 at 15:33:46.66
    R0:00B2C098 R1:00000001 R2:00A1A260 R3:7AD48F88 R4 :7AD48BD0 R5:
00B2C058
    R6:00158001 R7:7AD48FA0 R8:00A3F288 R9:00A3F170 R10:7AD48F88
R11:00000000

  LIB$INSERT_TREE returning 7 args
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    1 00A67B40 => 00B2C008
    2 7AD48F88 => 00A13CF0
    3 7AD48F98 => 00000000
    4 00A3F170 => 0116300A
    5 00A3F288 => 00083089
    6 7AD48FA0 => 00B2C058
                  =/X/
    7 00000000
  LIB$INSERT_TREE returned: 00158001 at 15:33:46.66
  R0:00158001 R1:00000000 R2:00A3F2D8 R3:00AFC008 R4 :00000001 R5:
0000000E
  R6:7AD49030 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


  LIB$VM_CALLOC at 15:33:46.66
  R0:00A65780 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65780
  R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$VM_CALLOC called with 2 args
    1 00000001
    2 00000204
  LIB$VM_CALLOC returning 2 args
    1 00000001
    2 00000204
  LIB$VM_CALLOC returned: 00B2C0C0 at 15:33:46.66
  R0:00B2C0C0 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65780
  R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


  LIB$VM_CALLOC at 15:33:46.66
  R0:00B2C0C0 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65778
  R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$VM_CALLOC called with 2 args
    1 00000001
    2 00000421
  LIB$VM_CALLOC returning 2 args
    1 00000001
    2 00000421
  LIB$VM_CALLOC returned: 00B2C300 at 15:33:46.66
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R0:00B2C300 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65778

  R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


  LIB$VM_CALLOC at 15:33:46.66

  R0:00B2C300 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65778

  R6:00B2C300 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$VM_CALLOC called with 2 args

     1 00000001

     2 00000421

  LIB$VM_CALLOC returning 2 args

     1 00000001

     2 00000421

  LIB$VM_CALLOC returned: 00B2C780 at 15:33:46.66

  R0:00B2C780 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65778

  R6:00B2C300 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_FOREIGN at 15:33:46.66

  R0:00000000 R1:00000000 R2:00A13DB8 R3:7FFCF934 R4 :00000000 R5:
00A65778

  R6:00B2C300 R7:00B2C780 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_FOREIGN called with 3 args

     1 7AD497D8 => 01000421

                 =/!/

     2 00000000

     3 7AD497E0 => 00000000

  LIB$GET_FOREIGN returning 3 args

     1 7AD497D8 => 01000421

                 =/!/

     2 00000000

     3 7AD497E0 => 00000000

  LIB$GET_FOREIGN returned: 00000001 at 15:33:46.66

  R0:00000001 R1:00000000 R2:00A13DB8 R3:7FFCF934 R4 :00000000 R5:
00A65778

  R6:00B2C300 R7:00B2C780 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


  LIB$VM_FREE at 15:33:46.66
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R0:FFFFFFFF R1:00000000 R2:00A1A130 R3:00B2C780 R4 :00000000 R5:
00A65778
  R6:00B2C300 R7:00B2C780 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$VM_FREE called with 1 arg
     1 00B2C780 => 20202020
                 =/                                              ...
  LIB$VM_FREE returning 1 arg
     1 00B2C780 => 00000000
  LIB$VM_FREE returned: 00000001 at 15:33:46.66
  R0:00000001 R1:77770000 R2:00A1A130 R3:00B2C780 R4 :00000000 R5:
00A65778
  R6:00B2C300 R7:00B2C780 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


DECC$MAIN returning 9 args
   1 7FFCF884 => 00010830
              =/0/
   2 7AE63EC8 => 00303089
   3 7FFCF814 => 00000003
   4 7FFCF934 => 001C0048
              =/H/
   5 00000028
   6 00000000
   7 7AD49B78 => 00000001
   8 7AD49B74 => 00B2C0C0
   9 7AD49B70 => 00A9E2B0
DECC$MAIN returned: 00A9E2B0 at 15:33:46.66
R0:00A9E2B0 R1:00000001 R2:00010830 R3:7AF08EB2 R4 :7FFCF814 R5:
7FFCF934
R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


LIB$GET_FOREIGN at 15:33:46.66
R0:00000001 R1:00B3C000 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
LIB$GET_FOREIGN called with 4 args
   1 7AD49A40 => 010E00FF
                 00070248 =>

   2 00000000
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    3 7AD49A78 => 00000000
    4 00000000
LIB$GET_FOREIGN returning 4 args
    1 7AD49A40 => 010E00FF
                  00070248 =>
    2 00000000
    3 7AD49A78 => 00000000
    4 00000000
LIB$GET_FOREIGN returned: 00000001 at 15:33:46.66
R0:00000001 R1:00000000 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8


SMG$CREATE_PASTEBOARD at 15:33:46.66
R0:00000001 R1:00000000 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:7FFA4F28
R11:7FFCDBE8
SMG$CREATE_PASTEBOARD called with 7 args
    1 00124B40 => 00000000
    2 00000000
    3 00124B60 => 00000000
    4 00090020 => 00000050
                =/P/
    5 00020000 => 00000001
    6 7AD49A50 => 00606000
    7 00000000


  LIB$CREATE_VM_ZONE at 15:33:46.66
  R0:00000000 R1:010E0009 R2:00B9AA00 R3:00000007 R4 :00124B60 R5:
00090020
  R6:00000000 R7:00BC8024 R8:00000000 R9:00090020 R10:7FFA4F28
R11:7FFCDBE8
  LIB$CREATE_VM_ZONE called with 11 args
      1 00BC8020 => 00000000
      2 7AD496B0 => 00000001
      3 00000000
      4 7AD496B8 => 000000A1
      5 00000000
      6 00000000
      7 00000000
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      8 00000000
      9 00000000
     10 00000000
     11 7AD49630 => 010E0009
                    00BA8750 => SMG$_ZONE
  LIB$CREATE_VM_ZONE returning 11 args
      1 00BC8020 => 007D0800
      2 7AD496B0 => 00000001
      3 00000000
      4 7AD496B8 => 000000A1
      5 00000000
      6 00000000
      7 00000000
      8 00000000
      9 00000000
     10 00000000
     11 7AD49630 => 010E0009
                    00BA8750 => SMG$_ZONE
  LIB$CREATE_VM_ZONE returned: 00000001 at 15:33:46.66
  R0:00000001 R1:0000C3A5 R2:00B9AA00 R3:00000007 R4 :00124B60 R5:
00090020
  R6:00000000 R7:00BC8024 R8:00000000 R9:00090020 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_EF at 15:33:46.66
  R0:00010001 R1:00008000 R2:00B9BA80 R3:7AD49628 R4 :00BA8860 R5:
7AD49670
  R6:00010001 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_EF called with 1 arg
      1 00BC80D0 => 00000000
  LIB$GET_EF returning 1 arg
      1 00BC80D0 => 0000003F
                 =/?/
  LIB$GET_EF returned: 00000001 at 15:33:46.66
  R0:00000001 R1:00008000 R2:00B9BA80 R3:7AD49628 R4 :00BA8860 R5:
7AD49670
  R6:00010001 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_VM at 15:33:46.66
  R0:00128069 R1:0000014C R2:00B9A500 R3:00BC8020 R4 :00BA8860 R5:
7AD49670
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_VM called with 3 args
     1 7AD492B8 => 0000014C
                 =/L/
     2 7AD49298 => 00000009
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD492B8 => 0000014C
                 =/L/
     2 7AD49298 => 007D2808
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.66
  R0:00000001 R1:09110000 R2:00B9A500 R3:00BC8020 R4 :00BA8860 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_VM at 15:33:46.66
  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_VM called with 3 args
     1 7AD49250 => 00000038
                 =/8/
     2 7AD49238 => 7AD49260
                 =/`/
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD49250 => 00000038
                 =/8/
     2 7AD49238 => 007D2960
                 =/`)}/
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.66
  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_VM at 15:33:46.66
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
7AD49670

  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM called with 3 args

     1 7AD49250 => 00004380

     2 007D2968 => 00000000

     3 00BC8020 => 007D0800

  LIB$GET_VM returning 3 args

     1 7AD49250 => 00004380

     2 007D2968 => 007D29A8

     3 00BC8020 => 007D0800

  LIB$GET_VM returned: 00000001 at 15:33:46.66

  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
7AD49670

  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_VM at 15:33:46.66

  R0:007D2960 R1:0000004A R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
0000FFFF

  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM called with 3 args

     1 7AD49250 => 0000004A

                 =/J/

     2 007D298C => 00000000

     3 00BC8020 => 007D0800

  LIB$GET_VM returning 3 args

     1 7AD49250 => 0000004A

                 =/J/

     2 007D298C => 007D6D38

                 =/8m}/

     3 00BC8020 => 007D0800

  LIB$GET_VM returned: 00000001 at 15:33:46.67

  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
0000FFFF

  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_VM at 15:33:46.68

  R0:00000001 R1:007D2808 R2:00B9A500 R3:00BC8020 R4 :00BA8860 R5:
7AD49670
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM called with 3 args

     1 7AD492A0 => 00000200

     2 007D2878 => 00000000

     3 00BC8020 => 007D0800

  LIB$GET_VM returning 3 args

     1 7AD492A0 => 00000200

     2 007D2878 => 007D6D90

     3 00BC8020 => 007D0800

  LIB$GET_VM returned: 00000001 at 15:33:46.68

  R0:00000001 R1:09110000 R2:00B9A500 R3:00BC8020 R4 :00BA8860 R5:
7AD49670

  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_VM at 15:33:46.68

  R0:007D2908 R1:007D290C R2:00B9BA80 R3:007D2808 R4 :00BA8860 R5:
7AD49670

  R6:000000FF R7:007D2908 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM called with 3 args

     1 007D2908 => 000000FF

     2 007D290C => 00000000

     3 00BC8020 => 007D0800

  LIB$GET_VM returning 3 args

     1 007D2908 => 000000FF

     2 007D290C => 007D6FA0

     3 00BC8020 => 007D0800

  LIB$GET_VM returned: 00000001 at 15:33:46.68

  R0:00000001 R1:09110000 R2:00B9BA80 R3:007D2808 R4 :00BA8860 R5:
7AD49670

  R6:000000FF R7:007D2908 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8


  LIB$GET_VM at 15:33:46.68

  R0:00000000 R1:00000000 R2:00B9AA00 R3:00000007 R4 :00124B60 R5:
00090020

  R6:0000000A R7:00BC8024 R8:00000000 R9:00BA8750 R10:00000000
R11:7FFCDBE8

  LIB$GET_VM called with 3 args

     1 7AD496B0 => 0000000A

     2 007D28C0 => 00000000
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      3 00BC8020 => 007D0800

   LIB$GET_VM returning 3 args

      1 7AD496B0 => 0000000A

      2 007D28C0 => 007D70B0

      3 00BC8020 => 007D0800

   LIB$GET_VM returned: 00000001 at 15:33:46.68

   R0:00000001 R1:09110000 R2:00B9AA00 R3:00000007 R4 :00124B60 R5:
 00090020

   R6:0000000A R7:00BC8024 R8:00000000 R9:00BA8750 R10:00000000
 R11:7FFCDBE8


 SMG$CREATE_PASTEBOARD returning 7 args

    1 00124B40 => 00000000

    2 00000000

    3 00124B60 => 00000024

                =/$/

    4 00090020 => 00000050

                =/P/

    5 00020000 => 00000001

    6 7AD49A50 => 00000006

    7 00000000

 SMG$CREATE_PASTEBOARD returned: 00000001 at 15:33:46.68

 R0:00000001 R1:00BC8024 R2:000106C0 R3:00070248 R4 :00070004 R5:
 001262C0

 R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:7FFA4F28
 R11:7FFCDBE8


 DECC$GETENV at 15:33:46.68

 R0:00000001 R1:00BC8024 R2:000106C0 R3:00070248 R4 :00070004 R5:
 001262C0

 R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:7FFA4F28
 R11:7FFCDBE8

 DECC$GETENV called with 1 arg

    1 00010790 => 24554644

                =/DFU$NOSMG/


   LIB$GET_SYMBOL at 15:33:46.68

   R0:00000000 R1:00000001 R2:00A27A18 R3:00010790 R4 :00000009 R5:
 00000001

   R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:7FFA4F28
 R11:7FFCDBE8

   LIB$GET_SYMBOL called with 4 args

      1 7AD492F8 => 010E0009
```

© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
                    00010790 => DFU$NOSMG
        2 7AD492C8 => 010E0401
                    7AD49310 =>
           D       q              °€  ÿÿÿÿºÐ‹
        3 7AD49300 => 00000000
        4 00000000
     LIB$GET_SYMBOL returning 4 args
        1 7AD492F8 => 010E0009
                    00010790 => DFU$NOSMG
        2 7AD492C8 => 010E0401
                    7AD49310 =>
           D       q              °€  ÿÿÿÿºÐ‹
        3 7AD49300 => 00000000
        4 00000000
     LIB$GET_SYMBOL returned: 00158364 at 15:33:46.68
     R0:00158364 R1:FFFFFFFF R2:00A27A18 R3:00010790 R4 :00000009 R5:
   00000001
     R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:7FFA4F28
   R11:7FFCDBE8


   DECC$GETENV returning 1 arg
      1 00010790 => 24554644
                  =/DFU$NOSMG/
   DECC$GETENV returned: 00000000 at 15:33:46.68
   R0:00000000 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5:
   001262C0
   R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:7FFA4F28
   R11:7FFCDBE8


   SMG$CREATE_VIRTUAL_KEYBOARD at 15:33:46.68
   R0:00000006 R1:00000000 R2:000106C0 R3:00070248 R4 :00070004 R5:
   001262C0
   R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
   R11:00090080
   SMG$CREATE_VIRTUAL_KEYBOARD called with 5 args
      1 00124270 => 00000000
      2 00000000
      3 00000000
      4 00000000
      5 00000000


     LIB$GET_VM at 15:33:46.68
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R0:00000001 R1:00008000 R2:00B98B18 R3:00000005 R4 :0000000C R5:
7AD495D0
  R6:00000014 R7:00BC8020 R8:00BC8018 R9:00090020 R10:00124270
R11:00090080
  LIB$GET_VM called with 3 args
     1 7AD493B8 => 00000014
     2 7AD492B8 => 020E0000
                   00000000 =>
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD493B8 => 00000014
     2 7AD492B8 => 007D70C8
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B98B18 R3:00000005 R4 :0000000C R5:
7AD495D0
  R6:00000014 R7:00BC8020 R8:00BC8018 R9:00090020 R10:00124270
R11:00090080

  LIB$GET_VM at 15:33:46.68
  R0:00000001 R1:00B1E388 R2:00B98B18 R3:00000005 R4 :0000000C R5:
7AD495D0
  R6:00000014 R7:00BC8020 R8:00BC8018 R9:00090020 R10:00124270
R11:00090080
  LIB$GET_VM called with 3 args
     1 7AD493B8 => 000001C0
     2 7AD49360 => 0000000F
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD493B8 => 000001C0
     2 7AD49360 => 007D70E8
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B98B18 R3:00000005 R4 :0000000C R5:
7AD495D0
  R6:00000014 R7:00BC8020 R8:00BC8018 R9:00090020 R10:00124270
R11:00090080

  LIB$GET_VM at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B98B18 R3:00000005 R4 :0000000C R5:
7AD495D0
  R6:00000014 R7:00BC8020 R8:00BC8018 R9:007D7120 R10:00124270
R11:00090080
  LIB$GET_VM called with 3 args
```

© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      1 7AD493B8 => 0000000C
      2 7AD49358 => 7AD49448
                    =/H/
      3 00BC8020 => 007D0800
   LIB$GET_VM returning 3 args
      1 7AD493B8 => 0000000C
      2 7AD49358 => 007D72B8
      3 00BC8020 => 007D0800
   LIB$GET_VM returned: 00000001 at 15:33:46.68
   R0:00000001 R1:09110000 R2:00B98B18 R3:00000005 R4 :0000000C R5:
 7AD495D0
   R6:00000014 R7:00BC8020 R8:00BC8018 R9:007D7120 R10:00124270
 R11:00090080


   LIB$GET_EF at 15:33:46.68
   R0:00000001 R1:09110000 R2:00B98B18 R3:00000005 R4 :0000000C R5:
 7AD495D0
   R6:00000014 R7:00BC8020 R8:00BC8018 R9:007D7120 R10:00124270
 R11:00090080
   LIB$GET_EF called with 1 arg
      1 00BC8000 => 00000000
   LIB$GET_EF returning 1 arg
      1 00BC8000 => 0000003E
                    =/>/
   LIB$GET_EF returned: 00000001 at 15:33:46.68
   R0:00000001 R1:09110000 R2:00B98B18 R3:00000005 R4 :0000000C R5:
 7AD495D0
   R6:00000014 R7:00BC8020 R8:00BC8018 R9:007D7120 R10:00124270
 R11:00090080


   LIB$GET_VM at 15:33:46.68
   R0:00128069 R1:00128069 R2:00B992F0 R3:00000000 R4 :00000050 R5:
 000001A0
   R6:00BA855C R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
 R11:00090080
   LIB$GET_VM called with 3 args
      1 7AD49210 => 000001E8
      2 7AD491E0 => 818D55A0
      3 00BC8020 => 007D0800
   LIB$GET_VM returning 3 args
      1 7AD49210 => 000001E8
      2 7AD491E0 => 007D72D0
      3 00BC8020 => 007D0800
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  LIB$GET_VM returned: 00000001 at 15:33:46.68

  R0:00000001 R1:09110000 R2:00B992F0 R3:00000000 R4 :00000050 R5:
000001A0

  R6:00BA855C R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080


  LIB$GET_VM at 15:33:46.68

  R0:00000001 R1:007D72D0 R2:00B992F0 R3:00000000 R4 :00000000 R5:
00000050

  R6:00BA8330 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080

  LIB$GET_VM called with 3 args

     1 7AD49208 => 000007A0

     2 7AD491E8 => 00000000

     3 00BC8020 => 007D0800

  LIB$GET_VM returning 3 args

     1 7AD49208 => 000007A0

     2 7AD491E8 => 007D74C8

     3 00BC8020 => 007D0800

  LIB$GET_VM returned: 00000001 at 15:33:46.68

  R0:00000001 R1:09110000 R2:00B992F0 R3:00000000 R4 :00000000 R5:
00000050

  R6:00BA8330 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080


  LIB$GET_VM at 15:33:46.68

  R0:00000001 R1:00000000 R2:00B98B18 R3:00000000 R4 :00000050 R5:
000000A0

  R6:00000014 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080

  LIB$GET_VM called with 3 args

     1 7AD493B8 => 000000A0

     2 007D729C => 00000000

     3 00BC8020 => 007D0800

  LIB$GET_VM returning 3 args

     1 7AD493B8 => 000000A0

     2 007D729C => 007D7C78

                   =/x|}/

     3 00BC8020 => 007D0800

  LIB$GET_VM returned: 00000001 at 15:33:46.68

  R0:00000001 R1:09110000 R2:00B98B18 R3:00000000 R4 :00000050 R5:
000000A0

  R6:00000014 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
SMG$CREATE_VIRTUAL_KEYBOARD returning 5 args
   1 00124270 => 007D7120
               =/ q}/
   2 00000000
   3 00000000
   4 00000000
   5 00000000
SMG$CREATE_VIRTUAL_KEYBOARD returned: 00000001 at 15:33:46.68
R0:00000001 R1:00008000 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080

SMG$ERASE_PASTEBOARD at 15:33:46.68
R0:00000001 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080
SMG$ERASE_PASTEBOARD called with 1 arg
   1 00124B40 => 00000000
SMG$ERASE_PASTEBOARD returning 1 arg
   1 00124B40 => 00000000
SMG$ERASE_PASTEBOARD returned: 00000001 at 15:33:46.68
R0:00000001 R1:FFFFF9F8 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080

SMG$CREATE_KEY_TABLE at 15:33:46.68
R0:00000003 R1:FFFFF9F8 R2:00010530 R3:00070000 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080
SMG$CREATE_KEY_TABLE called with 1 arg
   1 00070000 => 00000000

  LIB$GET_VM at 15:33:46.68
  R0:00BC8020 R1:007D0800 R2:00B99068 R3:00070000 R4 :00070004 R5:
001262C0
  R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080
  LIB$GET_VM called with 3 args
     1 7AD496B0 => 0000005E
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
                 =/^/
      2 00070000 => 00000000
      3 00BC8020 => 007D0800
   LIB$GET_VM returning 3 args
      1 7AD496B0 => 0000005E
                 =/^/
      2 00070000 => 007D7D28
                 =/(}}/
      3 00BC8020 => 007D0800
   LIB$GET_VM returned: 00000001 at 15:33:46.68
   R0:00000001 R1:09110000 R2:00B99068 R3:00070000 R4 :00070004 R5:
 001262C0
   R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
 R11:00090080


 SMG$CREATE_KEY_TABLE returning 1 arg
    1 00070000 => 007D7D28
               =/(}}/
 SMG$CREATE_KEY_TABLE returned: 00000001 at 15:33:46.68
 R0:00000001 R1:007D7D28 R2:00010530 R3:00070000 R4 :00070004 R5:
 001262C0
 R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
 R11:00090080


 SMG$ADD_KEY_DEF at 15:33:46.68
 R0:00000001 R1:007D7D28 R2:00010530 R3:00070000 R4 :00070004 R5:
 001262C0
 R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
 R11:00090080
 SMG$ADD_KEY_DEF called with 6 args
    1 00070000 => 007D7D28
               =/(}}/
    2 7AD49998 => 010E0002
                 00010570 => DO
    3 00000000
    4 7AD499A0 => 00000003
    5 7AD49998 => 010E0002
                 00010570 => DO
    6 00000000


   LIB$INSERT_TREE at 15:33:46.68
   R0:00000001 R1:007D7D6C R2:00B98D28 R3:00000006 R4 :007D7D28 R5:
 001262C0
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080
  LIB$INSERT_TREE called with 7 args
     1 007D7D28 => 00000000
     2 007D7D58 => 010E000A
                   007D7D64 => DEFAULT  (
     3 7AD49668 => 00000000
     4 00B98F60 => 00183089
     5 00B99028 => 00283089
     6 7AD49658 => 00000000
     7 00000000

    LIB$GET_VM at 15:33:46.68
    R0:0045C834 R1:00000054 R2:004C6FB0 R3:004C6F78 R4 :00000000 R5:
00000000
    R6:00158001 R7:7AD49658 R8:00B99028 R9:00B98F60 R10:007D7D58
R11:00000000
    LIB$GET_VM called with 3 args
       1 7AD49250 => 00000054
                     =/T/
       2 7AD49270 => 00000000
       3 00BC8020 => 007D0800
    LIB$GET_VM returning 3 args
       1 7AD49250 => 00000054
                     =/T/
       2 7AD49270 => 007D7D98
       3 00BC8020 => 007D0800
    LIB$GET_VM returned: 00000001 at 15:33:46.68
    R0:00000001 R1:09110000 R2:004C6FB0 R3:004C6F78 R4 :00000000 R5:
00000000
    R6:00158001 R7:7AD49658 R8:00B99028 R9:00B98F60 R10:007D7D58
R11:00000000

  LIB$INSERT_TREE returning 7 args
     1 007D7D28 => 007D7D98
     2 007D7D58 => 010E000A
                   007D7D64 => DEFAULT  (
     3 7AD49668 => 00000000
     4 00B98F60 => 00183089
     5 00B99028 => 00283089
     6 7AD49658 => 007D7D98
     7 00000000
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  LIB$INSERT_TREE returned: 00158001 at 15:33:46.68
  R0:00158001 R1:00158214 R2:00B98D28 R3:00000006 R4 :007D7D28 R5:
001262C0
  R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080


  LIB$SCOPY_R_DX at 15:33:46.68
  R0:00128412 R1:007D7D98 R2:00B98D28 R3:00000006 R4 :007D7D28 R5:
007D7D98
  R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080
  LIB$SCOPY_R_DX called with 3 args
     1 7AD49670 => 0000000A
     2 007D7D64 => 41464544
                 =/DEFAULT/
     3 007D7DA4 => 020E0000
                   00000000 =>
  LIB$SCOPY_R_DX returning 3 args
     1 7AD49670 => 0000000A
     2 007D7D64 => 41464544
                 =/DEFAULT/
     3 007D7DA4 => 020E000A
                   007CED2C => DEFAULT  (
  LIB$SCOPY_R_DX returned: 00000001 at 15:33:46.68
  R0:00000001 R1:FFFFFFFF R2:00B98D28 R3:00000006 R4 :007D7D28 R5:
007D7D98
  R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080


…
---Many lines omitted---
…
DECC$GXVSPRINTF at 15:33:46.96
R0:00000031 R1:7AD49998 R2:00010EC0 R3:00124930 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001
DECC$GXVSPRINTF called with 3 args
   1 00124930 => 20202020
               =/      UNDELETE    : Recover deleted files/
   2 00013D78 => 20202020
               =/      VERIFY      : Check and repair disk structur...
   3 7AD499B0 => 00000004
```

```
DECC$GXVSPRINTF returning 3 args
   1 00124930 => 20202020
               =/     VERIFY     : Check and repair disk structur...
   2 00013D78 => 20202020
               =/     VERIFY     : Check and repair disk structur...
   3 7AD499B0 => 00000004
DECC$GXVSPRINTF returned: 00000031 at 15:33:46.96
R0:00000031 R1:00000001 R2:00010EC0 R3:00124930 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001


DECC$STRLEN at 15:33:46.96
R0:00000001 R1:00000001 R2:00010EC0 R3:00124930 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001
DECC$STRLEN called with 1 arg
   1 00124930 => 20202020
               =/     VERIFY     : Check and repair disk structur...
DECC$STRLEN returning 1 arg
   1 00124930 => 20202020
               =/     VERIFY     : Check and repair disk structur...
DECC$STRLEN returned: 00000031 at 15:33:46.96
R0:00000031 R1:00000001 R2:00010EC0 R3:00124930 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001


SMG$PUT_LINE at 15:33:46.96
R0:00000001 R1:010E01FF R2:00010EC0 R3:00124930 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001
SMG$PUT_LINE called with 8 args
   1 00090080 => 007D80F8
   2 7AD49968 => 010E0031
                 00124930 =>      VERIFY     : Check and repair disk
structur
   3 00000000
   4 00000000
   5 00000000
   6 000201C8 => 00000001
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    7 00000000
    8 00000000


  LIB$SCANC at 15:33:46.96
  R0:00000000 R1:010E0031 R2:00B99660 R3:00000007 R4 :007D80F8 R5:
00000000
  R6:00000031 R7:00124930 R8:00000000 R9:00000000 R10:00000000
R11:00000000
  LIB$SCANC called with 3 args
      1 7AD49520 => 010E0031
                    00124930 =>      VERIFY     : Check and repair disk
structur
      2 00BA8594 => 01010101
      3 00BA8590 => 000000FF
  LIB$SCANC returning 3 args
      1 7AD49520 => 010E0031
                    00124930 =>      VERIFY     : Check and repair disk
structur
      2 00BA8594 => 01010101
      3 00BA8590 => 000000FF
  LIB$SCANC returned: 00000000 at 15:33:46.96
  R0:00000000 R1:00124961 R2:00B99660 R3:00000007 R4 :007D80F8 R5:
00000000
  R6:00000031 R7:00124930 R8:00000000 R9:00000000 R10:00000000
R11:00000000


SMG$PUT_LINE returning 8 args
    1 00090080 => 007D80F8
    2 7AD49968 => 010E0031
                  00124930 =>      VERIFY     : Check and repair disk
structur
    3 00000000
    4 00000000
    5 00000000
    6 000201C8 => 00000001
    7 00000000
    8 00000000
SMG$PUT_LINE returned: 00000001 at 15:33:46.96
R0:00000001 R1:00000000 R2:00010EC0 R3:00124930 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
SMG$END_PASTEBOARD_UPDATE at 15:33:46.96

