OpenVMS Technical Journal

V11, April 2008

Table of Contents

Host-Based Minimerge and Automatic Minicopy on Volume Processing in HP Volume Shadowing for OpenVMS

Akila Balasubramanian, OpenVMS Cluster Engineering

Overview

Volume Shadowing for OpenVMS guarantees that data is the same on all the members of a shadow set. The merge recovery operation is used to compare data on shadow set members to ensure that all of them are identical on every logical block. This recovery operation should be done as quickly as possible so that the shadow driver returns the identical data when read from any shadow set member at any time. Several enhancements have been introduced in the area of merge and copy operations. **Host Based Minimerge (HBMM)** improves merge operations by decreasing the number of comparisons needed to complete the merge operation. **Automatic Minicopy on Volume Processing (AMCVP)** helps the removed shadow set member to return to the shadow set using a minicopy operation instead of a full copy. Both of these enhancements allow significant gains in volume shadowing performance.

Overview of Merge Operations

Merge operations are required to avoid any discrepancy between the data (logical block number, or LBN) on shadow set members. During a merge operation, application I/O continues but at a slower rate.

Any of the following events can initiate a merge operation:

1. A system failure results in the possibility of incomplete application writes.

   When a write request to a shadow set fails from the system in which it is mounted, and the system fails before a completion status is returned to the application, the data might be inconsistent on the shadow set members:

   - All members might contain the new data.
   - All members might contain the old data.
   - Some members might contain new data and others might contain old data.

   The exact timing of the failure during the original write request determines the outcome. Volume Shadowing for OpenVMS reconciles the data using the MERGE operation.

2. A shadow set enters mount verification and then times out or aborts mount verification, under certain conditions.

   A shadow set that enters mount verification and either times out or aborts mount verification, will enter a merge state if the following conditions are true:

   - There are outstanding write I/O requests in the shadow driver's internal queues on the system or systems on which it has timed out.
   - The shadow set is mounted on other systems in the cluster.

3. The SET SHADOW/DEMAND_MERGE command is issued.

   This command is useful if:

   - The shadow set was created with INITIALIZE/SHAD command and without the /ERASE qualifier.

- You want to measure the impact of a minimerge or a full merge on I/O throughput.

Types of Merge Operations

A merge operation can be a full merge or minimerge. In a full merge operation, the members of a shadow set are compared with each other to ensure that they contain the same data. This is done by performing a block-by-block comparison of the entire volume and can be a very lengthy procedure.

In a minimerge operation, the merge is performed on only those areas of the shadow set where write activity occurred. This limitation avoids the need for the entire volume scan that is required by full merge operations, thus reducing consumption of system I/O resources.

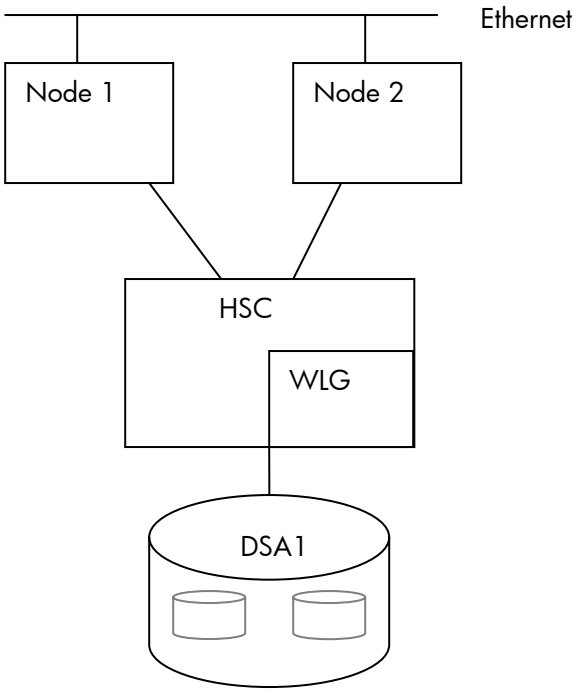
There are two types of minimerge that are based on the characteristic of the device:

- Controller based
- Host based (HBMM)

A controller-based minimerge is performed if the write-logging bit is set for the device in its UCB. Otherwise, a host-based minimerge is performed. If any of the members of the shadow set are capable of controller-based write logging (WLG), then HBMM is not allowed on the shadow set.

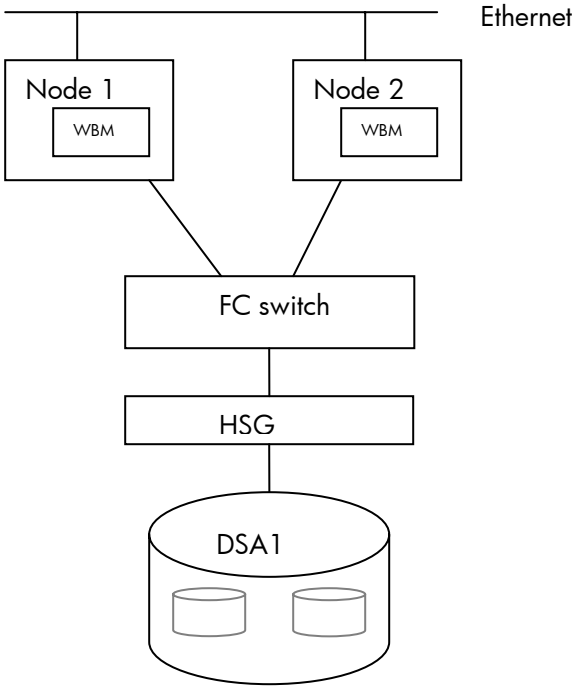
Controller-based minimerge is supported on HSC and HSJ controllers. As shown in Figure 1, by using information about write operations that were logged in controller memory, the minimerge is able to merge only those areas of the shadow set where write activity was known to have been in progress. The write logs contain information about exactly which LBNs in the shadow set had write I/O requests outstanding (from a failed node). The node that performs the minimerge operation uses the WLG to merge those LBNs that might be inconsistent across the shadow set. No controller-based write logs are maintained for a one-member shadow set or if the shadow set is mounted on only one system.

Figure 1. Controller-based minimerge



HBMM can be used with all disks that are supported by Volume Shadowing for OpenVMS except disks on HSJ, HSC, and HSD controllers. As shown in Figure 2, HBMM make use of write bitmaps (WBM) to track the writes that are occurring on the disk. The minimerge operation acts on only the LBNs marked in the bitmap. The next section discusses HBMM in detail.

Figure 2. Host