R0:00000000 R1:000900F0 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0

R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001

SMG$END_PASTEBOARD_UPDATE called with 1 arg

   1 00124B40 => 00000000

SMG$END_PASTEBOARD_UPDATE returning 1 arg

   1 00124B40 => 00000000

SMG$END_PASTEBOARD_UPDATE returned: 00000001 at 15:33:46.97

R0:00000001 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0

R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00000001


DECC$GETENV at 15:33:46.97

R0:00000001 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0

R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380

DECC$GETENV called with 1 arg

   1 000105B0 => 24554644

               =/DFU$TRACE/


  LIB$GET_SYMBOL at 15:33:46.97

  R0:00000000 R1:00000001 R2:00A27A18 R3:000105B0 R4 :00000009 R5:
00000001

  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380

  LIB$GET_SYMBOL called with 4 args

     1 7AD492F8 => 010E0009

                   000105B0 => DFU$TRACE

     2 7AD492C8 => 010E0401

                   7AD49310 =>
        a       q              °€  ÿÿÿÿ°Ð<

     3 7AD49300 => 00000000

     4 00000000

  LIB$GET_SYMBOL returning 4 args

     1 7AD492F8 => 010E0009

                   000105B0 => DFU$TRACE

     2 7AD492C8 => 010E0401

                   7AD49310 =>
        a       q              °€  ÿÿÿÿ°Ð<

     3 7AD49300 => 00000000
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      4 00000000

   LIB$GET_SYMBOL returned: 00158364 at 15:33:46.97

   R0:00158364 R1:FFFFFFFF R2:00A27A18 R3:000105B0 R4 :00000009 R5:
00000001

   R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380


DECC$GETENV returning 1 arg

    1 000105B0 => 24554644

                =/DFU$TRACE/

DECC$GETENV returned: 00000000 at 15:33:46.97

R0:00000000 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0

R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


LIB$CREATE_VM_ZONE at 15:33:46.97

R0:00126400 R1:000204A0 R2:00010E00 R3:00070248 R4 :00070004 R5:
001262C0

R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380

LIB$CREATE_VM_ZONE called with 13 args

    1 000204A0 => 00000000

    2 7AD499A8 => 00000002

    3 7AD499A0 => 00000080

    4 7AD49998 => 00000028

                =/(/

    5 7AD49990 => 00000008

    6 00000000

    7 00000000

    8 00000000

    9 00000000

   10 00000000

   11 7AD49988 => 010E0008

                  00010E30 => dfu_zone

   12 00000000

   13 00000000

LIB$CREATE_VM_ZONE returning 13 args

    1 000204A0 => 007D0858

                =/X/

    2 7AD499A8 => 00000002

    3 7AD499A0 => 00000080
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    4 7AD49998 => 00000028
               =/(/
    5 7AD49990 => 00000008
    6 00000000
    7 00000000
    8 00000000
    9 00000000
   10 00000000
   11 7AD49988 => 010E0008
                  00010E30 => dfu_zone
   12 00000000
   13 00000000
LIB$CREATE_VM_ZONE returned: 00000001 at 15:33:46.97
R0:00000001 R1:0000C3A5 R2:00010E00 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


DECC$GETENV at 15:33:46.97
R0:00000000 R1:000001D5 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
DECC$GETENV called with 1 arg
   1 00013D40 => 24554644
               =/DFU$DISABLE_CHECK/


  LIB$GET_SYMBOL at 15:33:46.97
  R0:00000000 R1:00000001 R2:00A27A18 R3:00013D40 R4 :00000011 R5:
00000001
  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$GET_SYMBOL called with 4 args
     1 7AD492F8 => 010E0011
                   00013D40 => DFU$DISABLE_CHECK
     2 7AD492C8 => 010E0401
                   7AD49310 =>
        a       q              °€  ÿÿÿÿ°Ð<
     3 7AD49300 => 00000000
     4 00000000
  LIB$GET_SYMBOL returning 4 args
     1 7AD492F8 => 010E0011
                   00013D40 => DFU$DISABLE_CHECK
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      2 7AD492C8 => 010E0401
                    7AD49310 =>
         a       q               °€  ÿÿÿÿºÐ‹
      3 7AD49300 => 00000000
      4 00000000
   LIB$GET_SYMBOL returned: 00158364 at 15:33:46.97
   R0:00158364 R1:FFFFFFFF R2:00A27A18 R3:00013D40 R4 :00000011 R5:
00000001
   R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380


DECC$GETENV returning 1 arg
   1 00013D40 => 24554644
              =/DFU$DISABLE_CHECK/
DECC$GETENV returned: 00000000 at 15:33:46.97
R0:00000000 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


LIB$GET_VM at 15:33:46.97
R0:00000001 R1:001263F0 R2:00010DC0 R3:7AD49A70 R4 :00070140 R5:
00070004
R6:00000005 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
LIB$GET_VM called with 3 args
   1 7AD498E8 => 00000028
              =/(/
   2 7AD49968 => 00000000
   3 000204A0 => 007D0858
              =/X/
LIB$GET_VM returning 3 args
   1 7AD498E8 => 00000028
              =/(/
   2 7AD49968 => 007DA800
   3 000204A0 => 007D0858
              =/X/
LIB$GET_VM returned: 00000001 at 15:33:46.97
R0:00000001 R1:01140000 R2:00010DC0 R3:7AD49A70 R4 :00070140 R5:
00070004
R6:00000005 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
DECC$STRNCMP at 15:33:46.98

R0:00000001 R1:7AD40008 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA800 R8:00000001 R9:00000000 R10:FFFFFFFF
R11:00126380

DECC$STRNCMP called with 3 args

   1 00070140 => 4C4C4947

               =/GILLINGS                                             ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000008

DECC$STRNCMP returning 3 args

   1 00070140 => 4C4C4947

               =/GILLINGS                                             ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000008

DECC$STRNCMP returned: 00000001 at 15:33:46.98

R0:00000001 R1:00000000 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA800 R8:00000001 R9:00000000 R10:FFFFFFFF
R11:00126380


DECC$STRNCMP at 15:33:46.98

R0:00000001 R1:7AD4000B R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA808 R8:00000002 R9:00000000 R10:FFFFFFFF
R11:00126380

DECC$STRNCMP called with 3 args

   1 00070140 => 45544E49

               =/INTERACTIVE                                          ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 0000000B

DECC$STRNCMP returning 3 args

   1 00070140 => 45544E49

               =/INTERACTIVE                                          ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 0000000B

DECC$STRNCMP returned: 00000001 at 15:33:46.98

R0:00000001 R1:00000000 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
R6:00000005 R7:007DA808 R8:00000002 R9:00000000 R10:FFFFFFFF
R11:00126380


DECC$STRNCMP at 15:33:46.98

R0:00000001 R1:7AD40005 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA810 R8:00000003 R9:00000000 R10:FFFFFFFF
R11:00126380

DECC$STRNCMP called with 3 args

   1 00070140 => 41434F4C

               =/LOCAL                                               ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000005

DECC$STRNCMP returning 3 args

   1 00070140 => 41434F4C

               =/LOCAL                                               ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000005

DECC$STRNCMP returned: 00000008 at 15:33:46.98

R0:00000008 R1:00000007 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA810 R8:00000003 R9:00000000 R10:FFFFFFFF
R11:00126380


DECC$STRNCMP at 15:33:46.98

R0:00002224 R1:7AD40005 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA818 R8:00000004 R9:00000000 R10:FFFFFFFF
R11:00126380

DECC$STRNCMP called with 3 args

   1 00070140 => 41434F4C

               =/LOCAL                                               ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000005

DECC$STRNCMP returning 3 args

   1 00070140 => 41434F4C

               =/LOCAL                                               ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000005
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
DECC$STRNCMP returned: 00000008 at 15:33:46.98

R0:00000008 R1:00000007 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA818 R8:00000004 R9:00000000 R10:FFFFFFFF
R11:00126380


DECC$STRNCMP at 15:33:46.98

R0:00002224 R1:7AD40005 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA820 R8:00000005 R9:00000000 R10:FFFFFFFF
R11:00126380

DECC$STRNCMP called with 3 args

   1 00070140 => 41434F4C

               =/LOCAL                                          ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000005

DECC$STRNCMP returning 3 args

   1 00070140 => 41434F4C

               =/LOCAL                                          ...

   2 000104E0 => 5F554644

               =/DFU_ALLPRIV/

   3 00000005

DECC$STRNCMP returned: 00000008 at 15:33:46.98

R0:00000008 R1:00000007 R2:00010468 R3:7AD49A70 R4 :00070140 R5:
00070004

R6:00000005 R7:007DA820 R8:00000005 R9:00000000 R10:FFFFFFFF
R11:00126380


SMG$READ_COMPOSED_LINE at 15:33:46.98

R0:00000000 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0

R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380

SMG$READ_COMPOSED_LINE called with 11 args

   1 00124270 => 007D7120

               =/ q}/

   2 00070000 => 007D7D28

               =/(}}/

   3 7AD49A40 => 010E00FF

                 00070248 =>

   4 7AD49A38 => 010E0005

                 00010640 => DFU>
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
     5 7AD49A80 => 00000000
     6 00090090 => 007D8598
     7 00000000
     8 00000000
     9 00000000
    10 00000000
    11 00000000


  LIB$GET_VM at 15:33:46.98
  R0:FFFFFFFE R1:00000000 R2:00B987D0 R3:7AD49414 R4 :00000007 R5:
001262C0
  R6:00128402 R7:00000002 R8:00000001 R9:007D7D28 R10:00000001
R11:00000000
  LIB$GET_VM called with 3 args
     1 7AD49248 => 0000005A
                  =/Z/
     2 7AD49418 => 00BA8138
                  =/8/
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD49248 => 0000005A
                  =/Z/
     2 7AD49418 => 007D8BE8
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.98
  R0:00000001 R1:09110000 R2:00B987D0 R3:7AD49414 R4 :00000007 R5:
001262C0
  R6:00128402 R7:00000002 R8:00000001 R9:007D7D28 R10:00000001
R11:00000000


  LIB$LOOKUP_TREE at 15:33:48.02
  R0:007D7D6D R1:007D7D6C R2:00B98F90 R3:007D8598 R4 :0000001A R5:
00000000
  R6:007D7D58 R7:00000002 R8:00000000 R9:007D7D28 R10:007D7D6C
R11:00000001
  LIB$LOOKUP_TREE called with 4 args
     1 007D7D28 => 007D7EB8
     2 007D7D58 => 010E000A
                    007D7D64 => DEFAULT
     3 00B98F60 => 00183089
     4 7AD49298 => 00B5D8E0
  LIB$LOOKUP_TREE returning 4 args
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    1 007D7D28 => 007D7EB8
    2 007D7D58 => 010E000A
                  007D7D64 => DEFAULT
    3 00B98F60 => 00183089
    4 7AD49298 => 00B5D8E0
  LIB$LOOKUP_TREE returned: 001582FC at 15:33:48.02
  R0:001582FC R1:00B98F60 R2:00B98F90 R3:007D8598 R4 :0000001A R5:
00000000
  R6:007D7D58 R7:00000002 R8:00000000 R9:007D7D28 R10:007D7D6C
R11:00000001


  LIB$SCANC at 15:33:48.02
  R0:00000001 R1:010E0005 R2:00B99660 R3:00000005 R4 :007D8598 R5:
00000000
  R6:00000005 R7:00010640 R8:00000000 R9:00000000 R10:00128402
R11:00000000
  LIB$SCANC called with 3 args
    1 7AD49180 => 010E0005
                  00010640 => DFU>
    2 00BA8594 => 01010101
    3 00BA8590 => 000000FF
  LIB$SCANC returning 3 args
    1 7AD49180 => 010E0005
                  00010640 => DFU>
    2 00BA8594 => 01010101
    3 00BA8590 => 000000FF
  LIB$SCANC returned: 00000000 at 15:33:48.02
  R0:00000000 R1:00010645 R2:00B99660 R3:00000005 R4 :007D8598 R5:
00000000
  R6:00000005 R7:00010640 R8:00000000 R9:00000000 R10:00128402
R11:00000000


  LIB$SCANC at 15:33:48.02
  R0:00000001 R1:010E0011 R2:00B99660 R3:00000004 R4 :007D8598 R5:
00000000
  R6:00000011 R7:00BA8518 R8:00000000 R9:00000000 R10:00128402
R11:00000000
  LIB$SCANC called with 3 args
    1 7AD48F10 => 010E0011
                  00BA8518 =>  7 [7m Exit  [m 8
    2 00BA8594 => 01010101
    3 00BA8590 => 000000FF
  LIB$SCANC returning 3 args
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    1 7AD48F10 => 010E0011
                  00BA8518 =>  7 [7m Exit  [m 8
    2 00BA8594 => 01010101
    3 00BA8590 => 000000FF
  LIB$SCANC returned: 00000001 at 15:33:48.03
  R0:00000001 R1:00BA8518 R2:00B99660 R3:00000004 R4 :007D8598 R5:
00000000
  R6:00000011 R7:00BA8518 R8:00000000 R9:00000000 R10:00128402
R11:00000000


  LIB$SCANC at 15:33:48.03
  R0:00000001 R1:010E000B R2:00B99660 R3:0000000B R4 :007D8598 R5:
00000000
  R6:00000011 R7:00BA8518 R8:00000001 R9:00000000 R10:0000000B
R11:00000000
  LIB$SCANC called with 3 args
    1 7AD48F10 => 010E000B
                  00BA851E =>  Exit  [m 8
    2 00BA8594 => 01010101
    3 00BA8590 => 000000FF
  LIB$SCANC returning 3 args
    1 7AD48F10 => 010E000B
                  00BA851E =>  Exit  [m 8
    2 00BA8594 => 01010101
    3 00BA8590 => 000000FF
  LIB$SCANC returned: 00000007 at 15:33:48.03
  R0:00000007 R1:00BA8524 R2:00B99660 R3:0000000B R4 :007D8598 R5:
00000000
  R6:00000011 R7:00BA8518 R8:00000001 R9:00000000 R10:0000000B
R11:00000000


  LIB$SCOPY_R_DX at 15:33:48.03
  R0:00000001 R1:00000000 R2:00B988C8 R3:00BA8138 R4 :00000001 R5:
7AD49458
  R6:7AD49414 R7:0000001A R8:007D7120 R9:7AD49458 R10:00000000
R11:00000024
  LIB$SCOPY_R_DX called with 3 args
    1 7AD493E0 => 00000000
    2 7AD49458 => 0007001A
    3 7AD49A40 => 010E00FF
                  00070248 =>
  LIB$SCOPY_R_DX returning 3 args
    1 7AD493E0 => 00000000
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    2 7AD49458 => 0007001A
    3 7AD49A40 => 010E00FF
                  00070248 =>
  LIB$SCOPY_R_DX returned: 00000001 at 15:33:48.03
  R0:00000001 R1:010E00FF R2:00B988C8 R3:00BA8138 R4 :00000001 R5:
7AD49458
  R6:7AD49414 R7:0000001A R8:007D7120 R9:7AD49458 R10:00000000
R11:00000024


  LIB$ANALYZE_SDESC at 15:33:48.03
  R0:00000001 R1:007D7299 R2:00B988C8 R3:00BA8138 R4 :00000001 R5:
00000000
  R6:7AD49A80 R7:0000001A R8:007D7C78 R9:7AD49458 R10:00000000
R11:00000024
  LIB$ANALYZE_SDESC called with 3 args
    1 7AD49A40 => 010E00FF
                  00070248 =>
    2 7AD49308 => 00000000
    3 7AD49310 => 00000000
  LIB$ANALYZE_SDESC returning 3 args
    1 7AD49A40 => 010E00FF
                  00070248 =>
    2 7AD49308 => 000000FF
    3 7AD49310 => 00070248
                =/H/
  LIB$ANALYZE_SDESC returned: 00000001 at 15:33:48.04
  R0:00000001 R1:000000FF R2:00B988C8 R3:00BA8138 R4 :00000001 R5:
00000000
  R6:7AD49A80 R7:0000001A R8:007D7C78 R9:7AD49458 R10:00000000
R11:00000024


SMG$READ_COMPOSED_LINE returning 11 args
  1 00124270 => 007D7120
              =/ q}/
  2 00070000 => 007D7D28
              =/(}}/
  3 7AD49A40 => 010E00FF
                00070248 =>
  4 7AD49A38 => 010E0005
                00010640 => DFU>
  5 7AD49A80 => 00000000
  6 00090090 => 007D8598
```

1 1

© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
   7 00000000
   8 00000000
   9 00000000
  10 00000000
  11 00000000
SMG$READ_COMPOSED_LINE returned: 00128402 at 15:33:48.04
R0:00128402 R1:000000FF R2:000106C0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


SMG$DISABLE_BROADCAST_TRAPPING at 15:33:48.04
R0:00000001 R1:00000002 R2:000103F0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
SMG$DISABLE_BROADCAST_TRAPPING called with 1 arg
   1 00124B40 => 00000000
SMG$DISABLE_BROADCAST_TRAPPING returning 1 arg
   1 00124B40 => 00000000
SMG$DISABLE_BROADCAST_TRAPPING returned: 00000001 at 15:33:48.04
R0:00000001 R1:0F0F0001 R2:000103F0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


SMG$SET_CURSOR_ABS at 15:33:48.04
R0:00000000 R1:0000004E R2:000103F0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
SMG$SET_CURSOR_ABS called with 3 args
   1 00090090 => 007D8598
   2 000200E8 => 00000002
   3 000200F0 => 00000001
SMG$SET_CURSOR_ABS returning 3 args
   1 00090090 => 007D8598
   2 000200E8 => 00000002
   3 000200F0 => 00000001
SMG$SET_CURSOR_ABS returned: 00000001 at 15:33:48.04
R0:00000001 R1:00000002 R2:000103F0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
SMG$DELETE_PASTEBOARD at 15:33:48.04
R0:00020140 R1:00000002 R2:000103F0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
SMG$DELETE_PASTEBOARD called with 2 args
   1 00124B40 => 00000000
   2 000200F8 => 00000000


  LIB$FREE_VM at 15:33:48.04
  R0:00000001 R1:007D8B00 R2:00B9A310 R3:007D2808 R4 :00000000 R5:
00BC80A4
  R6:007D8B50 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD49688 => 00000040
                   =/@/
     2 7AD49678 => 007D8AF8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD49688 => 00000040
                   =/@/
     2 7AD49678 => 007D8AF8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A310 R3:007D2808 R4 :00000000 R5:
00BC80A4
  R6:007D8B50 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_VM at 15:33:48.04
  R0:00000001 R1:007D8B50 R2:00B9A310 R3:007D2808 R4 :00000000 R5:
00BC80A4
  R6:007D8BA0 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD49688 => 00000040
                   =/@/
     2 7AD49678 => 007D8B48
                   =/H/
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      1 7AD49688 => 00000040
                    =/@/
      2 7AD49678 => 007D8B48
                    =/H/
      3 00BC8020 => 007D0800
   LIB$FREE_VM returned: 00000001 at 15:33:48.04
   R0:00000001 R1:00000001 R2:00B9A310 R3:007D2808 R4 :00000000 R5:
 00BC80A4
   R6:007D8BA0 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
 R11:00126380


   LIB$FREE_VM at 15:33:48.04
   R0:00000000 R1:007D8BA0 R2:00B9A310 R3:007D2808 R4 :00000000 R5:
 00BC80A4
   R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
 R11:00126380
   LIB$FREE_VM called with 3 args
      1 7AD49688 => 00000040
                    =/@/
      2 7AD49678 => 007D8B98
      3 00BC8020 => 007D0800
   LIB$FREE_VM returning 3 args
      1 7AD49688 => 00000040
                    =/@/
      2 7AD49678 => 007D8B98
      3 00BC8020 => 007D0800
   LIB$FREE_VM returned: 00000001 at 15:33:48.04
   R0:00000001 R1:00000001 R2:00B9A310 R3:007D2808 R4 :00000000 R5:
 00BC80A4
   R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
 R11:00126380


   LIB$FREE_VM at 15:33:48.04
   R0:007D2960 R1:7FF5E300 R2:00B9A4C0 R3:00124B40 R4 :007D2808 R5:
 00BC80A4
   R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
 R11:00126380
   LIB$FREE_VM called with 3 args
      1 7AD49680 => 00004380
      2 007D2968 => 007D29A8
      3 00BC8020 => 007D0800
   LIB$FREE_VM returning 3 args
      1 7AD49680 => 00004380
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    2 007D2968 => 007D29A8
    3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A4C0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_VM at 15:33:48.04
  R0:00000001 R1:007D2960 R2:00B9A4C0 R3:00000001 R4 :007D2808 R5:
00BC80A4
  R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
    1 7AD49680 => 0000004A
                  =/J/
    2 007D298C => 007D6D38
                  =/8m}/
    3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
    1 7AD49680 => 0000004A
                  =/J/
    2 007D298C => 007D6D38
                  =/8m}/
    3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A4C0 R3:00000001 R4 :007D2808 R5:
00BC80A4
  R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_VM at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A4C0 R3:00000001 R4 :007D2808 R5:
00BC80A4
  R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
    1 7AD49680 => 00000038
                  =/8/
    2 7AD49678 => 007D2960
                  =/`)}/
    3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    1 7AD49680 => 00000038
                  =/8/
    2 7AD49678 => 007D2960
                  =/`)}/
    3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A4C0 R3:00000001 R4 :007D2808 R5:
00BC80A4
  R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_EF at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_EF called with 1 arg
    1 7AD496F0 => 0000003D
                  =/=/
  LIB$FREE_EF returning 1 arg
    1 7AD496F0 => 0000003D
                  =/=/
  LIB$FREE_EF returned: 00000001 at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_VM at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
    1 7AD496F0 => 00000200
    2 007D2878 => 007D6D90
    3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
    1 7AD496F0 => 00000200
    2 007D2878 => 007D6D90
    3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.04
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_VM at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD496F0 => 0000000A
     2 007D28C0 => 007D70B0
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD496F0 => 0000000A
     2 007D28C0 => 007D70B0
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_VM at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD496F0 => 000000FF
     2 007D290C => 007D6FA0
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD496F0 => 000000FF
     2 007D290C => 007D6FA0
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380


  LIB$FREE_VM at 15:33:48.06
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R0:00000001 R1:00000000 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD496F0 => 0000014C
                   =/L/
     2 7AD496E8 => 007D2808
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD496F0 => 0000014C
                   =/L/
     2 7AD496E8 => 007D2808
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380

SMG$DELETE_PASTEBOARD returning 2 args
   1 00124B40 => 00000000
   2 000200F8 => 00000000
SMG$DELETE_PASTEBOARD returned: 00000001 at 15:33:48.06
R0:00000001 R1:00BC80A4 R2:000103F0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380

DECC$EXIT at 15:33:48.06
R0:00000001 R1:00BC80A4 R2:000103F0 R3:00070248 R4 :00070004 R5:
001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
DECC$EXIT called with 1 arg
   1 00000001

  LIB$FREE_VM at 15:33:48.06
  R0:00000000 R1:007D72B8 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8
  R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD495C8 => 0000000C
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    2 7AD495B8 => 007D72B8
    3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
    1 7AD495C8 => 0000000C
    2 7AD495B8 => 007D72B8
    3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8
  R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380

  LIB$FREE_VM at 15:33:48.06
  R0:00000001 R1:007D72D0 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8
  R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
    1 7AD495C8 => 000007A0
    2 007D72D4 => 007D74C8
    3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
    1 7AD495C8 => 000007A0
    2 007D72D4 => 007D74C8
    3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8
  R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380

  LIB$FREE_VM at 15:33:48.06
  R0:00000001 R1:00000174 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8
  R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
    1 7AD495C8 => 000001E8
    2 007D7294 => 007D72D0
    3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
    1 7AD495C8 => 000001E8
    2 007D7294 => 007D72D0
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
      3 00BC8020 => 007D0800

   LIB$FREE_VM returned: 00000001 at 15:33:48.06

   R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8

   R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380


   LIB$FREE_VM at 15:33:48.06

   R0:00000000 R1:000000A0 R2:00B984D0 R3:007D7120 R4 :00000013 R5:
00000014

   R6:000000A0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380

   LIB$FREE_VM called with 3 args

      1 7AD495C8 => 000000A0

      2 007D729C => 007D7C78

                    =/x|}/

      3 00BC8020 => 007D0800

   LIB$FREE_VM returning 3 args

      1 7AD495C8 => 000000A0

      2 007D729C => 007D7C78

                    =/x|}/

      3 00BC8020 => 007D0800

   LIB$FREE_VM returned: 00000001 at 15:33:48.06

   R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :00000013 R5:
00000014

   R6:000000A0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380


   LIB$FREE_VM at 15:33:48.06

   R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D70E8 R5:
00000014

   R6:000001C0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380

   LIB$FREE_VM called with 3 args

      1 7AD495C8 => 000001C0

      2 7AD495D0 => 007D70E8

      3 00BC8020 => 007D0800

   LIB$FREE_VM returning 3 args

      1 7AD495C8 => 000001C0

      2 7AD495D0 => 007D70E8

      3 00BC8020 => 007D0800

   LIB$FREE_VM returned: 00000001 at 15:33:48.06
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
  R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D70E8 R5:
00000014
  R6:000001C0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380

  exit status: 00000001
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

Appendix F: Fake RTL Command Procedure

This appendix contains the Fake RTL command procedure.

```
$! Creates a fake RTL from an input shareable image
$!
$! Alpha and I64 version
$!
$! Author:
$!                     John Gillings
$!                     Software Systems Consultant, OpenVMS Ambassador
$!                     Hewlett-Packard Pty Limited
$!                     OpenVMS Group, Customer Support Centre
$!                     Sydney, Australia
$!
$! @FAKE_RTL [name] [start-phase] [options]
$!
$! Phases are:
$!
$!   DEF     - Define fake rtl environment
$!   COPY    - Make a copy of the original shareable image
$!   VECTOR  - Analyze the symbol vector
$!   GEN     - Generate MACRO code
$!   COMPILE - Compile MACRO code
$!   LINK    - Link fake RTL
$!   USE     - Define logical names to use image
$!
$! Options are:
$!   DEBUG   - Compile and link with /DEBUG
$!   FORCE   - Force creation of files, even if apparently unnecessary
$!
$!
$ exp=p2.NES.""
$ IF .NOT.exp THEN p2="DEF"
$ IF p1.EQS.""
$ THEN
$   endphase="DEF"
$ ELSE
$   endphase="USE"
$   img=F$SEARCH(F$PARSE(p1,"SYS$SHARE:",".EXE"))
$   IF img.EQS."" THEN EXIT
$   name=F$PARSE(img,,,"NAME")
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$   newimg="[]REAL_''name'.EXE;"
$ ENDIF
$ dbg=(p3.NES."".AND.F$LOCATE("DEBUG",p3).LT.F$LENGTH(p3)).OR.-
   F$TRNLNM("FAKE_OPTION_DEBUG")
$ force=p4.NES."".AND.F$LOCATE("FORCE",p3).LT.F$LENGTH(p3).OR.-
   F$TRNLNM("FAKE_OPTION_FORCE")
$ pid=F$GETJPI("","PID")
$ arch=F$GETSYI("ARCH_NAME")
$ IF arch.NES."Alpha".AND.arch.NES."IA64"
$ THEN
$   WRITE SYS$OUTPUT "Sorry, FAKE_RTL can't do ''arch'"
$   EXIT
$ ENDIF
$ GOTO 'p2'
$
$ DEF:
$   IF force.OR.F$TRNLNM("FAKE_DIR").EQS.""
$   THEN
$     WRITE SYS$OUTPUT "Defining FAKE_RTL environment"
$     ThisFile=F$ENVIRONMENT("PROCEDURE")
$     ThisDisk=F$PARSE(ThisFile,,,"DEVICE")
$     ThisDir=F$PARSE(ThisFile,,,"DIRECTORY")
$     DEFINE/NOLOG FAKE_DIR 'ThisDisk''ThisDir'
$     DEFINE/NOLOG FAKE_RTL FAKE_DIR:FAKE_RTL
$   ENDIF
$   IF force.OR.F$SEARCH("FAKE_RTL:.MAR").EQS."" THEN GOSUB CreateMac
$   IF force.OR.F$SEARCH("FAKE_RTL:.OPT").EQS."" THEN GOSUB CreateOpt
$   IF force.OR.F$SEARCH("FAKE_RTL:.OBJ").EQS.""
$   THEN
$     WRITE SYS$OUTPUT "Compiling FAKE_RTL.MAR"
$     opt=""
$     IF dbg THEN opt="/NOOPT/DEBUG"
$     MACRO'opt' FAKE_RTL
$   ENDIF
$   IF force.OR.F$SEARCH("FAKE_RTL:.EXE").EQS.""
$   THEN
$     DEFINE/NOLOG REAL_LIBRTL FAKE_DIR:REAL_LIBRTL
$     IF F$SEARCH("REAL_LIBRTL",".EXE").EQS."" THEN -
   COPY SYS$SHARE:LIBRTL.EXE REAL_LIBRTL
$     DEFINE/NOLOG REAL_LIBOTS FAKE_DIR:REAL_LIBOTS
$     IF F$SEARCH("REAL_LIBRTL",".EXE").EQS."" THEN -
   COPY SYS$SHARE:LIBOTS.EXE REAL_LIBOTS
$     WRITE SYS$OUTPUT "Linking FAKE_RTL.OBJ"
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$     opt=""
$     IF dbg THEN opt="/DEBUG"
$     LINK/SHARE'opt' FAKE_RTL/OPT
$   ENDIF
$   ctx=1
$   ctxs=2
$   defloop: f=f$search("FAKE_DIR:REAL_*.EXE",ctx)
$     IF f.EQS."" THEN GOTO EndDef
$     f=f$element(0,";",f)
$     n=F$PARSE(f,,,"NAME")
$     r=n-"REAL_"
$     DEFINE/NOLOG 'n' 'f'
$     fake_'r'=="DEFINE/USER/NAME=CONFINE ''r' FAKE_DIR:FAKE_''r'"
$     real_'r'=="DEASSIGN ''r'"
$   GOTO defloop
$ EndDef:
$ IF endphase.EQS."DEF" THEN EXIT
$ COPY:
$   IF exp.OR.force.OR.F$SEARCH(newimg).EQS."" THEN COPY/LOG 'img' 'newimg'
$   newimg=F$SEARCH(newimg)
$ IF endphase.EQS."COPY" THEN EXIT
$ VECTOR:
$   ON WARNING THEN GOTO Cleanup
$   ON CONTROL_Y THEN GOTO Cleanup
$   IF .NOT.(exp.OR.force.or.F$SEARCH("''name'.VEC").EQS."") THEN GOTO GEN
$   WRITE SYS$OUTPUT "Generating symbol vector for ''name' on ''arch'"
$!
$! Theory...
$!
$!   The symbol vector and the GSMATCH information "define" a shareable image
$!   completely. On a VAX, the image identification information is critical
$!   and the programmer is expected to keep it up to date (by incrementing
$!   the minor ID whenever an entry is added to the transfer vector). On
$!   Alpha, the GSMATCH is considered, but there is an additional check made
$!   to ensure the symbol vector of the image being activated is at least
$!   the same size as the expected image. It is therefore reasonably common
$!   to find the GSMATCH values for shareable images never change.
$!
$!   So, to check if two shareable images are compatible, first look at the
$!   EIHD$K_LIM section in the image header (IHD$K_LIM on VAX). Major ID
$!   MUST match. The minor ID is subject to the match control. Typically
$!   ISD$K_MATLEQ. This means the minor ID of the image being activated must
$!   be equal or greater than that of the image which was linked.
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$!
$!   Sample image header from Alpha ANALYZE/IMAGE:
$!
$!-------------------------------------------------------------------------
$! This is an OpenVMS Alpha image file
$!
$! IMAGE HEADER
$!
$!        Fixed Header Information
$!
$!                image format major id: 3, minor id: 0
$!                header block count: 3
$!                image type: shareable (EIHD$K_LIM)
$!                        global section major id: %X'04', minor id:
%X'0003E9'
$!                        match control: ISD$K_MATLEQ
$!                I/O channel count: default
$!
$!-------------------------------------------------------------------------
$!
$!
$!   Now to check the symbol vector. ANALYZE/IMAGE lists each symbol as a
$!   "Universal Symbol Specification". For example:
$!
$!        6) Universal Symbol Specification (EGSD$C_SYMG)
$!                data type: DSC$K_DTYPE_Z (0)
$!                symbol flags:
$!                        (0)  EGSY$V_WEAK      0
$!                        (1)  EGSY$V_DEF       1
$!                        (2)  EGSY$V_UNI       1
$!                        (3)  EGSY$V_REL       1
$!                        (4)  EGSY$V_COMM      0
$!                        (5)  EGSY$V_VECEP     0
$!                        (6)  EGSY$V_NORM      1
$!                psect: 0
$!                value: 592 (%X'00000250')
$!                symbol vector entry (procedure)
$!                        %X'00000000 0005F9AC'
$!                        %X'00000000 0008B980'
$!                symbol: "DBASIC$DATE_T"
$!
$!
$!   So here's how we do it...
$
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$ IF arch.EQS."IA64" THEN GOTO IA64Image
$ IF arch.NES."Alpha"
$ THEN
$   WRITE SYS$OUTPUT "Sanity check failure. Architecture=''arch' - shouldn't
get to here!"
$   EXIT
$ ENDIF
$!
$! Alpha specific
$!   Start with full analysis text
$!
$ ANALYZE/IMAGE/OUT=TMP_'pid'.ANL 'newimg'
$!
$!   Find image ident and match control
$ SEARCH/OUT=TMP_'pid'.GSM TMP_'pid'.ANL "global section major id:",-
                              "match control:"
$ OPEN/READ in TMP_'pid'.GSM
$ what="Global section ID"
$ READ/END=BadImg in line
$ maj=F$ELEMENT(1,"'",line)
$ min=F$ELEMENT(3,"'",line)
$ what="Match control"
$ READ/END=BadImg in line
$ mat=F$ELEMENT(1,"_",line)
$!
$ dat=""
$ CLOSE in
$!
$! Remove headers
$ ff[0,8]=12
$ SEARCH/MATCH=NOR/EXACT/OUTPUT=TMP_'pid'.AN1 TMP_'pid'.ANL -
    "''ff'","''newimg'","Analyze Image  ","ANALYZ "
$!
$! Get symbol blocks
$ SEARCH/EXACT/NOHEAD/OUTPUT=TMP_'pid'.AN2 TMP_'pid'.AN1 "symbol:
"/WINDOW=(5,0)
$!
$! Remove other junk
$ SEARCH/EXACT/NOHEAD/OUTPUT=TMP_'pid'.AN3 TMP_'pid'.AN2 "symbol","%X"
$
$!   We should now have a list of symbol definitions like this:
$!
$!                 value: 0 (%X'00000000')
$!                 symbol vector entry (constant)
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$!                       %X'00000000 00000000'
$!                       %X'00000000 0035800C' (3506188)
$!                 symbol: "C$_EPERM"
$!
$!
$!   Now parse the list. The objective is to produce a file with value,
$!   type, vector entry and symbol name on one line, then sort on value.
$!
$!   There are 3 types of symbol - "constant", "procedure" and "data cell"
$!   For constants, we need the second symbol vector entry value - as it's
$!   the constant value. For data cells the symbol vector gives us the offset
$!   to the data from the start of the shareable image. If any data cell
$!   vector entries are present, we generate a file containing the symbol
$!   names and offsets, sorted on offset. This can be used to generate a
$!   replica data area in the fake image.
$!
$ OPEN/READ in TMP_'pid'.AN3
$ OPEN/WRITE out TMP_'pid'.VEC
$ OPEN/WRITE dat TMP_'pid'.DATA
$!
$ genloop: READ/END=endgenloop in line
$   line=F$EDIT(line,"COLLAPSE")
$   IF F$LOCATE("value:",line).NE.0 THEN GOTO genloop
$   val=F$ELEMENT(1,"'",line)
$   READ/END=endgenloop in line
$   type=F$EDIT(line,"COLLAPSE")-"symbolvectorentry"
$   IF type.EQS.line THEN GOTO loop  ! Read symbol vector values
$   READ/END=endgenloop in line
$   v1=F$ELEMENT(1,"'",line)-" "
$   READ/END=endgenloop in line
$   v2=F$ELEMENT(1,"'",line)-" "
$   READ/END=endgenloop in line
$   line=F$EDIT(line,"COLLAPSE")
$   IF F$LOCATE("symbol:",line).NE.0 THEN GOTO loop
$   sym=F$ELEMENT(1,"""",line)
$   WRITE OUT "''val' ''v1' ''v2' ''sym' ''type'"
$   IF type.EQS."(datacell)"
$   THEN
$     WRITE dat "''v2' ''sym'"
$     dat="Data Present"
$   ENDIF
$ GOTO genloop
$ endgenloop:
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$!
$!  Generate a match control string. Note required spaces at beginning
$!  of line to keep it at the top of the vector file even after sorting
$!  Add comment tag to indicate if data is present
$!
$ matall="ALWAYS"
$ matequ="EQUAL"
$ matleq="LEQUAL"
$ matnev="NEVER"
$ match="  GSMATCH="+'mat'+",%X''maj',%X''min' ! "+dat
$ write out match
$ SET NOON
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ SORT TMP_'pid'.VEC  'name'.VEC
$ SORT TMP_'pid'.DATA 'name'.DATA
$ Cleanup:
$ err=$status
$ DELETE TMP_'pid'*.*;*
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ IF .NOT.err THEN EXIT
$ IF endphase.NES."VECTOR" THEN GOTO GEN
$ EXIT
$ BadImg: WRITE SYS$OUTPUT "Can't parse image ''what' -> ''line'"
$ EXIT
$
$ IA64Image:
$
$  ANALYZE/IMAGE/NOPAGE/SECTION=SYMBOL_VECTOR/OUT=TMP_'pid'.ANL 'newimg'
$  SEARCH/OUT=TMP_'pid'.GSM TMP_'pid'.ANL "Algorithm:","Major ID:","Minor ID:"
$
$ OPEN/READ in TMP_'pid'.GSM
$ what="Algorithm"
$ READ/END=BadIA64Img in line
$ algo=F$ELEMENT(1,":",F$EDIT(line,"COLLAPSE"))-"/"
$ what="Major ID"
$ READ/END=BadIA64Img in line
$ maj=F$ELEMENT(1,":",F$EDIT(line,"COLLAPSE"))-"."
$ what="Minor ID"
$ READ/END=BadIA64Img in line
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$ min=F$ELEMENT(1,":",F$EDIT(line,"COLLAPSE"))-"."
$ CLOSE/NOLOG in
$
$!
$!  Find and parse the symbol vector. Note that the IA64 output is much closer
$!  to what we actually need than the Alpha output, but for historical reasons
$!  it will be "translated" back to the same format the Alpha code generates.
$!
$ OPEN/READ in TMP_'pid'.ANL
$ OPEN/WRITE out TMP_'pid'.VEC
$ OPEN/WRITE dat TMP_'pid'.DATA
$
$ dat=""
$ FindIA64SymVec: READ/END=NoIA64Vec in line
$   IF F$LENGTH(line).EQ.0.OR.F$LOCATE("SYMBOL VECTOR",line).GT.0 THEN GOTO
FindIA64SymVec
$ READ/END=NoIA64Vec in line
$ READ/END=NoIA64Vec in line
$ IF F$LOCATE("FileAddr Offset",line).GT.0 THEN GOTO NoIA64Vec
$ READ/END=NoIA64Vec in line
$
$ IF p4.NES."" THEN SET VERIFY
$ GenIA64loop: READ/END=EndIA64Vec in line
$  IF F$LENGTH(line).EQ.0 THEN GOTO GenIA64loop
$  IF F$LOCATE("The analysis",line).EQ.0 THEN GOTO EndIA64Vec
$  FileAddr=F$EDIT(F$EXTRACT(0,8,line),"COLLAPSE")
$  Offset=F$EDIT(F$EXTRACT(9,8,line),"COLLAPSE")
$  indx=F$EDIT(F$EXTRACT(18,7,line),"COLLAPSE")
$  val=F$EDIT(F$EXTRACT(28,16,line),"COLLAPSE")
$  IF FileAddr.EQS."".OR.Offset.EQS."".OR.indx.EQS."" THEN GOTO GenIA64loop
$  type=F$EDIT(F$EXTRACT(45,11,line),"COLLAPSE")-"("-")"
$  IF type.EQS."" THEN type="SPARE"
$  DATA="(datacell)"
$  DATAABS="(constant)"
$  PROCEDURE="(procedure)"
$  SPARE="(spare)"
$  type='type'
$  gp=F$EDIT(F$EXTRACT(56,16,line),"COLLAPSE")
$  IF gp.EQS."" THEN gp="0000000000000000"
$  sym=F$EDIT(F$ELEMENT(1,"""",line),"COLLAPSE")
$  IF sym.EQS."""" THEN sym="*spare*"
$  WRITE OUT "''Offset' ''val' ''val' ''sym' ''type'"
$  IF type.EQS."(datacell)"
$  THEN
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$    WRITE dat "''val' ''sym'"
$    dat="Data Present"
$  ENDIF
$ GOTO GenIA64loop
$ EndIA64Vec:
$!
$!  Generate a match control string. Note required spaces at beginning
$!  of line to keep it at the top of the vector file even after sorting
$!  Add comment tag to indicate if data is present
$!
$ ALWAYS="ALWAYS"
$ EQUAL="EQUAL"
$ LESSEQUAL="LEQUAL"
$ NEVER="NEVER"
$ match="  GSMATCH="+'algo'+",''maj',''min' ! "+dat
$ write out match
$ SET NOON
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ SORT TMP_'pid'.VEC  'name'.VEC
$ SORT TMP_'pid'.DATA 'name'.DATA
$ CleanupIA64:
$ err=$status
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ DELETE TMP_'pid'*.*;*
$ IF .NOT.err THEN EXIT
$ IF endphase.NES."VECTOR" THEN GOTO GEN
$ EXIT
$
$ NoIA64Vec: $status=4
$  WRITE SYS$OUTPUT "Error - could not find symbol vector"
$ GOTO CleanupIA64
$
$ BadIA64Img: WRITE SYS$OUTPUT "Can't parse image ''what' -> ''line'"
$ EXIT
$!
$ GEN:
$ VecEntrySize=16
$ IF arch.EQS."IA64" THEN VecEntrySize=8
$!
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$!  Generate MACRO code and options file for the fake image
$!
$ v=F$PARSE("SYS$DISK:[]''name'.VEC")
$ IF F$SEARCH(v).EQS.""
$ THEN
$   WRITE SYS$OUTPUT "Cannot find file ''name'.VEC"
$   EXIT
$ ENDIF
$ IF .NOT.(exp.OR.force.OR.F$SEARCH("FAKE_''name'.MAR").EQS."") THEN GOTO
COMPILE
$ WRITE SYS$OUTPUT "Generating MACRO code for ''name'"
$ ON WARNING THEN GOTO VecCleanup
$ ON CONTROL_Y THEN GOTO VecCleanup
$!
$!  Parse vector file
$!
$ OPEN/READ vec 'v'
$ READ vec match ! First record is the match control string
$ n=F$PARSE(v,,,"NAME")
$ OPEN/WRITE opt FAKE_'n'.OPT
$!
$!  Options file contains:
$ WRITE opt match             ! Match control string
$ WRITE opt "FAKE_''n'"       ! reference to FAKE_''n' object module
$ WRITE opt "FAKE_RTL/SHARE"  ! FAKE_RTL support image
$!
$!  Check to see if data was found in the source image
$ hasdata=F$LOCATE("Data Present",match).LT.F$LENGTH(Match)
$!
$!  MACRO source contains a title
$ OPEN/WRITE mac FAKE_'n'.MAR
$ WRITE mac "    .TITLE FAKE_''n'"
$ WRITE mac "    .PSECT RWData,RD,WRT,NOEXE,PAGE"  ! read/write psect
$ IF hasdata
$ THEN    ! If data present, we need to map it as an init routine
$   WRITE mac "  .EXTRN LIB$INITIALIZE"
$   WRITE mac "  .PSECT
LIB$INITIALIZE,NOPIC,CON,REL,GBL,NOSHR,NOEXE,NOWRT,LONG"
$   WRITE mac "  .LONG MapData"
$ ENDIF
$ WRITE mac "    .PSECT ROData,RD,NOWRT,NOEXE,PAGE"      ! read only psect
$!
$!  string containing the name of the real shareable image
$ WRITE mac " .ALIGN LONG"
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$ WRITE mac " RealRTL: .ASCID /REAL_''n'/"
$!
$!  Macros to define a normal routine call and a JSB call
$!
$ COPY SYS$INPUT mac
$ DECK
    .PSECT $CODE,RD,NOWRT,EXE,PAGE

         .MACRO CallRoutine,Routine,Lab,Pref
           .PSECT ROData
           .ALIGN LONG   ; string for routine name
           'Pref'S_%EXTRACT(0,28,Routine) : .ASCID /'Routine'/
      .PSECT RWData
      .ALIGN LONG; storage for routine address init to 0
      'Pref'A_%EXTRACT(0,28,Routine) : .LONG 0
           .PSECT $CODE  ; entry point be careful with registers!
           .CALL_ENTRY LABEL='Lab',-
           MAX_ARGS=127,HOME_ARGS=TRUE,-
                 INPUT=<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>,-
                 OUTPUT=<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>,-
                 SCRATCH=<>,PRESERVE=<>
       PUSHL AP   ; arg list
       PUSHAL 'Pref'A_%EXTRACT(0,28,Routine) ; address of routine
       PUSHAB 'Pref'S_%EXTRACT(0,28,Routine) ; name of routine
       PUSHAB RealRTL   ; name of real shareable image
       CALLS #4,G^FAKE_LOGCALL ; dispatch to routine
            RET
         .ENDM

         .MACRO JSBRoutine,Routine,Lab,Pref
           .PSECT ROData
           .ALIGN LONG   ; string for routine name
           'Pref'S_%EXTRACT(0,28,Routine) : .ASCID /'Routine'/
      .PSECT RWData
      .ALIGN LONG; storage for routine address init to 0
      'Pref'A_%EXTRACT(0,28,Routine) : .LONG 0
           .PSECT $CODE; entry point be careful with registers!
           .CALL_ENTRY LABEL='Lab',-
                 INPUT=<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>,-
                 OUTPUT=<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>,-
                 SCRATCH=<>,PRESERVE=<>

       MOVQ R21,-(SP)   ; pass register arguments
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
          MOVQ R20,-(SP)
          MOVQ R19,-(SP)
          MOVQ R18,-(SP)
          MOVQ R17,-(SP)
          MOVQ R16,-(SP)
     PUSHAL 'Pref'A_%EXTRACT(0,28,Routine) ; address of routine
     PUSHAB 'Pref'S_%EXTRACT(0,28,Routine) ; name of routine
     PUSHAB RealRTL   ; name of real shareable image
     CALLS #15,G^FAKE_LOGJSB ; dispatch to routine
          RET
       .ENDM
$ EOD
$ abort="FALSE" ! assume all will work
$ NextAddr=0     ! set expected next vector address
$ vec="SYMBOL_VECTOR=(" ! header for symbol vector entries
$!
$!  Walk through the file
$ vecloop: READ/END=EndVec vec rec
$  Addr=%X'F$ELEMENT(0," ",rec)'     ! get address
$  v1=%X'F$ELEMENT(1," ",rec)'                ! get vector values
$  v2=%X'F$ELEMENT(2," ",rec)'               !
$  SName=F$ELEMENT(3," ",rec)        ! get symbol name
$  Typ=F$ELEMENT(4," ",rec)          ! get symbol type
$  IF F$INTEGER(Addr).EQ.F$INTEGER(NextAddr) THEN GOTO AddrOK
$!
$!  Not at expected location
$!
$    IF Addr.LT.NextAddr
$    THEN
$   ! This should never happen
$      WRITE SYS$OUTPUT -
   "FATAL ERROR - vector overlap. Expecting ''NextAddr', got ''Addr'"
$      WRITE SYS$OUTPUT "Jacketing not feasible!"
$      abort="TRUE"
$      GOTO EndVec
$    ENDIF
$!
$    WRITE SYS$OUTPUT -
       "Warning - vector hole before ''sname'. Expecting ''NextAddr', got
''Addr'"
$!
$!  Fill the vector with "spare" entries up to next entry
$!
$    FillLoop: IF NextAddr.GE.Addr THEN GOTO AddrOK
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$      va=F$FAO("!XL",NextAddr)
$      NextAddr=NextAddr+VecEntrySize
$      WRITE opt "''vec'SPARE) ! ''va' ***Fill to ''rec'"
$    GOTO FillLoop
$ AddrOK:
$  NextAddr=NextAddr+VecEntrySize    ! reset next expected address
$  GOSUB GenSym         ! generate the symbol name
$  IF typ.EQS."(procedure)"
$  THEN
$!  Options file entry is a PROCEDURE type symbol vector entry
$!
$    WRITE opt "''vec'''Lbl'=PROCEDURE)  ! ''Addr'"
$!
$!  OpenVMS naming convention is that JSB routines end with "_Rn" where
$!  "n" is the highest register modified. Check to see if this routine
$!  name ends in _Rn. If it does, generate a JSB entry instead of a
$!  CALL entry and issue a message.
$!
$    tag=F$EXTRACT(F$LENGTH(sName)-3,3,sName)
$    rn=tag-"_R"
$    IF f$EXTRACT(0,2,tag).EQS."_R".AND.(rn.GT.0).AND.(rn.LT.10)
$    THEN
$      WRITE SYS$OUTPUT "Assumed JSB routine ''sName'"
$      WRITE mac "JSBRoutine ''sName', ''Lbl', ''Pref' ; ''Addr'"
$    ELSE
$      WRITE mac "CallRoutine ''sName', ''Lbl', ''Pref' ; ''Addr'"
$    ENDIF
$!
$  ELSE IF typ.EQS."(constant)"
$  THEN
$!  Options file entry is a DATA symbol vector entry
$!
$    WRITE opt "''vec'''Lbl'=DATA)  ! ''Addr' CONST"
$!
$!  MACRO code is jusr a global symbol definition
$    WRITE mac "''Lbl'==''v2' ; ''Addr'"
$!
$  ELSE IF typ.EQS."(datacell)"
$  THEN
$!  Options file entry is a DATA symbol vector entry
$!
$    WRITE opt "''vec'''Lbl'=DATA)  ! ''Addr' VAR"
$!
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$!  Code will be generated later... see below
$!
$  ELSE IF typ.EQS."(spare)"
$  THEN
$      WRITE opt "''vec'SPARE) ! ''addr' SPARE"
$  ELSE
$!
$!  Shouldn't happen - flag the record
$    WRITE SYS$OUTPUT "Error unexpected vector entry type /''rec'/"
$  ENDIF
$  ENDIF
$  ENDIF
$  ENDIF
$ GOTO vecloop
$ EndVec:
$!
$!  Finished with vector file (input) and options file (output)
$ CLOSE vec
$ CLOSE opt
$ IF abort THEN EXIT
$ IF .NOT.hasdata THEN GOTO Finish
$!
$!    Generate code for data - we read the data listing, which is sorted
$!    on offset
$!
$   d=F$PARSE("SYS$DISK:[]''name'.DATA")
$   psize=8192   ! Page size in bytes (Alpha only!)
$   OPEN/READ dat 'd'
$
$!
$!    PSECT for exported data, aligned on Alpha page boundary
$   WRITE mac ".PSECT MapData,RD,WRT,NOEXE,13"
$   WRITE mac ".ALIGN 13"
$!
$!    Loop through data symbols
$!
$   READ/END=EndDat dat rec
$   b=0
$BlockLoop:      !
$     b=b+1
$     WRITE SYS$OUTPUT "Start of data block ''b' ''rec'"
$     WRITE mac "map''b'_start:"     ! label at start of data block
$     v=F$ELEMENT(0," ",rec)  ! Get offset
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$     val=%X'v'                ! translate to decimal
$     i=val-(val/psize*psize) ! check alignment from beginning of page
$     base=val-i       ! calculate base offset
$     hwm=val                  ! set high water mark
$     IF i.GT.0 THEN WRITE mac ".BLKB ''i'" ! pad if not aligned
$     sname=F$ELEMENT(1," ",rec)! get symbol name
$     basesym'b'=sname         ! generate name for string
$     DatLoop:                 ! loop on data entries
$       READ/END=EndDat dat rec      ! get next entry
$       v1=F$ELEMENT(0," ",rec)      ! get offset
$       val1=%X'v1'            ! convert to decimal
$       GOSUB GenSym           ! generate symbol name
$       alloc=""       ! assume no allocation
$       delta=val1-val         ! calculate size
$!        If entry is non zero set allocation
$       IF delta.GT.0.AND.delta.LT.psize THEN alloc=".BLKB ''delta'"
$       WRITE mac "''Lbl':: ''alloc' ;''v'" ! declare symbol
$       v=v1                   ! set for next record offset (HEX)
$       val=val1       ! set next record (decimal)
$       sname=F$ELEMENT(1," ",rec)   ! next record name
$       IF delta.GT.psize      ! Delta larger than page, start next block
$       THEN
$         WRITE SYS$OUTPUT "Hole in data exceeded threshold at ''rec'"
$    GOSUB EndSec              ! Fill end of data section
$         WRITE mac ".ALIGN 13"
$         GOTO BlockLoop       ! next block
$       ELSE
$         hwm=val      ! set highwater mark
$       ENDIF
$   GOTO DatLoop
$
$ GenSym:
$!   This is needed to deal with upper and lowercase symbol names
$!   since MACRO cannot generate lowercase names, and images like DECC$SHR
$!   contain symbols which differ only in case, we need to generate "fake"
$!   names for lowercase symbols that won't clash with uppercase.
$!   Here we do that by prefixing the symbol with "L_" and truncating at
$!   31 characters. Uppercase only names go straight through.
$!
$   IF F$EDIT(sName,"UPCASE").NES.sName
$   THEN
$     Pref="L"
$     Lbl=F$EXTRACT(0,31,"''Pref'_''sName'")
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$   ELSE
$     Pref="U"
$     Lbl=sName
$   ENDIF
$ RETURN
$
$ EndSec: ! End of data section
$   size=hwm-base       ! calculate total data size
$   pages=(size+psize)/psize  ! work out size in pages
$   pages'b'=pages      ! save per block
$   extra=pages*psize-size    ! calculate padding at end
$   alloc=""
$   IF extra.GT.0 THEN alloc=".BLKB ''extra'"
$   WRITE mac "map''b'_pad: ''alloc'"
$   WRITE mac "map''b'_end:"  ! label at end of section
$ RETURN
$
$ EndDat: ! End of all data
$   CLOSE dat    ! close input file
$   GOSUB GenSym
$   WRITE mac "''Lbl'::;''v'" ! define last label
$   GOSUB EndSec ! fill end of section
$   WRITE mac "mapeop: .LONG" ! mark end of all data
$!
$!   Now generate init routine
$!
$ WRITE mac "    .PSECT $CODE         "
$ WRITE mac "    .ENTRY MapData,^M<>"
$   c=1   ! for each data block
$   gsdloop:
$       WRITE mac ";MAP GLOBAL SECTION ''n'''c'_DATA"    ! comment
$       sym=basesym'c'  ! get symbol name
$       WRITE mac " .PSECT ROData"   ! define name of global section
$  WRITE mac " .ALIGN LONG"
$       WRITE mac " gsd''c':.ASCID /''n'''c'_DATA/"
$  WRITE mac " .ALIGN LONG"   ! define name of symbol in
$!                            ! real image at start of block
$       WRITE mac " bsym''c':.ASCID /''sym'/"
$
$       pagelets=pages'c'*16  ! calculate pagelet size
$       bytes=pages'c'*psize  ! calculate size in bytes
$  WRITE mac " .PSECT $CODE"
$  WRITE mac " PUSHAL map''c'_end"         ! address of end
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$   WRITE mac " PUSHAL map''c'_start" ! address of start
$   WRITE mac " PUSHL #''pagelets'"          ! size
$   WRITE mac " PUSHAB bsym''c'"             ! start symbol
$   WRITE mac " PUSHAB gsd''c'"              ! section name
$   WRITE mac " PUSHAB RealRTL"              ! real image name
$   WRITE mac " CALLS #6,G^FAKE_MAP_DATA"    ! map data
$       c=c+1       ! repeat for next section
$    IF c.LE.b THEN GOTO gsdloop
$    WRITE mac "RET"     ! end of init routine
$ Finish:
$ WRITE mac "    .END"  ! end of macro module
$ CLOSE mac
$ IF endphase.NES."VECTOR" THEN GOTO COMPILE
$ EXIT
$!
$ VecCleanup:    ! close any files that might be open
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG DAT
$ IF F$TRNLNM("MAC").NES."" THEN CLOSE/NOLOG MAC
$ IF F$TRNLNM("VEC").NES."" THEN CLOSE/NOLOG VEC
$ IF F$TRNLNM("OPT").NES."" THEN CLOSE/NOLOG OPT
$ EXIT
$!
$ COMPILE:
$   IF exp.OR.force.OR.F$SEARCH("FAKE_''name'.OBJ").EQS.""
$   THEN
$     WRITE SYS$OUTPUT "Compiling FAKE_''name'"
$     opt=""
$     IF dbg THEN opt="/NOOPT/DEBUG"
$     MACRO'opt' FAKE_'name'
$   ENDIF
$ IF endphase.EQS."COMPILE" THEN EXIT
$ LINK:
$   IF exp.OR.force.OR.F$SEARCH("FAKE_''name'.EXE").EQS.""
$   THEN
$     WRITE SYS$OUTPUT "Linking FAKE_''name'"
$     opt=""
$     IF dbg THEN opt="/DEBUG"
$     LINK/SHARE'opt' FAKE_'name'/OPT
$   ENDIF
$ IF endphase.EQS."LINK" THEN EXIT
$ USE:
$   DEFINE/NOLOG REAL_'name' 'img'
$   fake_img=F$ELEMENT(0,";",F$SEARCH("FAKE_''name'.EXE"))
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
$   fake_'name'=="DEFINE/USER/NOLOG/NAME=CONFINE ''name' ''fake_img'"
$   real_'name'=="DEASSIGN ''name'"
$ EXIT
$
$ CreateOpt:
$!  Options file for FAKE_RTL support routines
$!
$ CREATE/LOG FAKE_RTL:.OPT
$ DECK
GSMATCH=LEQUAL,42,1     ! arbitrary GSMATCH
FAKE_RTL         ! link FAKE_RTL.OBJ
REAL_LIBRTL/SHARE      ! link against real LIBRTL and LIBOTS (important!)
REAL_LIBOTS/SHARE
SYMBOL_VECTOR=(-
FAKE_DOCALL=PROCEDURE,-       ! dispatch to routine by CALL
FAKE_LOGCALL=PROCEDURE,-      ! log argument list and dispatch by CALL
FAKE_DOJSB=PROCEDURE,-        ! dispatch to routine by JSB
FAKE_LOGJSB=PROCEDURE,-       ! log argument list and dispatch by JSB
FAKE_PUT=PROCEDURE,-          ! write a string to log file
FAKE_FIS=PROCEDURE,-          ! jacket for LIB$FIND_IMAGE_SYMBOL
FAKE_CALL=PROCEDURE,-         ! jacket for CALLG (OTS$EMUL_CALL)
FAKE_MOVE=PROCEDURE,-         ! jacket for MOVC3
FAKE_OUT=PROCEDURE,-          ! jacket for LIB$PUT_OUTPUT
FAKE_LOG=PROCEDURE,-          ! write to log file if logging enabled
FAKE_MAP_DATA=PROCEDURE-      ! map data area
)

$ EOD
$ RETURN
$
$ CreateMac:
$!  Source code for FAKE_RTL - shareable image with support routines
$!
$ CREATE/LOG FAKE_RTL:.MAR
$ DECK
   .TITLE FAKE_RTL
;
; Support routines for fake RTLs
;
   .PSECT ROData,RD,NOWRT,NOEXE,QUAD ; declare PSECTs
   .PSECT RWData,RD,WRT,NOEXE,QUAD
   .PSECT $CODE,RD,NOWRT,EXE
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
        .MACRO ASCIDStr name,val     ; define ASCID string with alignment
    .PSECT ROData
    .align long
    'name': .ASCID "'val'"
    .ENDM

;   FAO control strings for logging
;
ASCIDStr start,  ^?!ASArg tracing started at !%D?
ASCIDStr blank,  ^?!/?
ASCIDStr header, ^?!AS!AS at !%T?
ASCIDStr argcnt, ^?!AS!AS called with !UL arg!%S?
ASCIDStr argret, ^?!AS!AS returning !UL arg!%S?
ASCIDStr JSBhead,^?!AS!AS JSB call at !%T?
ASCIDStr return, ^?!AS!AS returned: !8XL at !%T?
ASCIDStr valarg, ^?!AS !3UL !8XL?
ASCIDStr refarg, ^?!AS !3UL !8XL => !XL?
ASCIDStr dscarg, ^?!AS!17< !>!XL => !AS?
ASCIDStr azsarg, ^?!AS!15< !>=/!AZ/?
ASCIDStr strarg, ^?!AS!15< !>=/!AF/?
ASCIDStr lstarg, ^?!AS!15< !>=/!AF...?
ASCIDStr eol,    ^?!ASexit status: !XL?
ASCIDStr regs1,  ^?!ASR0:!XL R1:!XL R2:!XL R3:!XL R4 :!XL R5: !XL?
ASCIDStr regs2,  ^?!ASR6:!XL R7:!XL R8:!XL R9:!XL R10:!XL R11:!XL?
ASCIDStr JSBReg1,^?!ASR16:!XQ R17:!XQ R18:!XQ?
ASCIDStr JSBReg2,^?!ASR19:!XQ R20:!XQ R21:!XQ?
ASCIDStr MapData,^?!ASMapped Data !AS real:!XL:!XL => fake:!XL:!XL?
ASCIDStr dumpargs,    <FAKE_DUMPARGS>

;
;   Init routine
        .EXTRN LIB$INITIALIZE
        .PSECT LIB$INITIALIZE,NOPIC,CON,REL,GBL,NOSHR,NOEXE,NOWRT,LONG
        .LONG FAKE_INIT



    .PSECT RWData
.align long
tracing:.LONG 0 ; set to 1 if tracing enabled
skip:   .LONG 0  ; block recursion
depth:  .LONG 0 ; nesting depth
exhsta: ; exit handler
    .LONG 1       ; buffer for final status
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
desblk: .LONG 0 ; Exit handler descriptor block
hndadr: .LONG 0 ; handler address
argc:   .LONG 1 ; argument count
staadr: .ADDRESS exhsta ; pointer to final status buffer
        .LONG 0

.ALIGN QUAD             ; FAB and RAB for output file
LogFAB:   $FAB FNM=ARGDUMP , DNM=<SYS$DISK:[].LOG> , RAT=CR
LogRAB: $RAB FAB=LogFAB

    ; desriptors for formatting strings
outlen: .WORD
.ALIGN QUAD
outdsc: .LONG ^X010E0000
    .ADDRESS outbuf
dscdsc: .LONG ^X010E0000
    .ADDRESS dscbuf

MaxPad=32 ; padding string for start of log strings
Padding: .ASCID /                                                     /

maxstr=48
outmax=128                    ; buffer length for strings
outbuf: .BLKB outmax
dscbuf: .BLKB outmax
$LIBDEF

;
;  Macro to format and output a string using FAO
;  Arguments are the FAO control string and corresponding FAO arguments

    .MACRO formout formstr, a1=#0,a2=#0,a3=#0,a4=#0,a5=#0,a6=#0,?OK
      MOVW Depth,Padding       ; set length of pad string
      CMPW #MaxPad,Depth       ; check max deptch
      BGTR OK
      MOVW #MaxPad,Padding     ; limit to macimum
    'OK':
      MOVW #outmax,outdsc      ; set descriptor to maximum size
      $FAO_S CTRSTR=formstr, OUTLEN=outlen, OUTBUF=outdsc, -
           P1=#Padding,P2=a1,P3=a2,P4=a3,P5=a4,P6=a5,p7=a6
      MOVW outlen,outdsc       ; set descriptor to actual size
      PUSHAB outdsc
      CALLS #1,FAKE_PUT ; write to log file
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
        .ENDM

        .PSECT $CODE

        .ENTRY FAKE_EXIT,^M<>      ;  Exit handler
        BLBC tracing,NoClose       ;    skip if not tracing
          formout eol,exhsta       ;    Format final status string
          $CLOSE FAB=LogFAB ;    Close log file
NoClose:RET

        .ENTRY FAKE_INIT,^M<>      ; setup log file
        PUSHAB dumpargs            ; check logical name to see if tracing enabled
        CALLS #1,G^LIB$GET_LOGICAL ; existence only
        BLBC R0,NoTrace
          MOVL #1,tracing   ; set tracing flag
          $CREATE FAB=LogFAB       ; Create and connect log file
               BLBC R0,fail
                 $CONNECT RAB=LogRAB
          BLBC R0,fail
                 MOVAB FAKE_EXIT,hndadr ;    init exit handler control block
          PUSHAB desblk
          $DCLEXH_S DESBLK=desblk  ; declare exit handler
          formout start            ; write start message
NoTrace:RET

        .MACRO GetRoutineAddress,?Ready
        ;  macro to check if a routine address is already known, and find
        ;  it if not. Note we must preserve all registers!

          CMPL #0,@12(AP)   ; is address zero?
          BNEQ Ready        ; no, already known
            PUSHR #^M<R0,R1>       ; save R0 & R1
              PUSHAL @12(AP)       ;  address buffer
              PUSHAL @8(AP) ;  symbol name
              PUSHAL @4(AP) ;  image name
              CALLS #3,G^LIB$FIND_IMAGE_SYMBOL
            POPR  #^M<R0,R1>       ; restore registers
             'ready':
          PUSHL  @12(AP)    ; return address on stack
        .ENDM

        .MACRO LogReturn,?done
        ;  macro to log the return from a routine
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
;  we must preserve all registers
  BLBC tracing,done ; skip if not tracing
       BLBS skip, done      ; recursion block
    MOVL #1,skip     ; protect from recursion
      PUSHR #^M<R0,R1>      ; save R0 & R1
        PUSHL R0     ; log return value
        PUSHAB @8(AP)       ; routine name
        CALLS #2,FAKE_RETURN      ; write return record
      POPR  #^M<R0,R1>      ; restore registers
         CLRL skip          ; unblock recursion
  'done':
.ENDM



      .CALL_ENTRY,MAX_ARGS=4,HOME_ARGS=TRUE,LABEL=FAKE_DOCALL
;
; Args
;   Image Name
;       Routine Name
;       Address of Routine Address
;   AP
;
   GetRoutineAddress
   CALLG @16(AP),@(SP)+       ; note weird mode!
   RET

      .CALL_ENTRY,MAX_ARGS=4,HOME_ARGS=TRUE,LABEL=FAKE_LOGCALL
;
; Args
;   Image Name
;       Routine Name
;       Address of Routine Address
;   AP
;
   BLBC tracing,DoneEntry     ; skip logging if not tracing
   BLBS skip, DoneEntry       ; skip logging if recursing
     MOVL #1,skip             ; block recursion
       PUSHR #^M<R0,R1>       ; preserve registers
         PUSHL #1
         PUSHAL @16(AP)       ; arg list
         PUSHAB @8(AP) ; routine name
         CALLS #3,FAKE_DUMPARGS
       POPR  #^M<R0,R1>       ; restore registers
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    CLRL  skip        ; unblock recursion
      DoneEntry:

  ADDW2 #2,depth              ; increment nesting detch
  GetRoutineAddress   ; get address
  CALLG @16(AP),@(SP)+        ; dispatch
  SUBW2 #2,depth              ; restore nesting depth
  BLBC tracing,DoneExit       ; skip logging if not tracing
  BLBS skip, DoneExit ; skip logging if recursing
    MOVL #1,skip              ; block recursion
      PUSHR #^M<R0,R1>        ; preserve registers
        PUSHL #0
        PUSHAL @16(AP)        ; arg list
        PUSHAB @8(AP) ; routine name
        CALLS #3,FAKE_DUMPARGS
      POPR  #^M<R0,R1>        ; restore registers
    CLRL  skip        ; unblock recursion
      DoneExit:
  LogReturn
  RET


      .CALL_ENTRY,MAX_ARGS=16,HOME_ARGS=TRUE,LABEL=FAKE_DOJSB
;
; Args
;   Image Name
;       Routine Name
;       Address of Routine Address
;   R16..R21 as quadwords
;
  GetRoutineAddress
  MOVQ 16(AP),R16     ; restore register arguments
  MOVQ 24(AP),R17
  MOVQ 32(AP),R18
  MOVQ 40(AP),R19
  MOVQ 48(AP),R20
  MOVQ 56(AP),R21
      JSB @(SP)+      ; dispatch to routine
  RET

      .CALL_ENTRY,MAX_ARGS=16,HOME_ARGS=TRUE,LABEL=FAKE_LOGJSB
;
; Args
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
;   Image Name
;       Routine Name
;       Address of Routine Address
;   R16..R21 as quadwords
;

    BLBC tracing,DoneJSBEntry  ; skip if not tracing
    BLBS skip, DoneJSBEntry          ; skip if recursing
      MOVL #1,skip                    ; block recursion
        PUSHR #^M<R0,R1>            ; save registers
          PUSHAQ 16(AP)             ; pass address of first register arg
          PUSHAB @8(AP)        ; routine name
          CALLS #2,FAKE_JSBENTRY      ; log the entry
        POPR  #^M<R0,R1>              ; restore registers
      CLRL  skip               ; unblock recursion
         DoneJSBEntry:
    GetRoutineAddress
    ADDW2 #2,depth             ; increment nesting depth
    MOVQ 16(AP),R16            ; restore register args
    MOVQ 24(AP),R17
    MOVQ 32(AP),R18
    MOVQ 40(AP),R19
    MOVQ 48(AP),R20
    MOVQ 56(AP),R21
         JSB @(SP)+            ; dispatch to routine
    SUBW2 #2,depth             ; restore nesting depth
    LogReturn
    RET

    .ENTRY FAKE_RETURN,^M<r2,r3>
; writes routine name, return value and time stamp
; Arguments
;    4(AP) = String descriptor, routine name
;    8(AP) = return value
;
      PUSHR #^M<R0,R1>  ; save registers
        formout return,4(AP),8(AP),#0
      POPR #^M<R0,R1>   ; restore registers
      formout regs1,r0,r1,r2,r3,r4,r5 ; dump registers
      formout regs2,r6,r7,r8,r9,r10,r11
      formout blank
    RET
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    .ENTRY FAKE_JSBENTRY,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>

    PUSHR #^M<R0,R1>
      formout blank
      formout JSBhead,4(AP)    ; write header
    POPR #^M<R0,R1>
    formout regs1,r0,r1,r2,r3,r4,r5         ; dump registers
    formout regs2,r6,r7,r8,r9,r10,r11
    MOVL 8(AP),R10      ; get address of first register arg
    CLRL R9                     ; reset arg counter
    .DISABLE FLAGGING   ; supress  QUADMEMREF informationals
    MOVQ (R10)+,R2              ; dump arguments
    JSB DumpArg
    MOVQ (R10)+,R2
    JSB DumpArg
    MOVQ (R10)+,R2
    JSB DumpArg
    MOVQ (R10)+,R2
    JSB DumpArg
    MOVQ (R10)+,R2
    JSB DumpArg
    MOVQ (R10)+,R2
    .ENABLE FLAGGING    ; restore informationals
    JSB DumpArg
    RET

    .ENTRY FAKE_DUMPARGS,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10>
;
; FAKE_DUMPARGS  dumps argument list
; Arguments
;    4(AP) = String descriptor, routine name
;    8(AP) = Argument pointer
;   12(AP) = Flag 1=write header
;
    BLBC 12(AP),NoHead
      PUSHR #^M<R0,R1>
        formout blank
        formout header,4(AP)    ; write header (routine name & timestamp)
      POPR #^M<R0,R1>
      formout regs1,r0,r1,r2,r3,r4,r5 ; dump registers
      formout regs2,r6,r7,r8,r9,r10,r11
    NoHead:
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    MOVL   8(AP),R8            ; get argument pointer
    PROBER #0,#4,(R8)   ; readable?
    BNEQ   DoArgs       ; Arg list is readable
         formout argcnt,4(AP),#0 ; zero args passed or unreadable argument list
    RET
DoArgs:
    MOVL   (R8)+,R7            ; get argument count
    CLRL   R9           ; init counter
    BLBC 12(AP),retargs
      formout argcnt,4(AP),R7
      BICL   #^XFFFFFF80,R7    ; sanity check to 127 args
      BRB ArgLoop
    retargs:
          formout argret,4(AP),R7
      BICL   #^XFFFFFF80,R7    ; sanity check to 127 args

         ArgLoop:
      SOBGEQ R7,MoreArg ; count down arguments
        RET
          MoreArg:
      MOVL (R8)+,R2            ; get next arg
      JSB DumpArg
    BRB ArgLoop



DumpArg:  .JSB_ENTRY input=<r2,r9>,output=<r9>
;
; arg value in r2
; arg number is in R9 (pre incremented and returned)
;
; Arguments are written in HEX.
; If address is valid, reference value will be written as a hex longword.
; If address looks like a valid descriptor the data will be converted to a
;  string written
; If address looks like a printable ASCII string it will be written to
;  a maximum of MAXSTR characters

    INCL R9             ; update counter
         PROBER #0,#4,(R2)     ; readable?
    BEQL ByValue        ; no, assume by value

          ; readable, assume by reference
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    formout refarg,r9,r2,(r2) ; write initial reference
;
;  We now want to see if the argument could be a descriptor. Use LIB$CVT_DX_DX
;  to convert to string. If the call succeeds we had a valid descriptor and
;  now have a string to display.
;  In the vast majority of cases this will be a string descriptor, but
;  LIB$CVT_DX_DX gives us all other scalar types for free.
;
    MOVW #maxstr,dscdsc ; set descriptor to maximum size
    PUSHAW outlen       ; variable to receive output length
    PUSHAB dscdsc       ; output descriptor
    PUSHL  R2           ; input descriptor?
    CALLS #3,G^LIB$CVT_DX_DX ; convert
    BLBS R0,IsDescr     ; conversion succeeded, valid descriptor
    CMPL R0,#LIB$_OUTSTRTRU    ; conversion OK, but truncated
    BEQL IsDescr

    ; NotDescriptor, might be a string
         CLRL R3         ; init counter
    MOVL R2,R4    ; get copy of start address
         CMPB (R4),#32   ; GEQ space character?
    BLSSU finarg  ; no, skip arg
    CMPB (R4)+,#126     ; printable? (and step to next character)
    BGTRU finarg  ; no, skip arg
chrloop:INCL R3         ; we now have at least one printable character
    PROBER #0,#4,(R4) ; check next character is readable
    BEQL str      ; if not, write what we've found as a string
    CMPB (R4),#0  ; NUL?
    BEQL AZstr    ; assume ASCIZ and write it
    CMPL R3,#maxstr     ; more than threshold?
    BGEQ longstr  ; yes, write maximum string
    CMPB (R4),#32 ; still printable?
    BLSSU str     ; no, write what we've found
    CMPB (R4)+,#126     ; still printable?
    BGEQU str     ; no, write what we've found
    BRB chrloop   ; keep looking


IsDescr:; valid descriptor
    MOVW outlen,dscdsc  ; set output descriptor to actual length
    formout dscarg,4(R2),#dscdsc ; output string
    RSB
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
str: ; string argument size in R3, address in R2
    formout strarg,R3,R2
    RSB

longstr: ; long string size in R3, address in R2
    formout lstarg,R3,R2
    RSB

AZstr:  ; nul terminated string, address in R2
    formout azsarg,R2
         RSB

ByValue: ; assume argument by value. count in R9, value in R2
    formout valarg,r9,r2
finarg:   RSB

;
; Routines to write the log file. Name is by logical name FAKE_ARGDUMP
; default is in current default directory, type ".LOG". File is created
; in FAKE_INIT and closed on image exit. Final status is written
; before closing.

    .ENTRY FAKE_PUT,^M<R2,R3,R4,R5,R6>
;
;    Write one string to file, argument passed by descriptor
;
    MOVL 4(AP),R1 ; get desriptor address
    MOVW (R1), LogRAB+RA
B$W_RSZ    ; set size
    MOVL 4(R1),LogRAB+RAB$L_RBF       ; set buffer address
    $PUT RAB=LogRAB      ; write record
    RET

    .ENTRY FAKE_LOG,^M<R2,R3,R4,R5,R6>
;
;    Write one string to file, if logging enabled argument passed by
descriptor
;
    BLBC tracing,NoLog
      MOVL 4(AP),R1      ; get desriptor address
      MOVW (R1), LogRAB+RAB$W_RSZ     ; set size
      MOVL 4(R1),LogRAB+RAB$L_RBF     ; set buffer address
      $PUT RAB=LogRAB    ; write record
NoLog:    RET
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
fail:     $EXIT_S R0    ; error exit
    RET

    .PSECT RWdata
fadr_s:   .LONG  ; data address range for fake image
fadr_e:   .LONG
radr_s:   .LONG  ; data address range for real image
radr_e:   .LONG

    .PSECT $CODE
    .ENTRY FAKE_MAP_DATA,^M<>
;
;   Routine to map data. A range of new addresses is created as a global
;   section. Data is copied from the old range to the global section, then
;   the old range is mapped to the section.
; Args
;   Image Name
;        Section name
;   base symbol name
;        Number of pages
;   start address
;   end address
;
    MOVL 20(AP),fadr_s  ; set fake address range
    MOVL 24(AP),fadr_e
    DECL fadr_e         ; adjust to last byte on previous page
    $DELTVA_S inadr=fadr_s  ; delete virtual addresses
    BLBC R0,MapFail
;
;          Create global section
    $CRMPSC_S inadr=fadr_s, gsdnam=@8(AP),pagcnt=16(AP),-
              flags=#SEC$M_GBL!SEC$M_PAGFIL!SEC$M_WRT
    BLBC R0,MapFail
    PUSHAL radr_s       ; find start of data
    PUSHAB @12(AP)             ; symbol name
    PUSHAB @4(AP)       ; image name
    CALLS #3,G^LIB$FIND_IMAGE_SYMBOL
    BICL  #^X01FFF,radr_s      ; page align
    MOVL  16(AP),radr_e ; get data size in pagelets
    MULL2 #512,radr_e   ; convert to bytes
    MOVC3 radr_e,@radr_s,@fadr_s ; copy data
    ADDL2 radr_s,radr_e ; calculate end address
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

```
    DECL  radr_e        ; adjust to last byte on previous page
    $DELTVA_S inadr=radr_s     ; delete address range
    BLBC R0,MapFail
;         map to section
    $MGBLSC_S inadr=radr_s, gsdnam=@8(AP), -
             flags=#SEC$M_GBL!SEC$M_PAGFIL!SEC$M_WRT
    BLBC R0,MapFail
    BLBC tracing,NoLogMap      ; If logging enabled, log mapping
      formout MapData,8(AP),radr_s,radr_e,fadr_s,fadr_e
    MOVL radr_s,R0             ; return real base address
NoLogMap:RET
MapFail: $EXIT_S R0
    RET

;
; Jacket routines.
; These are placed in this module to allow "fake" RTLs to access them, even
; if it would imply a self reference. For example, a fake LIBRTL cannot call
; LIB$FIND_IMAGE_SYMBOL directly. This image is linked against REAL_LIBRTL and
; REAL_LIBOTS so it always uses the "real" routine.
;
;  LIBRTL routines
         .CALL_ENTRY,MAX_ARGS=8,HOME_ARGS=TRUE,LABEL=FAKE_FIS
    CALLG (AP),G^LIB$FIND_IMAGE_SYMBOL
    RET
         .CALL_ENTRY,MAX_ARGS=8,HOME_ARGS=TRUE,LABEL=FAKE_OUT
    CALLG (AP),G^LIB$PUT_OUTPUT
    RET
;
;  LIBOTS emulated instructions.
         .CALL_ENTRY,MAX_ARGS=2,HOME_ARGS=TRUE,LABEL=FAKE_CALL
    CALLG @8(AP),@4(AP)
    RET
         .CALL_ENTRY,MAX_ARGS=3,HOME_ARGS=TRUE,LABEL=FAKE_MOVE
    MOVC3 4(AP),@8(AP),@12(AP)
    RET

    .END
$ EOD
$ RETURN
```


© Copyright 2006 Hewlett-Packard Development Company, L.P.

For more information

OpenVMS Linker Utility Manual, Order Number: AA--PV6CD—TK

http://h71000.www7.hp.com/doc/73final/4548/4548PRO.HTML

HP OpenVMS Calling Standard, Order Number: AA--QSBBE--TE

http://h71000.www7.hp.com/doc/82final/5973/5973PRO.HTML

HP OpenVMS Programming Concepts Manual

http://h71000.www7.hp.com/doc/82FINAL/5841/5841PRO.HTML

HP OpenVMS RTL Library (LIB$) Manual, Order Number: AA-QSBHE-TE

http://h71000.www7.hp.com/doc/82final/5932/5932PRO.HTML

HP OpenVMS MACRO Compiler Porting and User's Guide, Order Number: AA--PV64E--TE

http://h71000.www7.hp.com/doc/82final/5601/5601PRO.HTML

John Gillings
Software Systems Consultant, OpenVMS Ambassador
Hewlett-Packard Pty Limited
OpenVMS Group, Customer Support Centre
Sydney, Australia

OpenVMS homepage:	http://www.hp.com/products/openvms
OpenVMS Times:	http://www.hp.com/products1/evolution/customertimes
OpenVMS Patches:	http://itrc.hp.com/
OpenVMS Forum:	http://forums.itrc.hp.com/service/forums/familyhome.do?familyId=288

.

© Copyright 2006 Hewlett-Packard Development Company, L.P

OpenVMS Technical Journal V7

WASD in SOAP/XML Transaction-Oriented Environments

Authors: Mark Daniel, Jeremy Begg, Ben Burke, Howard Taylor

Overview

The WASD VMS Hypertext Services Package is a general-purpose Web service with sufficient additional specialized capabilities to find a niche in transaction-oriented network services. This article explains how to use WASD to deliver SOAP remote procedure calls. The first section describes the persistent scripting mechanism provided by WASD. This is followed by two case studies that explain the development of SOAP-based transaction systems in which WASD provides the Web infrastructure to interface front- and back-end processing. The case studies describe technology choices, implementation details, system performance, and general outcomes.

WASD PERSISTENT SCRIPTING

Author: Mark Daniel

Initial development of the WASD VMS Hypertext Services Package began in 1994. This was an early venture into the then-promising new world of distributed networked information resources. To ensure efficiency and low latency, the key design decisions were:

- Use as few layers as possible between the Web services and VMS
- Use VMS's AST delivery mechanism for servicing multiple, concurrent requests in an event-driven manner
- Provide extensibility by using an effective, native scripting environment

There was no concern about making the solution "too closely integrated with VMS"—it couldn't be too close! Finally, I was free to play with as much of OpenVMS as I possibly could. I particularly liked that one.

After eighteen months of development, the HFRD VMS Hypertext Services Package (as it was then called) was released as open source with the OpenVMS Freeware CD v3.0. At that point, it had much of the core functionality and support programs that is in WASD today, such as concurrent request processing, file serving, CGI scripting environment, Conan the Librarian, and Bookreader. After the product was released to the public, development became user-initiated. In subsequent years, SYSUAF authentication, SSL, and proxy services were introduced, as well as a host of refinements to existing functionality. Thanks to a truck-parts supplier in Michigan, who in 1997 suggested the need

© Copyright 2006 Hewlett-Packard Development Company, L.P

for a low-latency, high-efficiency, persistent scripting environment, the WASD facility now called CGIplus was developed.

Persistent Processes

WASD scripting was originally written using a generic CGI mechanism. CGI scripting is a standards-defined method for providing request data to the scripting environment and for conveying the response to the server and returning it to the client. This method has traditionally required the creation of a new process for each request, resulting in three critical performance issues:

- OpenVMS process creation is notoriously expensive. Dealing with this required the elimination of unnecessary process creation. In practice, most scripts do not alter the process environment significantly enough to warrant disposing of the process itself. CGIplus allows successive CGI scripts to reuse the process, after some minimal housekeeping each time that the server manages the process from idle to occupied and back to idle again.

  There have been a number of proprietary variations on CGI designed to improve performance through process context persistence (for example, ISAPI and FastCGI). CGIplus is similar, but more straight-forward than these approaches. In fact, the WASD package provides implementations of ISAPI and FastCGI built on its own CGIplus technology.

- Many scripts are expensive to instantiate. To resolve this, the script instantiates itself and its required resources within the process and executes multiple requests without performing significant initialization each time. (This is different from executing multiple different scripts through the one process.) Other resources that are used by scripts, such as databases, have their own instantiation and rundown demands. Some scripting environments therefore perform best if a script is started the once and then used repeatedly over multiple requests. Under WASD, the server manages which script is currently in use in which process, and directs relevant requests to that process. This is known as a CGIplus script.
- Interpreters like Java, Perl, PHP, and Python engines require significant resources and exhibit noticeable latency when initializing. To resolve this issue, CGIplus allows the scripting engine to be given successive, different scripts to execute without the requirement to reinstantiate the underlying interpreter each time. Again, the server manages which processes have which engines, and allocates one appropriately to each request. This is called the CGIplus Run Time Environment (RTE).

Use of CGIplus scripts and RTEs provides up to ten times the reduction in script latency (yes, that's 10x!) with the resulting improvement to throughput. The cost of this solution is a slight increase of complexity in coordinating script initiation and in obtaining the CGI variable information using a separate data stream. CGIplus is CGI, plus lower latency, plus greater throughput, plus far less system impact.

CGIplus Operation

The WASD CGI implementation defines a collection of environment variable names containing associated strings, providing a representation of the request parameters. These are known as the CGI variables, which WASD implements under standard CGI using DCL symbol names and values. A set of standard responses convey HTTP status information about the relative success of the request and any associated response content.

All WASD CGI/CGIplus/RTE scripts (with the exception of DECnet/OSU emulation scripts) use mailboxes for Inter-Process Communication (IPC) between server and script. These record-oriented, general-purpose communication devices are suitable for I/O in all OpenVMS environments (DCL, standard utilities, compiled programs) and made available using SYS$OUTPUT, SYS$INPUT, and CGIPLUSIN. WASD automatically adjusts carriage-control for output according to response content type, and also allows you to make fine adjustment of such behaviors under script control. Standard CGI scripts require no modification for use with WASD.

A CGIplus script is indicated to the server by a configuration mapping rule that modifies the management of the script and its associated process. With CGIplus, the request data is provided to the script using the same set of familiar CGI variables. However, instead of being implemented using DCL symbols, the CGI variable names and associated values are written by the server to the CGIplus

© Copyright 2006 Hewlett-Packard Development Company, L.P

process and read as a succession of records from the CGIPLUSIN stream. An initial sentinel record prepares the script for a new request. After that, each record contains a name-value pair until the receipt of an empty record indicates end of request data — a very simple protocol. Based on the supplied request data, the script generates a response using the standard CGI schema, terminating it with an end-of-output sentinel record. The script then enters a wait state until the server provides another initial sentinel record at the beginning of the next script request. This cycle repeats itself until the script itself exits or until a configurable period passes without a fresh request being directed to the script process. When that period expires, the server runs down the script process. The process is simple, elegant, and very efficient!

CGIplus is a straightforward variation on standard CGI scripting that allows DCL implementation. The following example from the WASD package shows all the functional elements of the operations described, and provides a plain-text list of the CGI variables associated with the request, along with the number of times the script has been invoked.

```
$! Simple demonstration that even DCL procedures can be CGIplus scripts!
$! 08-JUN-1997  MGD  initial
$!
$ say = "write sys$output"
$ UsageCount = 0
$ FirstUsed = f$time()
$ open /read CgiPlusIn CGIPLUSIN
$!
$ RequestLoop:
$!
$!    (block waiting for request, this initial read is always discardable)
$     read CgiPlusIn /end=EndRequestLoop Line
$     UsageCount = UsageCount + 1
$!
$     say "Content-Type: text/plain"
$     say ""
$     say "Number of times used: ''UsageCount'"
$     say "First used: ''FirstUsed'"
$     say "Now: ''f$time()'"
$     say ""
$!
$!    (read and display the CGIplus variable stream)
$     CgiVarLoop:
$        read CgiPlusIn /end=EndCgiVarLoop Line
$        if Line .eqs. "" then goto EndCgiVarLoop
$        say Line
$        goto CgiVarLoop
$     EndCgiVarLoop:
$!
$     say f$trnlnm("CGIPLUSEOF")
$     goto RequestLoop
$!
$ EndRequestLoop:
```

A working example is available from http://wasd.vsm.com.au/cgiplus-bin/cgiplusproc. The same elements and structure can be seen in Case Study 1 (the telecommunications industry case study) and in Case Study 2 (implemented in GT.M (or MUMPS)).

Example code in C, Perl, and Java is provided with the WASD package, as well as a C library for transparently handling CGI and CGIplus scripts. A more efficient variant of the CGI variable transfer, struct mode, further improves CGIplus performance by up to another 2x!

CGIplus is the basic construct used by developers to implement persistent scripting environments and keep specific resources instantiated. The server also carefully controls script shutdown. Scripting environments need to be run down for various reasons, such as usage limits, processing errors, application shutdown, or resource exhaustion. The WASD server attempts to allow scripts to elegantly release instantiated resources by using the $FORCEX system service to invoke exit handlers. It subsequently uses $DELPRC to shut down any particularly recalcitrant scripts. This staged approach

© Copyright 2006 Hewlett-Packard Development Company, L.P

permits databases to be released without rollback and to meet similar rundown requirements to be met.

During 2000, the WASD CGI/CGIplus/RTE scripting infrastructure was revised significantly to allow the management of detached and non-server account processes. Using completely detached processes removes issues that are associated with pooled quotas because each process is a completely independent scripting entity. Detached processes also allow scripts to be run in their native account environment, if necessary. These facilities are available on all platforms and versions of OpenVMS from V6.0 through V8.2-1.

CASE STUDY 1: TELECOMMUNICATIONS INDUSTRY

Authors: Jeremy Begg and Ben Burke

This case study describes a WASD+SOAP installation at a large Australian telecommunications carrier that provides landline and cell phone services and Internet services. Due to Commercial in Confidence considerations, we cannot identify this customer.

Site Overview

The billing system for Postpaid Mobile business has been in continuous use for over a decade, providing integrated rating, billing, and online provisioning to network elements that make up the GSM network. Although originally built as an OpenVMS/VAX-only configuration, the billing system has been expanded to include other platforms, including OpenVMS/Alpha. The VAXes are now primarily being used to interface to the GSM network hardware. Recent changes and enhancements to the company's internal systems required a new data schema to encapsulate the entire profile of a customer's service and a way to distribute the data between the different operating systems and applications in use. This schema, the Service Profile, includes billing product information, discounts, promotions, and mobile features information.

To permit existing corporate systems and middleware to exchange this Service Profile information, we decided to use web services technology based on XML, SOAP 1.1, and HTTP. The web services operations for exchanging XML Service Profile information to and from OpenVMS was implemented in just four APIs:

- SetServiceProfile, a SOAP service for processing changes to the database & GSM network
- GetServiceProfile, an HTML form for retrieving a current Service Profile document for any mobile service
- GetProvisioningXML, a SOAP service by which certain non-OpenVMS systems retrieve information to perform network provisioning
- SetCompletionStatus, a SOAP service used by those systems to inform the billing system that the network provisioning has been completed (successfully or otherwise)

The APIs had to run on both OpenVMS/Alpha and OpenVMS/VAX. However, HP's Secure Web Server (based on Apache) was not available for VAX. Furthermore, all of HP's standard offerings in the web services area were based on Java – also not available on VAX. Therefore we built our own solution using available technology.

Selecting the Technology

First we needed a web server. WASD is a widely used web server available on both OpenVMS/VAX and OpenVMS/Alpha. The following features made it a good choice for us:

- The CGIplus architecture permits us to satisfy API requests without the overhead of per-request process creation. This was very important to us because these APIs are expected to process tens of thousands of transactions per day.
- The WASD implementation includes monitoring and troubleshooting capabilities. Because we standardized on a single Service Profile document schema, our APIs were complex. The WASD facilities enables us to debug during development and to monitor during implementation.
- WASD includes support for most commonly used web technologies (for example, SSL).

© Copyright 2006 Hewlett-Packard Development Company, L.P

Next, we needed an XML parser. The Gnome libxml2 toolkit was an open source XML parser and library written in ANSI C and available for both OpenVMS/VAX and OpenVMS/Alpha. We found that libxml2 is both rich in features and very efficient. It was not necessary evaluate any other XML parsers.

Writing a SOAP Processor

The SetServiceProfile API is the most complex of the four APIs. This API is implemented as a SOAP 1.1 POST request. SetServiceProfile reads a SOAP document, then calls libxml2 routines to parse the SOAP envelope header and to extract the ServiceProfile body. The header of the request contains various mandatory items such as:

- The phone number(s) being operated upon (there can be up to three—for speech, fax, and data)
- The transaction ID
- The date on which the change should take effect.

The body of the request contains the ServiceProfile document itself: a description of the service offerings that are enabled and disabled.

Textbooks on XML and SOAP suggest that, to process XML documents packaged inside SOAP envelopes, you need a fully web services-compliant environment with the ability to read Web Service Description Language (WSDL) documents, determine on-the-fly what a particular piece of XML is describing, and implement the business logic. However, we were able to implement SOAP processing simply by using the WASD web server and the libxml2 library.

Each of the APIs is coded following the standard CGIplus program structure:

```
Initialize
Loop
   Wait for request
   Process request
   Send response
Until (time to exit)
```

In our application, the "Process request" portion included ORACLE Rdb database activity that uses existing routines written in COBOL and BASIC. We coded the CGIplus programs in C, from which we make the calls to the COBOL and BASIC routines.

The other APIs were more simple than SetServiceProfile, and were therefore implemented using HTML constructs with parameters passed using GET. For example, GetServiceProfile accepts a phone number and historical date, looks up the database to determine the services subscribed to that phone on the requested date, and uses the libxml2 routines to assemble the Service Profile in XML format. The completed Service Profile is then returned to the requesting system.

WASD Features

WASD has some unique features that greatly facilitated both development and implementation, including process termination, proxy authentication, and script monitoring.

Process Termination

The script process does not terminate between requests; therefore, overall throughput and response times is very good. WASD provides mechanisms for gracefully shutting down inactive CGIplus processes. However, we implemented the following mechanisms for shutting down the program proactively:

- Limit the number of iterations of the main processing loop. This mitigates the possible effects of memory leaks, unreleased database locks, and other latent programming errors in the legacy database routines.
- Exit the program if an error is detected in the input XML or during database operations. This minimizes the disruption to service when a error detected in one request causes a later request to fail without warning.

During testing, we discovered a second form of process termination for which we have no real solution. When WASD creates a process to run the CGIplus program, SYS$INPUT and SYS$OUTPUT

© Copyright 2006 Hewlett-Packard Development Company, L.P

are mapped to VMS mailboxes that are used by WASD to communicate with the process. WASD discards all process output received before the first request is accepted by the process, which avoids the problem of miscellaneous output from SYS$SYLOGIN and LOGIN.COM. However, after the process has accepted a CGI request, it conforms strictly with CGI scripting protocols. The next output by the process after it has read the request parameters must be a CGI or HTTP response header. If it is not, WASD runs down the script and terminates the process. As a result, debugging is difficult because only the first line of output is recorded in the WASD process log file.

Proxy Authentication

The billing applications maintain audit trails that include the name of the OpenVMS user who changes the database. HTTP requests are performed on behalf of mobile phone resellers, and each reseller is mapped to a specific OpenVMS user account. Therefore, the CGIplus process calling application specifies the identity of the reseller in the request. (In this context, the term "reseller" refers to a chain of stores operating under a single brand, not an individual retail outlet.) To ensure that the CGIplus process was created in the appropriate OpenVMS account, authenticated HTTP is used to make the request, forcing the client to include an authorization header in the HTTP request. The WASD configuration runs the script OpenVMS user name specified in the HTTP authorization header.

WASD's Proxy Authentication mechanism maps HTTP-supplied credentials to OpenVMS user names to avoid having to maintain valid OpenVMS user name/passwords combinations on the remote system. Proxy Authentication uses the following configuration files:

- A plain text authentication file, which lists the remote user names and their corresponding passwords
- A proxy mapping file, which associates the HTTP-supplied user names to one or more OpenVMS user names

If the remote system's HTTP request supplies a valid username and password from the authentication file, WASD Proxy Authentication maps the request to the appropriate OpenVMS user account.

Script Monitoring

WATCH is an excellent WASD facility that shows the processing of requests in detail. The incoming request header and body are optionally viewable. The generated response (header and/or body) can be viewed. The associations and authorization rules, CGI variables, and so on, are available at run time and can be viewed using a web browser interface. WATCH makes it much simpler to debug web scripts.

WASD meets the requirements of this environment with quality monitoring tools, features, documentation, and support. It is unlikely that any other webserver, on any platform, would be as good a match in all these areas.

CASE STUDY 2: COAST CAPITAL SAVINGS

Author: Howard Taylor

Coast Capital Savings is a credit union servicing 300 000 customers in the Lower Mainland and southern Vancouver Island regions of British Columbia, Canada. Coast Capital Savings' banking system runs on OpenVMS AlphaServers and is written in Greystone Technology M (M). The banking system is natively accessed through a VT-terminal interface. In 2002 we (the Information Technology Group of Coast Capital Savings) were asked to deliver a GUI interface to the banking system. After evaluating various proposals, we decided to build a Windows .NET client application and interface.

From SOAP to M

To make it easier to reuse existing banking system routines and to ensure that the banking application's business rules were enforced, we chose to implement a remote procedure call (RPC) mechanism. Of the available RPC mechanisms, including CORBA, Microsoft DCOM, and JavaBeans, the one which seemed to be the most flexible for a cross-architecture scheme such as ours was the XML-based SOAP-RPC.

© Copyright 2006 Hewlett-Packard Development Company, L.P

We needed to take an XML SOAP-RPC request and transform it into a data structure that could be read by GT.M. To illustrate, here is an example of a SOAP-RPC procedure "GetTranCodes" with a single parameter "CustomerID" equal to "123456789":

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
 <env:Body>
  <p:GetTranCodes xmlns:p="http://coastcapital.local/ibsclient">
  <p:CustomerID>123456789</p:CustomerID>
  </p:GetTranCodes>
 </env:Body>
</env:Envelope>
```

This XML structure does not map directly to an M structure, so we mapped the procedure name and parameters into an M local array as:

```
LOCAL("ProcedureName")="GetTranCodes"
LOCAL("Body","GetTranCodes","CustomerID")=123456789
```

The XML namespaces and their prefixes wer mapped as:

```
LOCAL("Envelope","~xmlns","href")="http://www.w3.org/2002/06/soap-envelope"
LOCAL("Envelope","~xmlns","prefix")="env"
LOCAL("Body","GetTranCodes","~xmlns","href")="http://coastcapital.local/ibsclient
"
LOCAL("Body","GetTranCodes","~xmlns","prefix")="p"
```

We had already installed the Gnome libxml2 parser on our OpenVMS systems, so we built a C external routine to use the libxml2 API to parse SOAP-RPC XML documents into M local variables as shown above. M code then:

- Reads the local array.
- Accesses a table in the banking database to validate the RPC name and arguments.
- Invokes an M routine referenced by the RPC procedure name.
- Formats the results or errors into the response format specified in the SOAP 1.2 recommendation.

Integrating with WASD

We needed a robust, high-performance delivery mechanism for this new application. We were already using the WASD web server on our system for serving up BookReader documentation, so we investigated using WASD to deliver SOAP-RPC web services.

We were pleasantly surprised by the rich functionality we found, including:

- CGI scripts can be started from a DCL command procedure
- CGIplus avoids the penalty of OpenVMS image activation
- WASD can run detached CGIplus processes under different user accounts
- CGIplus processes can be gracefully shut down to permit dynamic software updates
- Load-balancing and throttling are built in

All this functionality came with excellent performance. A .NET single-stream test application we created to randomly exercise 47 of our commonly-used RPCs reported an average round-trip response of 128 ms (minimum 16 ms).

Programming WASD's CGIplus interface is very straightforward. This small amount of M code performs all of the interfacing our RPC server needs to do with WASD:

```
        S CGIIN="CGIPLUSIN"                 ; CGIplus standard input
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

```
        S CGIOUT="SYS$OUTPUT:"             ; CGIplus standard output
        O CGIOUT:(BLOCKSIZE=65535)         ; Open input
        O CGIIN:(READONLY:BLOCKSIZE=65535) ; Open output
        F  U CGIIN R LINE Q:$L(LINE)=0  D  ; Loop reading input until blank line
        . S KNAM=$P(LINE,"=",1)            ; Get key name of CGI variable
        . S KNAM=$E(KNAM,5,$L(KNAM))       ; remove "WWW_" prefix
        . S WWW(KNAM)=$P(LINE,"=",2)       ; Store value in local array WWW
EOF     ;
        I WWW("REQUEST_METHOD")="POST" D POSTIN ; If POST input, do POSTIN
        U CGIOUT W $ZTRNLNM("CGIPLUSEOF")  ; Send EOF (required by CGIplus)
        K (CGIOUT,CGIIN,NULLDEV)           ; Clean up
        Q
    ;
POSTIN  ; Read POST input
        ;
        S HTTPIN="HTTP$INPUT"               ; POST input stream
        O HTTPIN:(READONLY:BLOCKSIZE=32767) ; Open POST
        S POST=""
        U HTTPIN:EXCEPTION="G EXIT"
        ; Read contents of POST into a local variable
        F  Q:($L(POST)=WWW("CONTENT_LENGTH"))  R REC S POST=POST_REC
        C HTTPIN                            ; Close POST
        . . . (process the POST contents)
        Q
```

Using a .NET class to interface to the banking system's SOAP-RPC server, we were able to quickly build our Windows GUI client application in VB.NET. We were also able to reuse the .NET class in other client-server applications that needed to communicate with our banking system.

The WASD SOAP-RPC mechanism implemented at Coast Capital Savings today serves approximately 1500 interactive workstations, as well as a busy customer-facing IVR system, and many new applications are scheduled to use it in the near future. Overall we have been impressed by the ease of implementation, high performance, and minimal overhead of this excellent web server.

For more information

The WASD package is demonstrated and available at http://wasd.vsm.com.au/.

The libxml2 toolkit is described and available at http://www.xmlsoft.org/.

SOAP and its RPC are described at http://www.w3.org/TR/soap12-part2/.

Jeremy Begg may be contacted via the VSM site at http://www.vsm.com.au/.

© Copyright 2006 Hewlett-Packard Development Company, L.P

OpenVMS Technical Journal V7

Reusing OpenVMS Application Programs from Java

David J. Sullivan, Expert Member Technical Staff

Overview

This article introduces a new product developed by HP OpenVMS engineering. The Web Services Integration Toolkit (WSIT) for OpenVMS was designed to ease the burden of calling non-Java applications from Java applications. Non-Java applications are typically older, stable, and provide a significant business value. An example of a non-Java application might be a C or COBOL application written in the 1980s. Java applications are typically newer and leverage the latest technology. An example of a Java application might be a web service application called by Microsoft .NET. With WSIT, these two applications can be easily integrated while keeping the original application running in its current form.

The Web Services Integration Toolkit can be used to integrate application libraries written in programming languages such as C, BASIC, COBOL, and FORTRAN. It also has support for integrating ACMS applications.

The latest kit can be obtained from the website http://hp.com/products/openvms/wsit. The kit will be bundled with the next OpenVMS Alpha e-Business CD and the Foundation Operating Environment (FOE) on HP OpenVMS Integrity servers.

The Problem Addressed by WSIT

The popularity of the Java programming language has grown within the OpenVMS installed base because of its platform neutrality and ease of use. However, businesses using OpenVMS cannot rewrite all of their applications in Java -- they need to reuse the logic in their older non-Java applications. Unfortunately, Java was not designed to allow developers to call programs written in other languages. The exception here is C. Java does have some low level support for calling C programs, but it is very difficult and cumbersome to use. Simply put, writing the code to call legacy application from Java is difficult, time consuming, and error prone.

WSIT was designed to address this issue. It handles all of the difficulties of writing Java to some-other-

© Copyright 2006 Hewlett-Packard Development Company, L.P

language integration code. The toolkit is composed of small and simple tools. These tools can be extended and customized by the developer. Each tool is specific and obvious in it use. The developer using WSIT is able to wrap older application libraries and exposes them as a Java classes.

The Opportunity for Web Services and OpenVMS

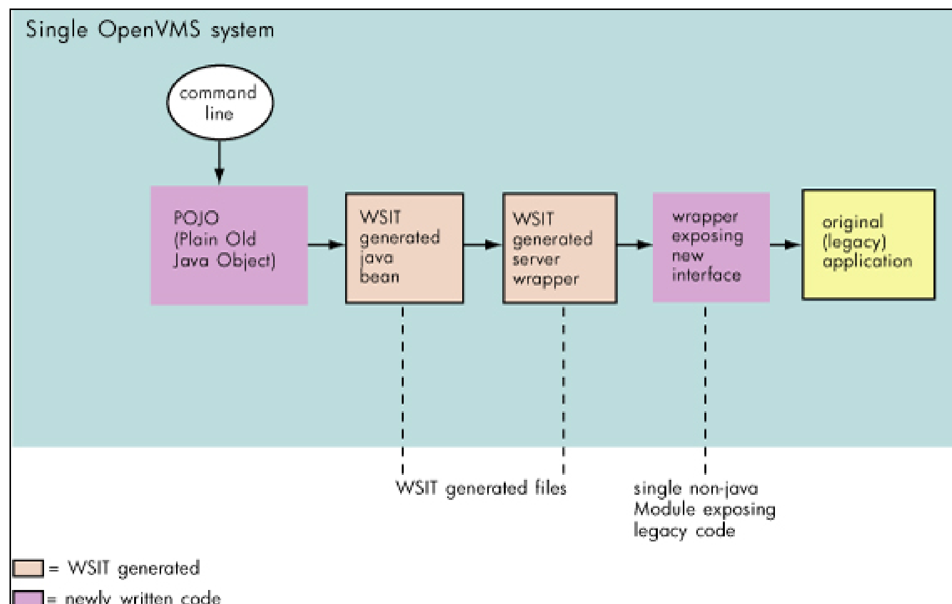
The toolkit generates Java classes that can be called from any Java technology. However, a popular use of these Java classes is likely to be from web services. Web services are the fastest growing integration technology. They are designed for language-neutral and operating-system-neutral application integration. OpenVMS applications are perfect candidates to benefit from web services.

Use Scenarios

As mentioned earlier, the Java classes generated by WSIT can be called from any Java technology on OpenVMS. Some of the most popular technologies for calling the WSIT Java classes are:

- POJO (Plain Old Java Object), perhaps accessed by a command line interface
- JSP (Java Server Page), accessed by a web browser
- Web service accessed by a web service client on any platform (for example Microsoft .NET)

In the following section we will take a high level look at each of these scenarios.



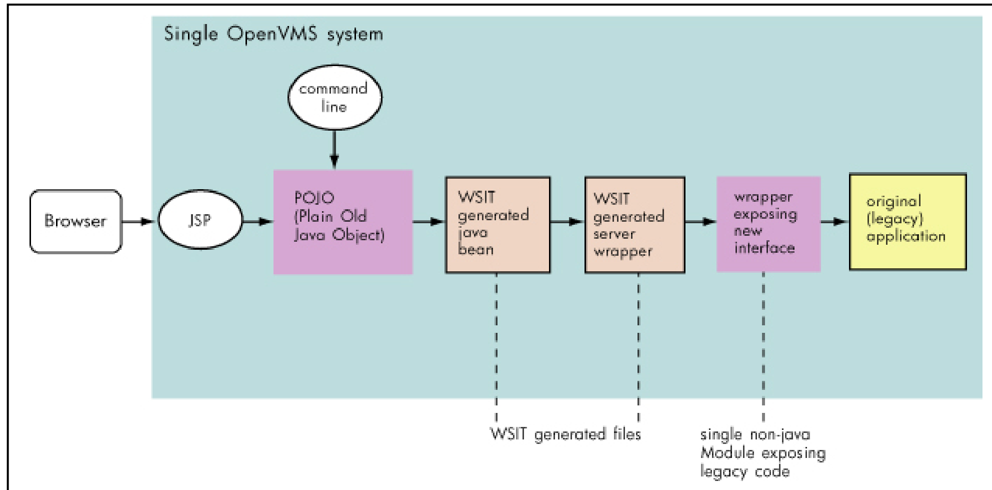
POJO (Plain Old Java Object) Called from the Command Line

This is the most basic scenario. It is a great way to get started using WSIT without having to worry about writing clients in other environments. The figure below illustrates a simple Java class calling another Java class which was generated by WSIT. The generated classes forward the clients calls to the non-Java application and return the result.

Note that all the code executes on a single OpenVMS system.

The POJO with command line processing can be written using any version of the Java Software Development Kit (SDK) for OpenVMS.

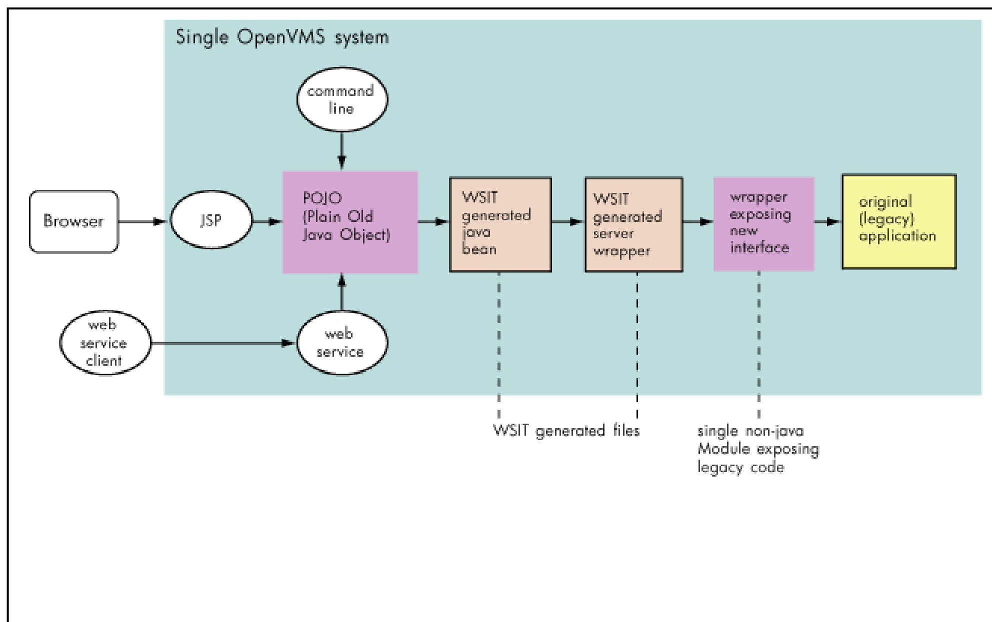
© Copyright 2006 Hewlett-Packard Development Company, L.P



Java Server Pages (JSPs) Called from Browser

This scenario illustrates how to add a web interface to an older non-Java OpenVMS application. A JSP on OpenVMS provides a web base interface to a browser. When the user clicks on the browser web page, it triggers the appropriate calls to the JSP. The JSP then forwards the calls to the Java class from the POJO scenario. Note that the POJO may still have the command line interface in addition to the web interface.

JSP are deployed using web servers. Web servers on OpenVMS include: Apache Tomcat, BEA WebLogic Server.



Web Services and Web Services clients (Microsoft .NET, J2EE, Java)

This scenario illustrates how to reuse the OpenVMS application library by exposing it as a web service that can be called from any operating system and language that supports web service clients. The web service client connects to the service on OpenVMS and calls a method. The service forwards the request to the POJO which returns the result of the methods call. This data is then returned to the web service client.

Web Service Engines on OpenVMS include: Apache Axis, BEA WebLogic Server.

Web Service Clients can be written in many different ways including: Microsoft .NET applications, Java JAX-RPC clients, J2EE application servers such as BEA WebLogic Server, most handheld software, and many more.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Using WSIT

Next we will take a look at the WSIT tools and see how they are used to wrap a non-Java application.

The Goal

For the purpose of illustration we will use a very simple application written in the C programming language. For future reference, more complex sample applications in various languages, as well as an ACMS application, are included in the WSIT kit.

The math application has two routines, *sum* and *product*. The interface is exposed in the file DISK$:[VTJ]math.c

```
$ set default DISK$:[VTJ]
$ ty math.c

unsigned int sum ( int number1, int number2) {
  return number1 + number2;
}

unsigned int product ( int number1, int number2) {
  return number1 * number2;
}
$
```

The purpose of the WSIT tools is to generate a Java class that presents a Java version of the math.C routines. The Java interface should look similar to the figure below.

```
public interface Imath {

    public int sum (int number1, int number2)
                            throws WsiException;

    public int product (int number1, int number2)
                            throws WsiException;
}
```

WSIT provides the following tools: The use of these tools will become clear as we look at the typical development steps. For now just know that these tools exist.

OBJ2IDL.EXE (I64 only) takes an OpenVMS object file (.obj) as input and generates an XML description of the applications interface. Referred to in this article as an XML IDL file.

© Copyright 2006 Hewlett-Packard Development Company, L.P

STDL2IDL.JAR the ACMS equivalent to obj2idl.exe. It takes an ACMS STDL description as input and generates an XML description of the applications interface. Referred to in this article as an XML IDL file.

VALIDATE.JAR takes an XML IDL file and validates the XML against the OpenVMS IDL schema.

IDL2CODE.JAR takes an XML IDL file and generates a Java class with the same interface. The Java class knows how to call the original non-Java application.

Typical Development Steps

Step 1: Prepare the application. [Not required but highly encouraged]

Step 2: Describe the interface with an XML Interface Definition Language (IDL) file.

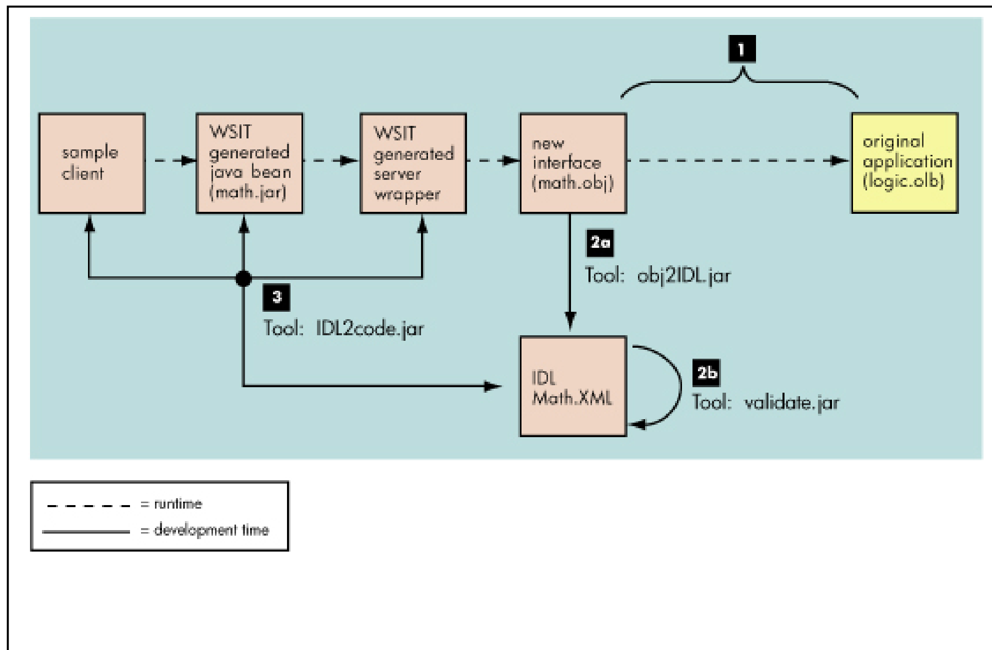
- Tools: **OBJ2IDL.EXE (for C,BASIC,COBOL,FORTRAN), STDL2IDL.JAR (for ACMS), VALIDATE.JAR**

Step 3: Generate Java wrapper based on XML IDL file from step 2.

- Tools: **IDL2CODE.JAR**

Step 4: Test the generated code from a client.

- Tools: **IDL2CODE.JAR** (WSIT version 1.1 or higher)

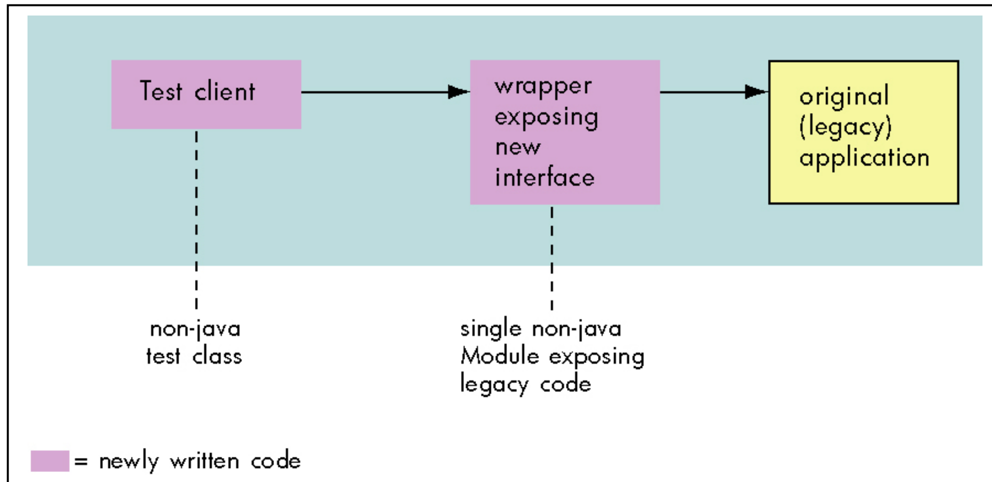


Step 1: Prepare the application [not required but highly encouraged]

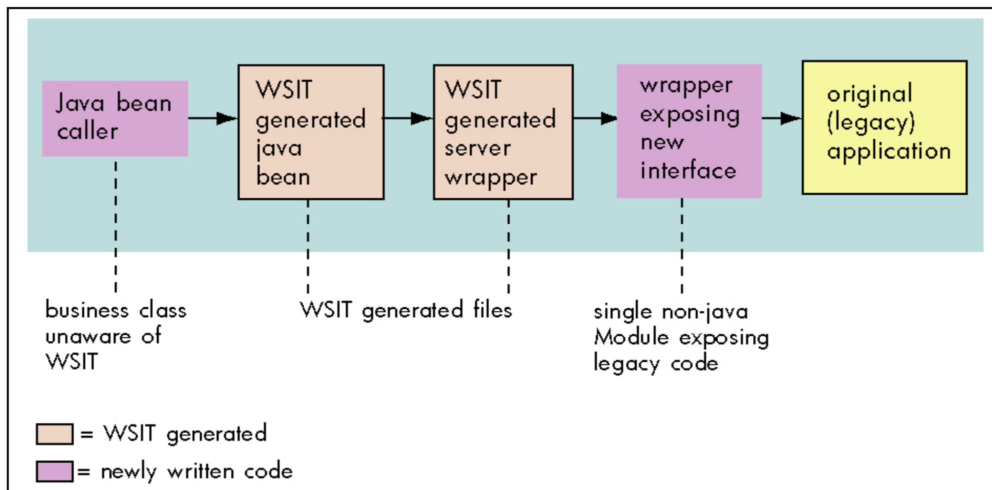
Before using any integration technology, you should evaluate the original application. The application is likely to have been written long ago and will benefit from having a wrapper expose a new and clean interface. The new interface will expose the legacy implementation. Separating the interface from the implementation provides encapsulation and the ability to easily extend and reuse the implementation.

© Copyright 2006 Hewlett-Packard Development Company, L.P

To avoid complexity, these new interfaces should be tested with a simple client before you use the Web Services Integration Toolkit. When you know that the interface classes are working properly, you can use WSIT to extend the use of the new interface to the Java environment.



After you have prepared the application, WSIT can extend the features of the new interface to Java, as shown in the following diagram.



Step 2: Describe the interface

The WSIT development tools generate and consume a simple XML file that describes an application's interface.

You create an XML IDL file using the tool named **OBJ2IDL.EXE** (for 3GL languages) or **STDL2IDL.JAR** (for ACMS).

Note: OBJ2IDL.EXE runs on OpenVMS I64 only. If you are using WSIT on OpenVMS Alpha, you can create the XML IDL file manually in any editor using the many samples as a guide. If you have both OpenVMS I64 and Alpha systems, run OBJ2IDL.EXE on I64 and copy the resulting XML IDL file to your Alpha system.

To generate an XML IDL File for the math.c application interface, use the command line below. The obj2idl executable accepts a switch ***–f*** which specifies the name of the object file from which to extract the interface definition. The object file must be compiled with /debug /noopt.

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
!Establish WSIT logicals
$ @wsi$root:[tools]wsi-setenv - wsi$dev

!Establish a foreign command for the obj2idl tool:
$ obj2idl = "$WSI$ROOT:[tools]obj2idl.exe"

!Compile the wrapper that exposes the new interface:
$ set default DISK$:[VTJ]
$ cc/debug/noopt math.c

!Use obj2idl to generate an xml file with the interface
definition:
$ obj2idl -f math.obj
```

The XML file generated by the obj2idl tool is shown in the appendix. Even those unfamiliar with XML should be able to understand what this file is doing.

Whenever you use the obj2idl tool you must verify that the XML IDL file correctly describes the interface being exposed. If it does not, manually update the XML IDL file until the interface definition is correct.

If you have modified the XML file you can ensure that it is still well-formed and valid by using the tool **VALIDATE.JAR**. For those unfamiliar with XML, an XML file is considered well formed if it is syntactically correct. The file is considered valid if it is semantically correct. The XML rules for validity are defined in the file openvms-integration.xsd.

The validate tool is an executable jar file that accepts the following arguments:

-x DISK$:[VTJ]math.xml is a switch that specifies the name of the XML file to be validated.

-***s wsi$root:[tools]openvms-integration.xsd*** is a switch that specifies the XML schema file defining the semantic rules for validating the XML file specified with the switch -x.

```
$ java -jar wsi$root:[tools]validate.jar -x
DISK$:[VTJ]math.xml -s wsi$root:[tools]openvms-
integration.xsd
XML file validated sucessfully
$
```

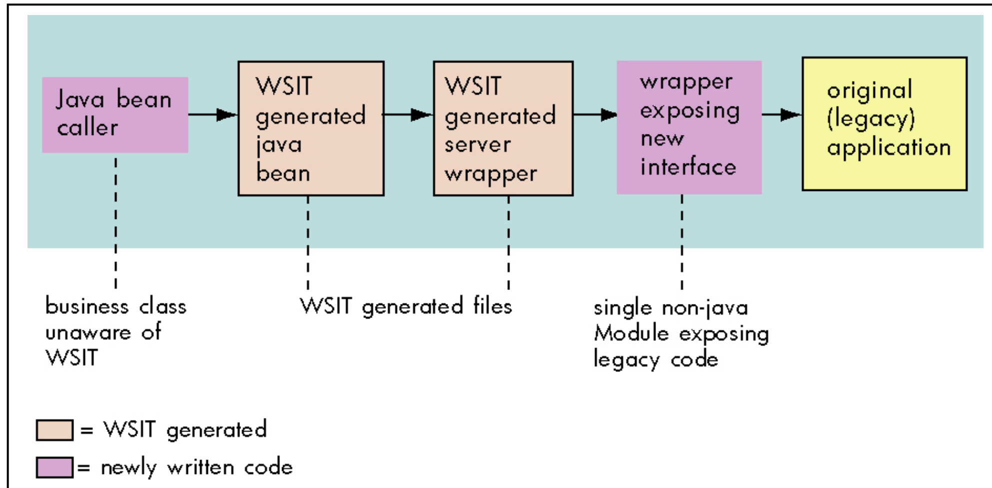

© Copyright 2006 Hewlett-Packard Development Company, L.P

Step 3: Generate code

The **IDL2CODE.JAR** tool generates the necessary code to wrap the routines described in the XML IDL file. There are two components generated:

1. One WSIT server wrapper: this code knows how to call the routines in math.obj.

2. One WSIT JavaBean: This code is the Java version of the routines in the math application. It also knows how to call the server wrapper.



The figure below illustrates how to use the **idl2code** tool. The main routine in the tool is named **com.hp.wsi.Generator**. The arguments are as follows:

-i math.xml is a required switch that specifies the name of the xml file that describes the interface being wrapped.

-a math is a required switch that specifies the name to be used for the generated files.

-c SJ is an optional switch that specifies that one or more sample clients should be generate. The argument S will generate a command line based interface client. The argument J will generate a JSP web based interface client. We will look at the clients in more detail in later. Note that this switch was added in WSIT version V1.1. If you are using an earlier version, the switch will be ignored.

-o [.generated] is an optional switch that specifies a root directory where the generated files should be placed.

Note: the command below illustrates a new switch added in WSIT Version 1.1. If you are using an earlier version of WSIT the switch ***–c SJ*** *will be ignored in the command line.*

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
$ set default DISK$:[VTJ]
$ java "com.hp.wsi.Generator" -i math.xml  -a math -c SJ -o
[.generated]
File: ./generated/mathServer/build-math-server.com generated.
File: ./generated/mathServer/methIds.h generated.
File: ./generated/mathServer/structkeys.h generated.
File: ./generated/mathServer/math.wsi generated.
File: ./generated/mathServer/math.opt generated.
File: ./generated/mathServer/math-server.h generated.
File: ./generated/mathServer/math-server.c generated.
File: ./generated/math/build-math-jb.com generated.
File: ./generated/math/Imath.java generated.
File: ./generated/math/mathImpl.java generated.
File: ./generated/mathSamples/POJO/mathMain.java generated.
File: ./generated/mathSamples/POJO/build-math-PoJoClient.com
generated.
File: ./generated/mathSamples/JSP/index.html generated.
File: ./generated/mathSamples/JSP/mathMethodList.html generated.
File: ./generated/mathSamples/JSP/mathPopulate.jsp generated.
File: ./generated/mathSamples/JSP/mathDoCall.jsp generated.
File: ./generated/mathSamples/JSP/build-math-JspClient.com
generated.*** Application math generated! ***
$
```

Build the generated server wrapper

The Server build procedure creates an executable named math.exe. This file was linked with math.obj. Then math.exe is automatically copied to the WSIT deployment directory wsi$root:[deploy]

```
$ set default DISK$:[VTJ.generated.mathServer]
$ @BUILD-MATH-SERVER
Begin server build procedure.
  ..configuring switches and compiler options
  ..compiling native server code
  ..linking shareable image
  ..installing server image
End server build procedure.
$
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

Build the generated JavaBean

The JavaBean build procedure creates a JAR file that contains the WSI Java classes used to call the server wrapper generated earlier.

```
$ set default DISK$:[VTJ.generated.math]
$ @BUILD-MATH-JB
Begin java bean build procedure.
The New JAVA$CLASSPATH is:
   "JAVA$CLASSPATH" = "WSI$ROOT:[LIB]WSIRTL.JAR" (LNM$JOB_86D5AE00)
        = "[]"
        = "WSI$ROOT:[LIB]VELOCITY-DEP-1_4.JAR"
        = "WSI$ROOT:[TOOLS]IDL2CODE.JAR"
  ..Compiling structure classes
  ..Compiling math Interface classes
  ..Creating math.JAR file from classes
End of JavaBean build procedure.
$
```

Congratulations! You have used WSIT to generate a wrapper for the math applications routines. The generated Java wrapper is packaged in DISK$:[VTJ.generated.math]math.jar

Step 4: Test the generated code from a client

In the previous step the switch **–c SJ** was used to tell the WSIT generator to generate a sample client with a command line interface (**S**) and to also generate a sample client with a JSP interface (**J**). These samples are provided for your convenience. They are intended to ease testing and development when using WSIT.

Using the generated POJO client sample

Normally this class will be written to integrate the WSIT generated JavaBean with the Java technology of your choice.

The sample must first be built as illustrated in the figure below.

```
$ @wsi$root:[tools]wsi-setenv - DISK$:[VTJ.generated.math]math.jar
$ set default DISK$:[VTJ.generated.mathSamples.POJO]
$ @build-math-PoJoClient
Begin client build procedure.
  ..Compiling mathMain client class
End of client build procedure.

To run this client, type the following at the command line:
     $ java "math.mathMain"
$
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

The sample client is able to make calls to the methods of the generated JavaBean. There is a limitation that only methods with primitive arguments can be called. To see which methods the sample client can call use the switch **–m** as in the figure below.

```
$ set default DISK$:[VTJ.generated.mathSamples.POJO]
$ java math.mathMain -m
 The list of available methods within math are:


    sum(int P1, int P2)
    product(int P1, int P2)


(Methods that take structures or arrays as parameters are not callable
from this command line interface.  These methods are denoted by the *
next to them.)
$
```

To call the *sum* and *product* methods with arguments of 5 and 2 use the commands below.

```
$ set default DISK$:[VTJ.generated.mathSamples.POJO]
$ java math.mathMain -m sum -p1 5 -p2 2
Calling mathImpl.sum:
 P1 = 5
 P2 = 2
 Return value = 7
 ++ The client was successful ++

$ java math.mathMain -m product -p1 5 -p2 2
Calling mathImpl.product:
 P1 = 5
 P2 = 2
 Return value = 10
 ++ The client was successful ++
$
```

Using the generated JSP client sample

As with the POJO sample, the JSP sample client must first be built as illustrated in the figure below.

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
$ @wsi$root:[tools]wsi-setenv - DISK$:[VTJ.generated.math]math.jar
$ set default DISK$:[VTJ.generated.mathSamples.JSP]
$ @build-math-JSPClient
Begin JSP client build procedure.
Unpacking static files into current location.
Copying math.jar into local [.WEB-INF.lib] directory
Creating mathJsp.war file
End of JSP client build procedure.

To deploy this client:
        Copy mathJsp.War into the deployment
        directory for your JSP server.
$
```

To deploy the JSP, copy the mathJsp.War file to a web server servlet deployment directory. For example, if you have Tomcat on OpenVMS, the command may look like this:

```
$ copy mathJSP.war sys$common:[apache.jakarta.tomcat.webapps]
```

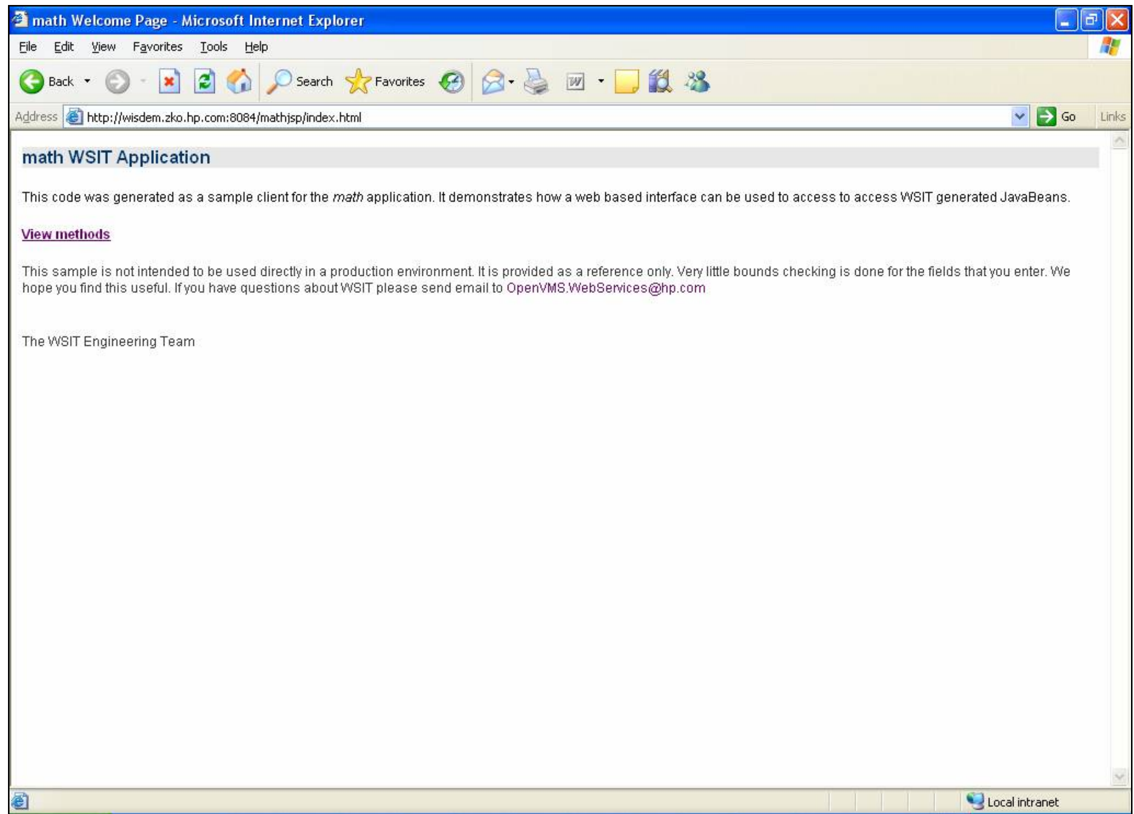
Once the war file has been copied, you can view the JSP pages by using a URL similar to the one shown below. (Replace yourwebserver.hp.com with the actual name of your web server.) By default, Tomcat listens on port 8080. If the system manger changed the port number, replace 8080 with the new number.

```
http://yourwebserver.hp.com:8080/mathjsp/index.html
```

The following screen captures illustrate the JSP sample client calling the C Math application.

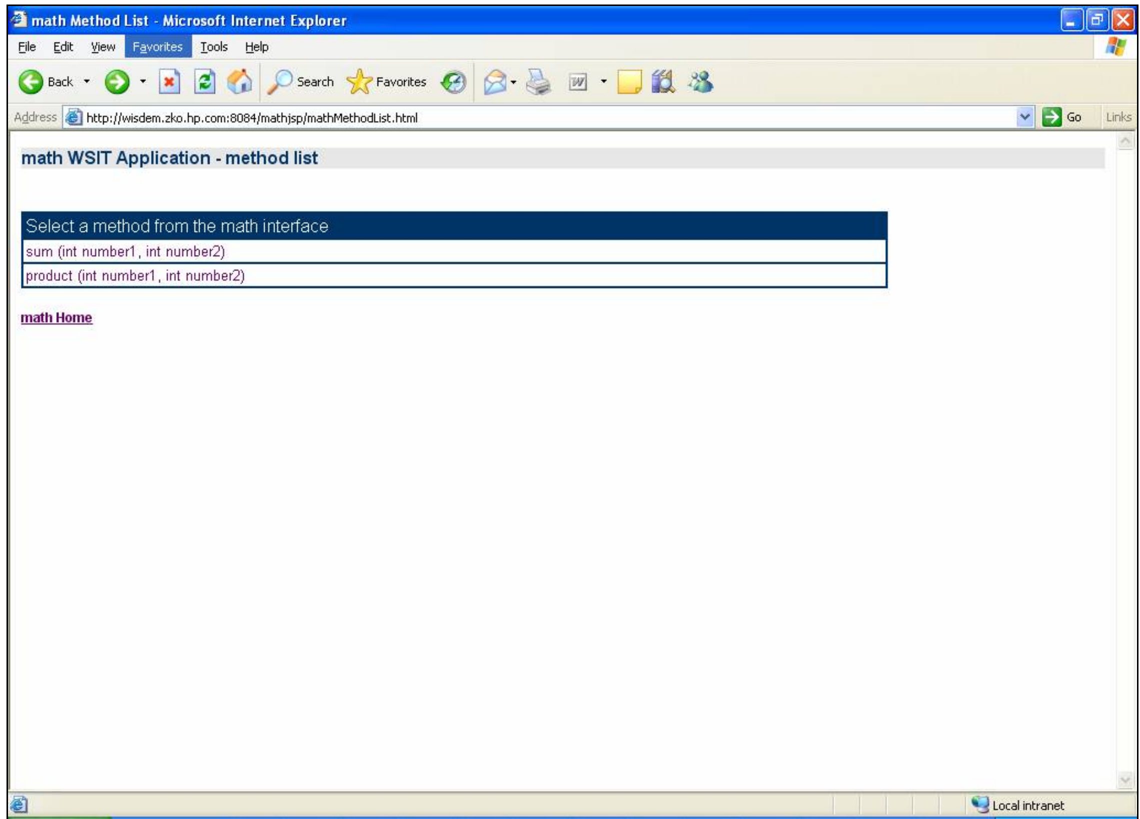
© Copyright 2006 Hewlett-Packard Development Company, L.P

Web Page 1: The mathJSP Application Homepage

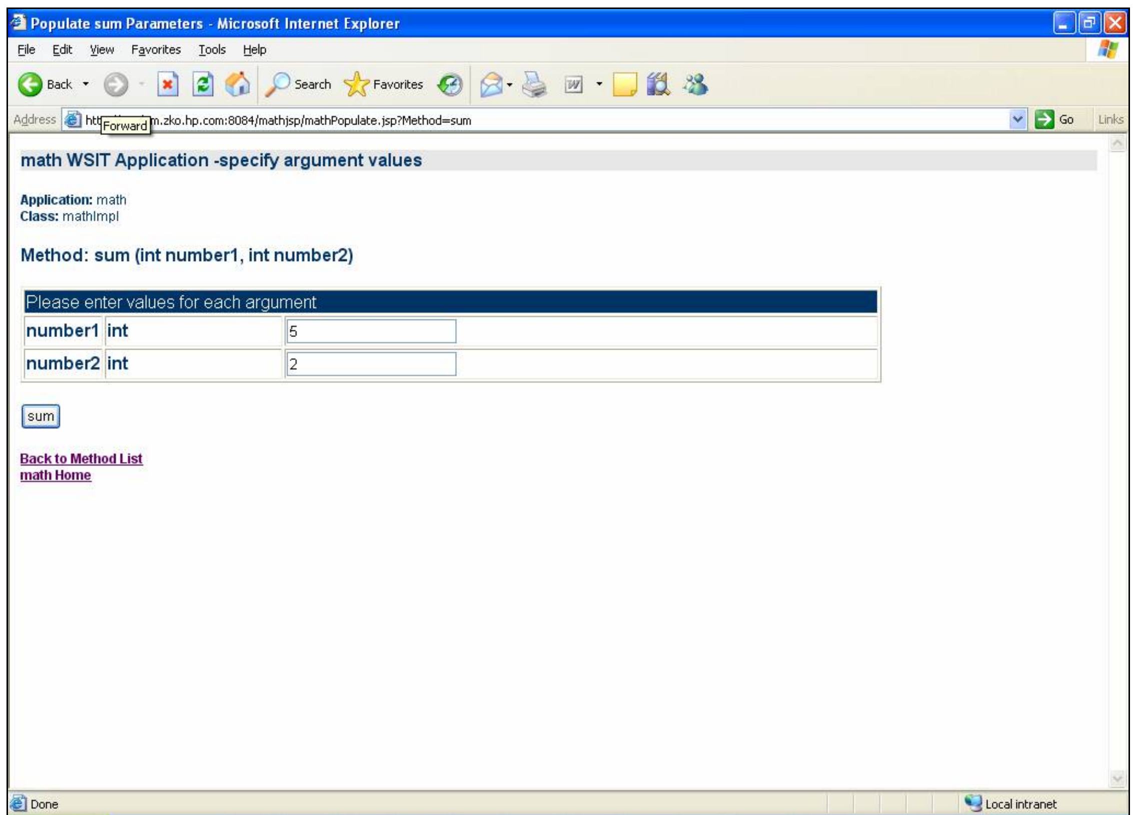


© Copyright 2006 Hewlett-Packard Development Company, L.P

Web Page 2: The MathJSP Application Methods

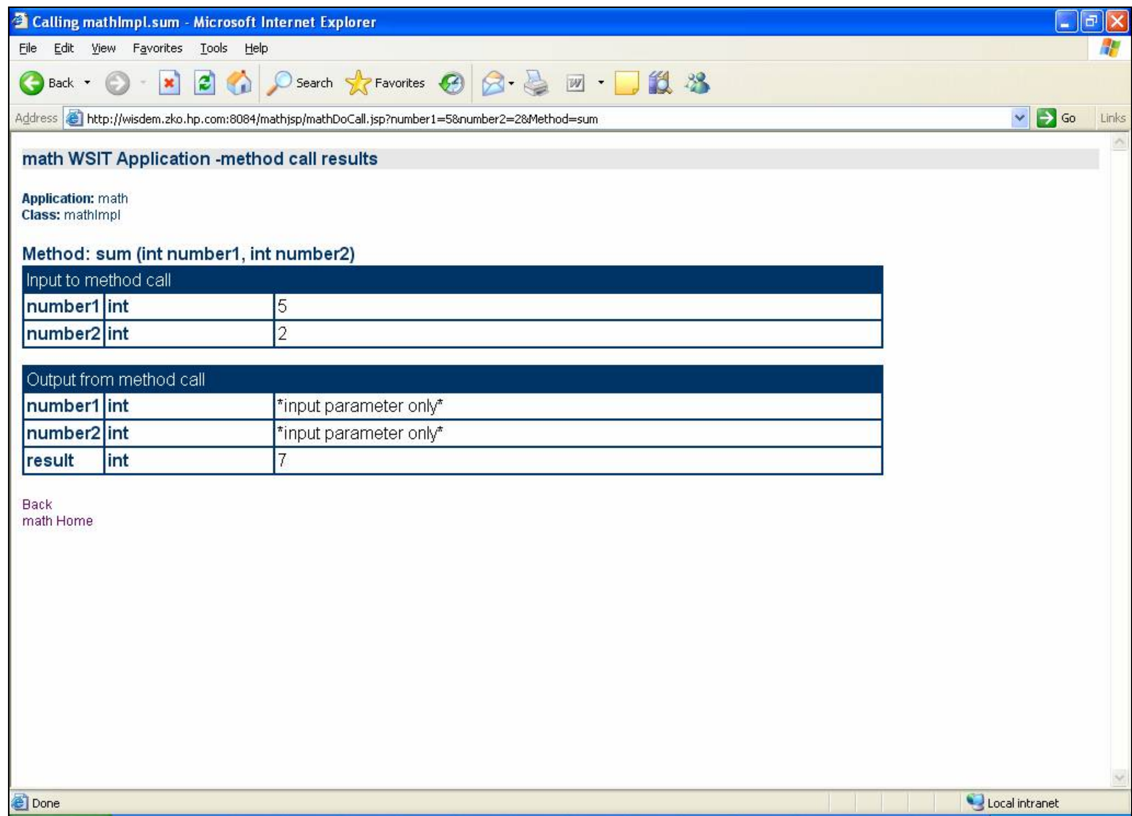


Web Page 3: The MathJSP Application Method *sum*



© Copyright 2006 Hewlett-Packard Development Company, L.P

Web Page 4: The MathJSP Application Method *sum* results



Deploying the Application Inproc / Outproc

There are two ways in which you can deploy your application using WSIT: **in-process** deployment and **out-of-process** deployment.

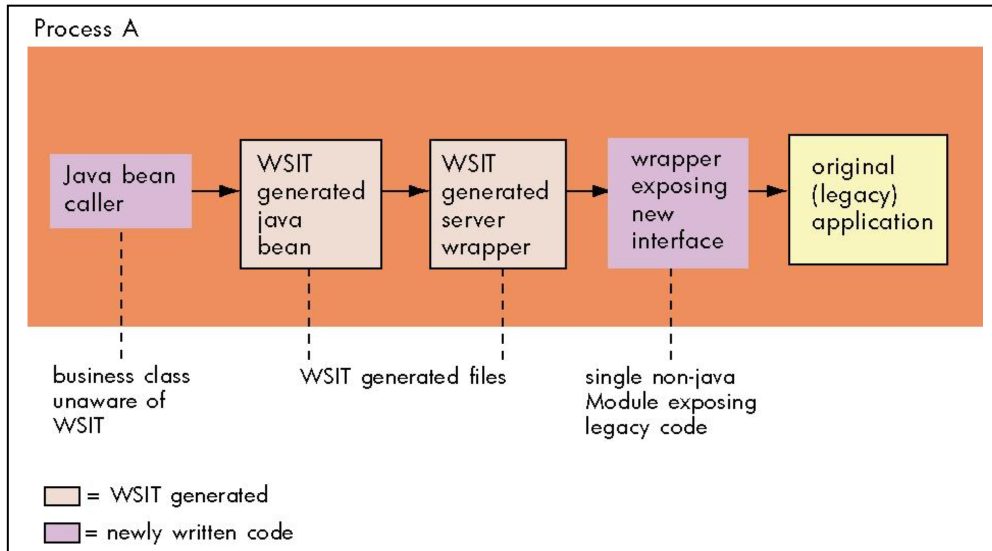
In the section Use Scenarios, we looked at a few of the different types of clients which can be used to access the WSIT wrapped application. These clients may interact with the application in different ways and may dictate the application's deployment settings.

For example, a JSP client will be deployed in a web server. The web server may accept multiple concurrent browser requests and execute each request on a separate thread. Many older OpenVMS applications were written with the assumption that they would never have more than one client using its services at a single time. In this scenario each client would need a private copy of the application to ensure that other clients do not interfere with its work. This can be accomplished by using out-of-process deployment.

In-Process Deployment

In-process deployment occurs when the application and the client are called from the same process, as illustrated in the following diagram.

© Copyright 2006 Hewlett-Packard Development Company, L.P



There are advantages and disadvantages to using in-process deployment.

Pros: Fastest execution time. No overhead added by the WSIT runtime.

Cons: A crash will bring down both client and server applications.

In-process deployment provides the fastest execution time, but it requires that the developer ensure that the client calling the wrapped application does not establish an environment in which the application will fail.

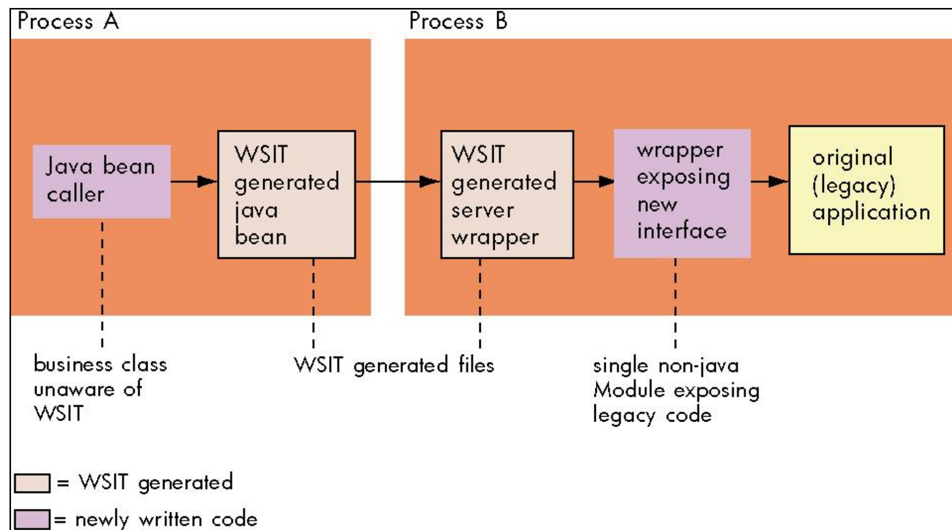
If you do not specify out-of-process deployment settings (described in the following sections), **your application will run in-process by default.**

Out-of-Process Deployment

As mentioned in Use Scenarios, the WSIT wrapper can be called from a variety of different technologies. Sometimes it may be desirable for the original application to run in a separate process than the calling technology. For example, a web server may allow multiple requests to be sent to the wrapped application. Each request may be running on a separate thread. If the original application was not designed to run in a multithreaded environment you can ensure that each request has its own private instance of the original application by running it out-of-process.

Out-of-process deployment occurs when the client and application are run in different processes, as illustrated in the following diagram. The WSIT runtime environment manages the interaction between the two processes. You can customize this environment by modifying a deployment descriptor file.

© Copyright 2006 Hewlett-Packard Development Company, L.P



There are advantages and disadvantages to using out-of-process deployment.

Pros: Typically scales better than in-process deployments. Allows the use of the WSIT runtime deployment properties.

Cons: Adds complexity and overhead to every call.

Most older applications benefit from using an out-of-process deployment to avoid complex issues that result from mixing older and newer environments.

Migrating from Other Products

BridgeWorks users have the ability to migrate their applications to WSIT. A tool is provided that generates a WSIT XML IDL file from a BridgeWorks connection description. This tool is named BWX2IDL and can be downloaded from the WSIT download site. See http://hp.com/products/openvms/wsit for more information.

Summary

The Web Services Integration Toolkit (WSIT) focuses on providing Java wrappers for non-Java applications. The Java wrappers can be used from a wide rage of technologies to provide both local and remote access.

Getting started with WSIT is easy. Simply pick one of the many WSIT samples and use the tools provided. In a few steps you can be calling a program written in C, BASIC, COBOl, FORTRAN or ACMS.

For more information

For more information about integrating OpenVMS applications, contact the author.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Appendix A: Contents of math.xml File Generated from obj2idl Tool

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenVMSInterface
       xmlns="hp/openvms/integration"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="hp/openvms/integration openvms-integration.xsd"
       ModuleName="DISK$:[VTJ]math.OBJ"
       Language="C89">
       <Primitives>
         <Primitive Name = "unsigned int"
                    Size = "4"
                    VMSDataType = "DSC$K_DTYPE_LU"/>
         <Primitive Name = "signed int"
                    Size = "4"
                    VMSDataType = "DSC$K_DTYPE_L"/>
       </Primitives>
       <Routines>
         <Routine  Name = "sum"
                   ReturnType = "unsigned int">
                   <Parameter Name = "number1"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
                   <Parameter Name = "number2"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
         </Routine>
         <Routine  Name = "product"
                   ReturnType = "unsigned int">
                   <Parameter Name = "number1"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
                   <Parameter Name = "number2"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
         </Routine>
       </Routines>
</OpenVMSInterface>
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

OpenVMS Technical Journal V7

Preliminary OLTP Performance Comparisons of Oracle Rdb V7.2 on OpenVMS I64 and Alpha

Author: Norman Lastovica, Oracle Rdb Engineering

Overview

Moving from the familiar environment of OpenVMS on Alpha and VAX systems to the world of OpenVMS running on Integrity Servers allows us to evaluate the performance and capabilities of another computer architecture and the systems built on it. Oracle is at the initial stages of optimizations of Oracle Rdb on the OpenVMS I64 platform, and we have performed preliminary performance tests comparing Alpha and Integrity servers.

This article provides some background information about the Oracle Rdb port to the OpenVMS I64 platform, and presents some observations based on the early performance tests performed with the HP OpenVMS engineering team during April of 2005.

To date, the results of the OLTP-oriented commercial workload tests indicate that system performance of current Integrity servers is at least as good as performance on the corresponding Alpha servers.

Run-time generated query-specific executable code

Rdb has always generated executable code at run time. This code is comprised of subroutines that are specific to the queries, fields, tables, data conversions that are necessary for database access. The performance of this code is a significant factor in the general database system performance characteristics of Oracle Rdb.

On the VAX architecture, VAX instructions are built into subroutines. Once the subroutine is complete, an REI instruction is executed, to make sure that:

- The CPU instruction caches are prepared for a change in the instruction stream
- The subroutine is available to be called

To move from the VAX architecture to the Alpha architecture, a new code generator in Rdb produces Alpha instructions, which are created into subroutines. Once this task is complete, an IMB (instruction memory barrier) causes the instruction cache to be invalidated before a new subroutine is called.

When porting Oracle Rdb to run on the Windows/NT operating system on the Intel I86 architecture, we took a slightly different approach. Rather than generating the I86 instruction set, a “pseudo code”

© Copyright 2006 Hewlett-Packard Development Company, L.P

instruction set was created. These instructions are executed with a run-time interpretation software engine rather than the processor chip itself. This permitted rapid development and specialized high performance interpretation of potentially complex instructions. For example, rather than having to generate code “long hand” to perform a sorting operation of a vector of longwords, a single pseudo instruction indicates that a sort is required. The run-time interpreter calls a general sort subroutine. This results in improved performance and significantly reduced maintainability requirements.

Performance of the instruction interpreter is always a concern, but general product performance is at least as good as it is on the Alpha with Rdb V7.1. Therefore, minor performance optimizations can be performed as specific issues are identified. The high level of performance of Rdb over a wide variety of applications continues to be a primary goal.

I/O performance

For any database engine, the performance of disk I/O operations (both reading and writing) is of special concern. The entire I/O path (from $QIO or $IO_PERFORM though the operating system, drivers, I/O adaptors and controllers and so on) must be fast, reliable, and efficient.

Impact of Alignment Faults

Alignment faults can impact performance. For operations that occur infrequently, an occasional alignment fault does not cause any noticeable impact to performance. However, alignment faults that occur during an inner loop (starting or stopping a transaction, or fetching rows from the database) may cause measurable performance impacts. Furthermore, an alignment fault on the I64 system can be up to an order of magnitude slower than on the Alpha, due to the complexities of the I64 architecture. We minimized alignment faults on I64 (and, obviously, on Alpha as well).

Initial test system configuration

In order to make an equitable comparison, we configured a two-node clustered rx4640 I64 system, and an ES45 Alpha system, sharing a common SAN and EVA5000 storage controller. OpenVMS V8.2 ran on both machines, along with internal Oracle Rdb T7.2 field test build (approximately equal to the T7.2-030 field test kit). A simple application simulator executed simple transactions by updating a random record from a random database table. All indexes were cached in physical memory using the Oracle Rdb Row Cache feature.

The database consisted of 500,000,000 rows, and each row consisted of 32 quadword (BIGINT) columns of random data. Including after-image journals, the database was approximately 50,000,000 disk blocks.

Database population

The first test measured the relative performance of populating the database itself. Each table is loaded by a program that generates random row content and stores a row. 50,000 rows per transaction were stored in the database. Four copies of the loader program were run simultaneously.

Numerous tests were run during the initial configuration stage to make sure that the system was correctly set up and that all components were working as expected. Back-to-back performance tests showed that the ES45 consistently loads about 49,500 records per second, while the rx4640 loads about 71,000 records per second — a very positive sign that things are looking great

After the unexpectedly good performance of the database load process, we did not perform further analysis. Later analysis and testing revealed a bug in the Alpha code and a bug in the I64 code, which explains the difference in performance during the data loading stage. Both problems have been corrected.

Initial OLTP workload runs

Several preliminary test runs on both architectures revealed the typical application performance.

Comparisons between the two systems showed that the Alpha system consistently provides about 20% better performance than the IA64 system, which is surprising because our previous tests led us to expect some parity between the two configurations.

The difference in performance was due to a large difference in CPU consumption between the two machines. The I64 system had 100% CPU utilization while the Alpha system was running at less than

© Copyright 2006 Hewlett-Packard Development Company, L.P

25% utilization. The database AIJ Log Server was the trigger to this problem. When the AIJ Log Server was disabled, the systems returned to roughly the behavior that we expected (the I64 system was no longer CPU-bound).

We suspect a synchronization problem between database users and the AIJ Log Server specific to internal differences in internal threading package within Rdb. We chose to disable the AIJ Log Server on both platforms for the remainder of the performance test and to analyze and correct the problem in the Rdb engineering labs at Oracle later; an I64-specific bug was found and corrected in the Rdb code.

Tools used to find performance problems

We used several of the tools available as SDA extensions to examination the CPU usage on the two systems more closely, including FLT and PRF.

FLT

The FLT tool showed that a number of alignment faults were generated within the Rdb database engine. Several BLISS macros at the root of the problem were modified to explicitly identify data that was not naturally aligned in memory, allowing the compiler to produce code that deals with the unaligned data better and avoids the alignment faults. Because our time was limited, we were not able to correct all of the faulting locations in the source code, but we addressed many of them during and after the testing.

PRF

The PRF tool analyzes the behavior of the running program. Because the Rdb database engine runs primarily in executive mode, we specified PC sampling of only those instructions that were executed in executive mode. At first we were astonished to discover that 20% of the CPU samples was used by three lines of source code in one module of Rdb. We repeatedly collected samples because it was so surprising, but the answer remained the same.

We found that, as each transaction was committed, a particular internal list was being scanned to clean up the transaction. For some classes of applications (including this one), the list itself became very large and scanning it was expensive. We replaced this list scanning method a different technique: a single-linked list was used to indicate only those blocks that were accessed during each particular transaction.

Our time on the systems was limited, so rather than attempting to make algorithmic changes in unfamiliar code in the Rdb database engine, we modified the test workload program to use dynamic SQL rather than pre-compiled statements. Dynamic SQL compiles a single outstanding request at a time, from Rdb's point of view. Contrast this with the hundreds of pre-compiled requests used by the original method. The goal was to reduce the size of the internal list of requests that was being scanned at each transaction commit.

Changing the program to use just a single dynamic request at a time changes the test profile considerably. Perprocess memory usage is reduced and user-mode CPU use is increased. (Both of these results were predicted and expected. Less memory is required because a large number of the request data structures need not be stored in memory, and additional CPU time is used to compile the dynamic request for each transaction.)

We ran the FLT tool again and the system executed many thousands of alignment faults per second. The faults occurred in the code that parsed the query BLR as database update statements were compiled in the database engine. These faults as well were corrected by modifying a few BLISS macros. We rebuilt and reinstalled an Rdb kit and most of the run-time alignment faults were eliminated on both platforms.

In equal circumstances, we anticipate that reducing the alignment faults benefits the I64 system somewhat more than the Alpha system, because alignment faults on the I64 platform are more expensive than on the Alpha. (Obviously they are expensive on Alpha as well.)

We then used the PRF tool to identify other hot spots in the OpenVMS executive code, as well as in SQL and the Rdb executive. With the workload running in a steady state, we collected CPU cycle

© Copyright 2006 Hewlett-Packard Development Company, L.P

samples in Kernel, Executive, and User processor modes. User-mode samples represented code executing in the simulator program, and in the SQL-generated code and sharable images. Executive-mode CPU samples represent RMS and Rdb; in our test case, all samples were from within the RDMSHRP72 image. Finally, Kernel-mode execution is from within the VMS executive itself.

The resulting CPU samples for each of the processor modes surprised us. A significant percentage of the CPU tables fell into a very few modules and, in several cases, individual lines of code.

Improvements Based on Analysis

The collected data and analysis indicated additional attention for several areas. Rdb engineering made some algorithmic changes in transaction commit processing and SQL statement processing, as well as reducing alignment faults. Simultaneously, OpenVMS engineering rapidly prototyped several enhancements to avoid CPU consumption in several key areas. The prototype changes in Rdb and OpenVMS were carried forward into production release of both products.

The changes made by both HP and Oracle resulted improved Rdb performance improvement and OpenVMS performance. Application performance improved dramatically (on the order of 20%) on both the Alpha and the IA64 system. Most significantly, however, the transaction rate on the rx4640 and ES45 test systems was effectively identical (1,701 as opposed to 1,726 transactions per second).

The performance improvements made in Rdb V7.2 increase throughput and improve response time on Alpha systems as compared to Rdb V7.1. These increases are due to algorithmic improvements, code optimizations, and alignment fault avoidance.

Comparing Rdb V7.2 with Rdb V7.1 on Alpha

A requirement of Oracle Rdb V7.2 is that it must be at least as fast as the prior release. Oracle fully expects that the performance improvements made in Rdb V7.2 for both Alpha and I64 yield a positive result.

We ran many tests back to back on Alpha systems in Oracle's development lab. Comparisons of Rdb V7.2 with Rdb V7.1 running both OLTP and reporting workloads show that Rdb V7.2 is generally faster (over 10% faster for some isolated test cases) or, at worst, equally as fast as Rdb V7.1. In addition, database creation operations with Rdb V7.2 issues only about half of the total number of I/O operations that are issued by Rdb V7.1, while using less CPU time and completing in less elapsed time.

Future I64 Platforms

OpenVMS running on I64 systems performs comparably with Alpha systems, in the size range that we tested. We fully expect that future I64-based processors and systems will provide even higher levels of performance.

Oracle Rdb is still in the early stages of performance analysis and development on OpenVMS I64 systems. We expect that analysis of production applications will yield more information that Oracle will be able to use to improve the performance and reliability of Oracle Rdb.

OpenVMS and Oracle Engineering Credits

The performance tests and improvements described in this paper were carried out in a very short time. Craig Showers and Bill Gabor at the HP OpenVMS benchmark center in Nashua, New Hampshire, arranged and configured the systems used. Christian Moser and Greg Jordan of OpenVMS engineering provided invaluable performance analysis techniques and tools. Without their help, it would have been much more difficult to identify hot spots in the Rdb database engine. They also prototyped significant improvements for several OpenVMS performance-related areas.

All of the Oracle Rdb engineers deserve credit for porting the Rdb product family to OpenVMS on I64 and for providing the highest levels of performance in OLTP workloads.

For more information

For more information, please visit us on the Internet at www.oracle.com/rdb.

© Copyright 2006 Hewlett-Packard Development Company, L.P

OpenVMS Technical Journal V7

The Development Of A High Performance VAX 6000 Emulator

Dr. Robert Boers, CEO, CTO, Software Resources International S.A.

Overview

Software Resources International (SRI) develops commercial emulators for VAX hardware. Designed as a hardware abstraction layer (HAL), a VAX emulator is essentially a software mathematical model of VAX hardware. If the HAL is accurate enough, the original VAX operating systems and applications can be executed on it. This enables the use of unmodified VAX software on any platform for which such a HAL is available, thereby avoiding the cost and risks of using aged VAX hardware. This article describes the development of a HAL for the large VAX SMP systems, the ultimate performance step in replacing existing VAX systems.

Introduction

Since the advent of commercial computing, users have sought to simultaneously take advantage of new hardware advances while preserving their existing applications. Computer manufacturers like Digital Equipment Corporation (DEC) strove for backwards compatibility, but keeping systems compatible at the hardware level limited the innovation that could be applied as larger word lengths, more address space, and more sophisticated operating systems became available. When applications are written in a higher-level language, the amount of required changes can be limited by compiler compatibility, but the source code has to be available. Translating binary application code requires translated system calls[1] and is limited to application code.

Hardware Emulation

The ultimate way to support legacy software is hardware emulation. The computer's CPU interprets binary instructions. It does not matter whether this interpretation is done directly by hardware or by software routines, as long as the ultimate result is same. This is simple in principle, but in reality this solution is complex.

---

[1] Examples of such translators are the FX32 I86-to-Alpha converter or the VAX-to-Alpha and Alpha-to-Itanium converters for OpenVMS applications.

© Copyright 2006 Hewlett-Packard Development Company, L.P

A computer contains not only a CPU, but also peripherals, interconnect hardware, clocks, and many other elements, each of which require a precise representation. The exact hardware component functionality must be recreated in software models, as well as the correct interaction in time between those component models. The result is an accurate software model of computer hardware, or a hardware abstraction layer (HAL), perceived by the legacy software as the original hardware system.

While exact hardware emulation is difficult to achieve, the rewards are significant. If the HAL is accurate enough, the hardware diagnostics and hardware verification tools can be used for testing. The emulator becomes independent of a particular legacy operating system; hence in principle it can run any code that ran on the original hardware. When the HAL is structured as a library of components representing hardware elements, it can be configured in real time into any system configuration for which the emulated elements are available.

Note that HALs can be found in most operating systems, where they make the work of an operating system designer easier by masking variations of the underlying CPUs, minor hardware variations, and so forth. In the case of a hardware system emulator, the HAL is not a thin layer of code requiring few system resources, but a collection of emulated system components with much higher complexity. Such components — and in particular the emulated CPU — require a large amount of computer horsepower. Usable emulators of computer hardware are only feasible due to the much higher performance of the systems on which such emulators are executed, as compared to the original hardware.

The emulation ratio (that is, the number of instructions of the host system required to implement one instruction of the simulated system), is a good metric by which to understand the cost of system emulation. Each emulated component contributes in its own way to the overall performance, so there is no uniform ratio for all components. In practice, we use VUPs (VAX Units of Processing) to compare the performance of our CHARON-VAX emulator products. Tests show that the average emulation ratio measured this way is determined mainly by the efficiency of the simulated components and the host CPU instruction set. The compatibility of host system floating point formats with the legacy hardware influences mathematical performance. A fast thread switching capability in the host system and low memory latency are also important, since emulated CPU components are usually represented as host memory locations.

For example, a VAX 4000 model 90 uses a master clock of 71 MHz for 32 VUPs, while a basic version of CHARON-VAX on a 3 GHz P4 yields approximately 20 VUPs, resulting in an emulation ratio of about 60. Unfortunately, because of the sequential nature of the CPU emulation, executing a HAL on a host system with a larger word length (for instance, emulating a 32-bit VAX system on a 64-bit host) does not provide additional performance, while a shorter host word length carries a heavy performance penalty.

20 VUPs is an acceptable performance for home use and low-end commercial VAX emulation, and in addition to several freeware and hobbyist implementations, SRI's low-end CHARON-VAX/XM product is a bestseller. However, the majority of the business-critical VAX systems still in operation have much higher performance requirements. Single-CPU VAX systems deliver up to 50 VUPs and VAX SMP systems (such as the VAX 7860) can deliver close to 300 VUPs. Replacing such systems with a simple interpretive emulator would require a host system with a clock frequency of 40–50 GHz, which is simply not feasible.

Advanced CPU Emulation (ACE)

In 2002, Software Resources International developed advanced CPU emulation (ACE), which allows us to break the 20 VUPs barrier on currently available I86 architecture using a method similar to the way hardware CPUs use multiple pipelines and look-ahead optimization to improve performance.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Each CPU is represented by two process threads. The first thread analyzes a VAX page of instructions and data, calculates potential future page references, and reorders execution instructions to optimize the use of the host system.[2] As a result, a much larger VAX page is created and buffered.

The second thread executes the processed page, which requires several refinements. Page processing involves a delay that can be intolerably long when driver code is executed, so ACE includes the original VAX page code in the extended page and uses it until the other thread has completed processing. Self-modifying code (for instance, in Oracle RdB) is handled by trapping write operations to VAX instructions, and the instruction sequence is reoptimized. It took a year to tune the prefetch and buffering mechanisms to reach consistently high performance for all VAX system architectures. AXE (the original VAX CPU verification test suite) was used to verify the correct operation of ACE.

The ACE technology allowed us to develop a high performance single-CPU VAX emulator that can deliver up to 80 VUPs on I86 platforms. It is interesting to note that, in spite of their lower CPU frequency, the AMD Opteron-based systems perform better than the Intel Xeon platforms. Opteron CPUs have an on-chip memory controller running at the CPU clock speed, which provides a very low memory latency. They also excel in fast floating point processing. The ACE- based products (sold under the CHARON-VAX *Plus* family name) range from the MicroVAX 3600 (running at 30 times its original speed) to the 512 MB VAX 4100-108, thereby covering most single-CPU VAX systems that are still in operation.

We had exceeded single VAX CPU performance by a big margin. The last hurdle was to reach performance sufficient to replace VAX hardware of any performance range. Following the VAX hardware development history, we had to emulate an SMP VAX system. This has implications for the emulator host system specifications, as SMP system emulation requires running a CPU emulator “*N*” times. The CPU emulation component, driven by the operating system that it is executing, takes all the resources it can get. Hence each emulated CPU requires a dedicated host CPU. The other emulator components cause a much lower load.[3] Therefore, emulating for instance a three-CPU VAX system requires a four-CPU host system.

We started developing a VAX SMP emulator at the end of 2003. At that time, there were not many four-way or eight-way CPU systems on the market, and nearly all had clock frequencies far below that of a common 1 GHz single-CPU system. We needed to emulate at least 3–6 VAX CPUs to make a significant step forward.

Modifying the ACE Design

The functionality of a VAX emulator depends on the VAX model that it represents, but its performance depends on the host system. As shown in the MicroVAX 3600 example, emulating a VAX does not mean it can only run at its “native” speed. For our VAX SMP implementation, we narrowed our choice to either the VAX 6000 or the VAX 7000 family. The VAX 7000 LSB backbone and its attached buses would be much more complex to emulate, so we chose the VAX 6000, which has a well-documented XMI bus. Its synchronous operation and its fixed number of slots (14, of which 10 are available for CPUs or peripheral controllers) allows straightforward configuration. While one team developed the SMP CPU implementation on a skeleton XMI bus, another team focused on the memory subsystem, Ethernet, and disk/tape controllers.

The original ACE implementation was synchronized with its VAX code execution, but with multiple CPUs the page reordering delays were too long to be acceptable. Properly synchronizing the emulated CPUs with the XMI bus involves strict timing constraints. The solution was the implementation of an asynchronous ACE mechanism. As a nice side effect, this new mechanism reduced emulated VAX interrupt latency. The field test experience was so positive that our single-CPU emulator products adopted this method as well.

---

[2] This rather complex part is host system–specific; the rest of the emulator is fully portable.

[3] Consequently, our high performance single-CPU VAX emulators alsorequire a dual-CPU host system.

© Copyright 2006 Hewlett-Packard Development Company, L.P

The VAX 6000 emulator presented unique challenges. For example, every node on the XMI bus is capable of setting up an interrupt request, and each of them is able to respond to that interrupt request, becoming an interrupt server. Unlike single-CPU systems, where the CPU is mostly in control, peripheral nodes can specify which nodes are eligible to become interrupt servers.

The SMP VAX emulator project borrowed components from the existing products but produced many improvements as well. Asynchronous ACE, the most complex component, took only a year to develop by testing on an existing MicroVAX emulator. The XMI MSCP controller was developed the same way. In the SMP project we rewrote most of the emulator core, establishing a new code base for all our emulator products.

Implementing Multi-CPU Emulation

Once the XMI bus behaved properly, the VAX 6610 (single-CPU) emulator posed few problems. However, the step up to multi-CPU emulation required more design work. When multiple CPUs execute the same code, they can share the same VAX page that is currently being analyzed for reordering. For efficiency, each CPU has its own page processor thread. When a thread starts a page-reordering, other threads should back off. If a location is written in a page, its modified version should be invalidated, but another CPU might still be working on it. This required the development of an additional level of synchronization for the VAX page processing.

The solution results in heavily-threaded application code,[4] so the host operating system must be capable of efficient thread switching. To avoid catastrophic failure, CHARON-VAX constantly watches its resources. If there are not enough system resources available to run the VAX page analysis process, it slows down in a "safe" mode to prevent a brutal interruption of the services that the VAX operating system needs to keep running.

A well-functioning VAX 6000 SMP emulator prototype was not yet the result we wanted. Our goal was to create a product family, CHARON-VAX/66x0, capable of replacing all large single-CPU and SMP VAX systems, including large configurations with multiple Ethernet controllers, disks, and several gigabytes of memory, with higher performance.

Our emulation of the standard VAX 6000 hardware was too limited. The largest VAX 6000 memory module was 128 MB, and the KDM70 disk controller supported only eight devices. With only 14 slots to use, and using up one for each VAX CPU, memory board, disk controller, or Ethernet adapter, we could not create, for instance, a six-CPU, 2 GB VAX with 200 disks.

Designing New "Hardware"

Initially we emulated the VAX XMI-to-BI adapter for additional peripheral support (we actually built a prototype). But a more elegant solution emerged. We decided to become "hardware engineers" and design higher-density memory boards and larger disk controllers than the VAX 6000 hardware ever had. We estimated how 256-, 512- or 1024-MB VAX 6000 memory boards would have looked if Digital Equipment Corporation had designed them[5] and we emulated the boards.

We designed the XMI KDM70 disk controller the same way, implementing the MSCP protocol, through which it communicates with the VAX operating system. MSCP devices are autonomous units that inform the operating system of their capabilities. By modifying its protocol responses, we made the controller capable of supporting several thousand disk drives, but because each drive requires a certain amount of buffer space, we limited the number to a more practical 256. Also, we added support for SCSI drives, which the VAX sees as MSCP drives. Similarly, we added support for SCSI tape drives to the modified KDM70.[6]

---

[4] The CHARON-VAX/6660 emulator uses about 30 parallel threads, including two for each emulated CPU.

[5] If they were designed, they never became products.

[6] Data-only, because the VAX 6000 does not know how to boot from a TA tape drive.

© Copyright 2006 Hewlett-Packard Development Company, L.P

To our delight, OpenVMS accepted our modifications without complaining or requiring new drivers. In this way, we produced an SMP VAX emulator that behaves like a 3.5 GB VAX 6000. (The last 0.5 GB is occupied by the XMI I/O space.) For the current products, the emulated memory is limited to 2 GB, and it is easy to configure. You specify the memory size, then the emulator calculates the number and size of the memory boards for the four preallocated XMI memory slots. Even in a six-CPU VAX configuration with four memory boards, four XMI slots remain. These are typically used for one KDM70 and three Ethernet adapters. Each drive can be 8 GB or larger, and we have not seen a user exceed the limit of 256 drives, but a second controller could be configured at the expense of one Ethernet adapter.

The result is a flexible product that can provide fast one-to-six CPU VAX emulation on suitable hardware. Using four 275 dual-core Opteron CPUs in a four-way DL585 (effectively eight cores), the emulator delivers nearly 500 VUPs. We tried to emulate a seven-CPU VAX 6000-670 on an eight-way server, but there was no LMF key for that unusual number of VAX CPUs!

Transfer Licenses

A non-technical aspect of VAX emulation should be noted, involving the legal right to run a licensed VAX operating system and layered products. A few years ago, after we passed the original hardware certification tests, Compaq established transfer licenses for CHARON-VAX, which authorize the transfer of an existing OpenVMS version and specific listed layered software products to the CHARON-VAX emulator, whereby the existing LMF keys can be copied. The transfer licenses have order numbers and can be obtained from HP.

If a user upgrades from a small hardware VAX to a CHARON-VAX system providing a larger VAX system, the OpenVMS or layered products license units copied from the original system are not sufficient. As more customers take the opportunity to consolidate several hardware systems in one powerful emulator, this problem will become more common. To resolve this, we have an agreement with HP to provide the base transfer licenses and the operational licenses for OpenVMS, DECnet, and clustering, depending on the configuration of CHARON-VAX/66x0 products.

The development of three generations of VAX emulators has given us insight into how to design commercial emulators. The CHARON emulator core and our approach to emulator design are not restricted to VAX emulation. Our legacy product, CHARON-11, is selling in increasing numbers. We also implemented a prototype of an HP 3000 emulator, but we have not yet pursued product development. The emulation technology can also be used to economically replace embedded custom computers, and we have been approached by companies to do so. Concerning the CHARON product family, after our successful implementation of 16- and 32-bit systems emulation, we have started work on emulating 64-bit systems, and a prototype booted OpenVMS/Alpha succesfully. Stay tuned.

For more information:

For more information on Software Resources International or the CHARON-VAX family, visit our website at www.softresint.com or contact us at the address shown below. Dr. Boers can be reached via e-mail at: r_boers@softresint.com.

Software Resources International S.A.
Ch. DuPont-Du-Centenaire 109
1228 Plan-les-Ouates
Switzerland
Telephone: (41) 22 794 1070
FAX: (41) 22 794 1073
www.softresint.com

© Copyright 2006 Hewlett-Packard Development Company, L.P

OpenVMS Technical Journal V7

Bringing Seismic Data to the Web with OpenVMS

Dipl. Math. Bernd Ulmann

Overview

This article describes the techniques employed in gathering online seismic data from various geophysical instruments, filtering these data, and creating plots of seismic events that are made publicly available by a web server. All of these tasks are performed by a VAX-7820 running OpenVMS, which has run flawlessly for more than five years now. Excerpts from programming examples show the various interesting parts of the programs involved.

Introduction

When I first began to build seismometers many years ago, I used a conventional strip chart recorder as the main output device. This is a nearly perfect tool for the development of seismometers and geophones, because it allows direct access to the data delivered by the instruments and is readily available in most lab setups. When the instruments matured to a degree where it was possible to detect teleseismic events all over the world from my location in Germany, as well as making it possible to run the instruments unattended for extended periods of time, the strip chart recorder became a burden. At that point, it became necessary to build something which would allow access to the data being gathered by the instruments and to present plots to people located far away from the instruments.

Due to noise considerations, it was necessary to place the instruments as far away from the next house as possible. This required building a sturdy hut on a heavy concrete base plate. This hut is about 20 meters away from the house, so the first problem which needed to be solved was the data transmission over this distance. At first glance, two solutions came to mind.

The first idea involved using a commercially available analog/digital-converter system in conjunction with a PC (normally running Windows). These types of systems are available from a number of companies, such as National Instruments. The alternate idea was to develop a new analog/digital-converter that could send the converted data over a serial line and then develop a method of processing these data.

© Copyright 2006 Hewlett-Packard Development Company, L.P

The first variant could be easily ruled out for at least two reasons. First of all, most commercially available analog/digital-converters either have a resolution too small for seismic applications (12 bits are definitely not enough) or they are far too expensive. More importantly, there is no PC with Windows or anything similar being used, since everything is done on my VAX-7820 running OpenVMS. I would never consider using anything other than my VAX system, so the second variant became the obvious choice.

The first step was to build a multi-channel analog/digital-converter with a resolution of at least 16 bits. This resulted in a design based on the ADS7807 chip from TI, which is a monolithic, high speed, 16-bit converter. This converter was preceded by some signal forming stages, incorporating the necessary low pass filters and a one-out-of-eight analog multiplexer. General control of this setup is done with a 68HC11 microcontroller, which was programmed in assembler language. This control program contains a main loop that samples each of the eight input ports, starts a conversion for each port, and builds a datagram containing two synchronization characters (`0xFF`) followed by eight 16-bit values, as shown in the following diagram:

```
---------------------------------------------------------------------------
! 0xFF ! 0xFF ! LOW_0 ! HIGH_0 ! LOW_1 ! HIGH_1 ! ... ! ... ! LOW_7 ! HIGH_7 !
---------------------------------------------------------------------------
 Synchroniza-   Channel 0    Channel 1          Channel 7
 tion bytes
```

This datagram is then transferred to a host computer system using an asynchronous serial line running at 9600 baud. This converter can run in standalone mode (that is, it has a software time base to trigger conversions in a regular fashion), or it can be used with an external clock, which allows multiple converters to be used together to increase the number of analog input channels. The current installation uses a custom-built time base based on a highly stable crystal oven.

Getting the data to and into the VAX

Because the instrument hut is about 20 meters away from the house, it was decided to buffer the serial line with two leased-line modems which were readily available from another project. (Apart from this converter there are some additional instruments, such as a precision magnetometer, etc., which also communicate by serial lines.) Using these modems, the output from the analog/digital-converter is brought into the house at a speed of 9600 baud. The serial lines are connected to a DECserver 900TM sitting in a DEChub 900, which is in the same network as the previously mentioned VAX-7820. The datagrams are purely binary in their nature, so the configuration of the LAT device to which the converter is connected is a bit tricky. On the OpenVMS side, for example, the terminal setup is as follows:

```
$ SET TERM TA57:/PERM/NOHOSTSYNC/NOWRAP/NOBROAD/NOMODEM/NOECHO-
$_/NOSCOPE/NOTTSYNC/NOLINE/EIGHT/NOHANGUP/PASTHRU/NOINTER-
$_/TYPE_AHEAD/ALTYPEAHD
```

It is necessary to use the alternate type-ahead buffer because datagrams are frequently dropped when the VAX is heavily loaded. This in turn requires setting the system parameter TTY_ALTYPAHD to a value of 4096, which turned out to be enough even for a heavily loaded system. The LAT device used to connect to the serial output line of the converter is defined as follows:

```
$ MC LATCP CREATE PORT LTA57:/APPL
$ MC LATCP SET PORT LTA57:/PORT=PORT_7/NODE=DSRV02
```

Thus, all of the subsequent communication takes place using the device `LTA57`. On the side of the DECserver, the corresponding port is configured as follows:

```
Port  7: (Remote)                       Server: DSRV02
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

```
Character Size:              8          Input Speed:              9600
Flow Control:             None          Output Speed:             9600
Parity:                   None          Signal Control:       Disabled
Stop Bits:                   1          Signal Select:  CTS-DSR-RTS-DTR

Access:                 Remote          Local Switch:             None
Backwards Switch:         None          Name:                   PORT_7
Break:                Disabled          Session Limit:               4
Forwards Switch:          None          Type:                     Ansi
Default Protocol:          LAT          Default Menu:             None
Autolink Timer One:10 Two:10            Dialer Script:            None

Preferred Service: None
Authorized Groups:   0
(Current)  Groups:   0

Enabled Characteristics:
```

Reading the data from `LTA57` is accomplished with a short FORTRAN program called `GATHER.FOR, which` gathers data sent to `LTA57` by the analog/digital-converter on an hourly basis, and then writes the data into a file containing a time stamp in its name. A new file containing raw data is created every hour; meanwhile, the previous file is closed and ready for post processing. The file names look like this: `CHNL_20051105_132304.RAW`. The prefix `CHNL` is a relic from a time when not all of the eight channels were used and the numbers of the channels contained in a file were represented in the file name. The time stamp shows that this file was opened on `05-NOV-2005` at `13:23:04`. The file is organized into eight columns: one column for each channel sampled. The converter samples the analog signals delivered by the seismometers with 25 samples per second. This is sufficient for most purposes because teleseismic events are very low frequency signals. The resulting file contains 25 lines of data per second.

`The GATHER.FOR program` converts the raw data, which is sent in binary (two's complement) from the converter, into single precision floating point numbers to facilitate post processing. These floating point values are written as clear text. While this admittedly wastes some disk space, it makes data transfer to and from other systems (sometimes even PCs) as easy as a copy command. On startup, `GATHER.FOR` first tests to see if the selected input device is indeed a terminal device. For example:

```
      STRUCTURE /ITMLST/
       INTEGER*2 BUFLEN, CODE
       INTEGER*4 BUFADR, RETLENADR
      END STRUCTURE
      RECORD /ITMLST/ DVI_LIST
C
      DVI_LIST.BUFLEN = 4
      DVI_LIST.CODE = DVI$_DEVCLASS
      DVI_LIST.BUFADR = %LOC (CLASS)
      DVI_LIST.RETLENADR = %LOC (CLASS_LEN)
C
      STATUS = SYS$GETDVIW ( , , INPUT_DEVICE, DVI_LIST, , , , , )
      IF ((.NOT. STATUS) .AND. (STATUS .NE. SS$_IVDEVNAM))
      1 CALL LIB$SIGNAL (%VAL (STATUS))
      IF ((STATUS .NE. SS$_IVDEVNAM) .AND. (CLASS .EQ. DC$_TERM)) THEN
...continue processing...
      ELSE
       ...abort program...
      ENDIF
```

Following these two initial steps, the `GATHER.FOR` program opens a new output file, reads 3,600 times 25 datagrams (an hour's worth of data), converts each datagram into eight floating point values, and writes these to the file.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Reading data from the LAT device is not completely straightforward. Because the data sent by the analog/digital-converter is strictly binary, there is no guarantee that the data part of a datagram will not contain two bytes containing `0xFF` in direct succession, which would lead to confusion with the synchronization bytes. The `GATHER.FOR program` reads single bytes from `LTA57` and waits for the first occurrence of two `0xFF`-bytes. If two bytes like these are received, the program reads the following 16 bytes of raw data, converts it, and writes it into the output file. The program then checks the next two bytes, checking for the value `0xFF`. If this test succeeds, the loop described is triggered again. If this test fails, the `GATHER.FOR program` writes an error message and skips data until it finds two bytes with the value `0xFF`. As a result, the `GATHER.FOR` program leaves a file containing 90,000 lines of data (and an additional first line containing a time stamp), each of which has eight columns containing the data received from the seismometers.

Processing the raw data

The next step in the process is a DCL batch job called `DISPLAY_HOURLY_PLOT.COM`, which checks the directory to which the `GATHER.FOR` program writes the data files for the existence of at least two raw data files. The newest file is always the one the `GATHER.FOR program` currently writes to. Older files have already been closed and are ready for post processing. If there is a file to process, the `DISPLAY_HOURLY_PLOT.COM` batch job starts another program (about 1,600 lines of C code) called `STK`.

`The STK` program reads the contents of a raw data file into memory and applies a digital filter which is configurable by channel. This is necessary because the raw data contains lots of artifacts resulting from noise (traffic, microseismic events caused by storms and waves in oceans, and so forth). Most of these artifacts can be removed easily by using a low-pass filter that is implemented as an FFT-filter (Fast-Fourier-Transform-Filter). First the raw data of a single channel is transformed from the "normal" time domain into the frequency domain (that is, the data is split into spectral information representing the data). This spectral data (the data in the frequency domain) is then modified; in the simplest case, the amplitudes of all frequencies above the desired low-pass frequency are set to zero. Then another FFT-transformation is employed, but in the reverse direction, to transform the data from the frequency domain back into the time domain, leaving data filtered by the low-pass filter. This process is quite time consuming. On the VAX-7820, it takes several minutes to apply the low-pass filter to all eight channels.

Note

> The Fast-Fourier-Transformation is accomplished using `FFTW`, the "Fastest Fourier Transformation in the West." The project homepage for this collection of highly-optimized routines for performing Fourier transformation is located at `http://www.fftw.org`. The OpenVMS port of this package is available at `http://www.vaxman.de/openvms/fftw/fftw-2_1_3_vms.zip`.

The filtered data forms the basis for an hourly plot containing one line of data for each channel sampled. There are many tools available to plot scientific data (`gnuplot`, to name one of the best known). It was tempting to use an out-of-the-box tool like this, but normally these tools are quite powerful, containing lots of techniques which do not apply to the simple problem of plotting some lines of data. Thus, it was decided to create the plots directly from within the `STK` program. The program generates Postscript code that represents an hourly (or, selectively, daily) plots of seismic data. After the `STK` program creates an hourly plot as a Postscript file, ghostscript is used to convert the file to .JPG format. For example:

```
$ GS "-sDEVICE=jpeg -sPAPERSIZE=ledger -dNOPAUSE
-sOutputFile=" destinationfile postscriptfile
```

Once the raw data file is processed, the file is copied to another disk (`DISK$SEISMIC_0:[DATA.RAW]`) for archiving and some additional processing, then the cycle is repeated. The `DISPLAY_HOURLY_PLOT.COM` batch job looks for two files in the directory used by

© Copyright 2006 Hewlett-Packard Development Company, L.P

the `GATHER.FOR program` to which to write its raw data files, and the process continues as described.

Bringing the data to the web

Finally, the `DISPLAY_HOURLY_PLOT.COM` batch job copies the resulting .JPG file to the following directory: `DISK$USER_0:[ULMANN.PUBLIC_HTML.SEISMIC_ONLINE]`. This directory is visible to the Internet through a web server running on the VAX-7820 and contains 24 .JPG files per day, which represent the seismic data gathered from the instruments in the instrumentation hut.

The web server used is WASD. The reasons that I use WASD instead of the featured CSWS are as follows:

- There is no CSWS port for OpenVMS/VAX. Since I have no intention of switching to an Alpha (because the VAX-7820 is rock stable, solid, and a truly wonderful system), this alone would rule out the CSWS.
- I decline to use software that is written for UNIX and then ported to OpenVMS, when there is equivalent software already developed with OpenVMS in mind.

The decision to use WASD turned out to be a good one. It is simple to configure and maintain, and when it comes to CGI scripts (DCL as well as Perl-based scripts), the WASD running on the VAX-7820 is substantially faster than the CSWS on a DS10.

It is cumbersome to scroll through a directory listing containing hundreds of thousands of entries with a web browser. Normally, only the last few dozens of plots are of interest, so I decided to write a small DCL-based CGI script to facilitate the selection of plots to be displayed in a remote web browser. This script is called `SEISMIC_ONLINE.COM` and resides in `DISK$USER_0:[ULMANN.PUBLIC_HTML.CGI-BIN]`. When called initially, it displays a simple web page. This page contains a form with four Submit buttons that are used to select the time frame for which seismic plots should be displayed. The selections are: "One day," "10 days," "30 days," and "All," each allowing the display of a more or less limited list of relevant files.

Selecting one of these buttons causes the `SEISMIC_ONLINE.COM` batch job to be called as a CGI script again, but this time with a parameter. The WASD web server supports a simple mechanism to access parameters like these from within DCL by maintaining symbols, like `WWW_QUERY_STRING` for example. Parameters are sent in the form NAME=VALUE; therefore, the first thing to do is split the contents of `WWW_QUERY_STRING` on the first equal sign found:

```
$ ACTION = F$EDIT (F$ELEMENT (1, "=", WWW_QUERY_STRING), "UPCASE,
COLLAPSE")
```

In the next step, the `SEISMIC_ONLINE.COM` batch job determines the first date for which existing seismic plots shall be displayed:

```
$ IF ACTION .EQS. "ONE+DAY" THEN DATE = F$CVTIME ("TODAY-1-00:00:00")
$ IF ACTION .EQS. "10+DAYS" THEN DATE = F$CVTIME ("TODAY-10-00:00:00")
$ IF ACTION .EQS. "30+DAYS" THEN DATE = F$CVTIME ("TODAY-30-00:00:00")
$ IF ACTION .EQS. "ALL"     THEN DATE = "0000-00-00 00:00:00.00"
$ DATE = F$EXTRACT (0, 4, DATE) + F$EXTRACT (5, 2, DATE) + -
         F$EXTRACT (8, 2, DATE)
```

Then the directory containing the plots (`DISK$USER_0[ULMANN.PUBLIC_HTML.SEISMIC_ONLINE]`) is scanned, and every file satisfying the selected time restriction is displayed as a link in a table with six columns:

```
$ COLUMN_COUNTER = 0
$ FILE_LOOP:
$   FILE = F$SEARCH ("''BASE_DIRECTORY'*.JPG")
$   IF FILE .EQS. "" THEN GOTO END_LOOP
$   FILE = F$ELEMENT (0, ";", F$ELEMENT (1, "]", FILE))
$   FILE_DATE = F$ELEMENT (1, "_", FILE)
```


© Copyright 2006 Hewlett-Packard Development Company, L.P

```
$   IF FILE_DATE .LT. DATE THEN GOTO FILE_LOOP
$   DESCRIPTION = FILE_DATE + "/" + F$ELEMENT (0, ".", F$ELEMENT (2,
"_", FILE))
$   WRITE SYS$OUTPUT "      <TD><A
HREF="""''BASE_URL'''FILE'""><PRE>''DESCRIPTION'</PRE></A></TD>"
$   COLUMN_COUNTER = COLUMN_COUNTER + 1
$   IF COLUMN_COUNTER .EQ. COLUMNS
$    THEN
$     TYPE SYS$INPUT
     </TR>
     <TR>
$     COLUMN_COUNTER = 0
$   ENDIF
$ GOTO FILE_LOOP
$ END_LOOP:
```

Additional tasks and features

Another batch job called `DISPLAY_DAILY_PLOT.COM` does many of the same things as the `DISPLAY_HOURLY_PLOT.COM batch job`, except that it runs only once a day. The processing of 24 files containing 90,000 lines of data with eight columns each takes a considerable amount of CPU time on the VAX-7820 (a normal run takes about an hour). This processing chain is almost identical to the one described above.

The CGI script allowing access to daily plots is called `SEISMIC_DAILY.COM` and after processing is complete, all files of a day's processing are compressed using ZIP and stored on a large disk for later retrieval. In addition to the batch mode of the `STK` program, this tool also allows an interactive display of seismic data. Using its filter techniques, it is possible to analyze a set of data varying the low-pass filter cutoff frequencies for individual channels, scale the data displayed, and so forth. This interactive mode makes heavy use of the `MESA` graphics package available for OpenVMS on the freeware disk.

A similar set of scripts and programs exists for processing data gathered from a high resolution magnetometer, with the exception that this data is displayed in another way. The JPG picture created every hour contains the last 24 hours of data. There is no direct way to access older data via the web.

Conclusion

The system processing data gathered from various geophysical instruments, including seismometers, a magnetometer, and more, has proven to be incredibly reliable. (In all honesty, I expected nothing less, since the basis of all of this work is OpenVMS!) The VAX experiences normal uptimes of up to about 200 days before a power outage takes it offline. Unfortunately, I have no three-phase UPS, so there is no way to survive power failures, which occur about twice a year where I live. Using QIO system calls and an alternate type-ahead buffer for terminal devices, it is possible to eliminate the potential loss of data caused by a heavy load on the VAX.

For more information

Current hourly and daily plots of seismic data may be viewed on the following locations:
http://fafner.dyndns.org/~ulmann/cgi-bin/seismic_online.com
http://fafner.dyndns.org/~ulmann/cgi-bin/seismic_daily.com

Data from the magnetometer is available at
http://fafner.dyndns.org/~ulmann/magnetometer_online/

The author may be reached at ulmann@vaxman.de, and the VAX-7820 is available to the public at http://fafner.dyndns.org.

© Copyright 2006 Hewlett-Packard Development Company, L.P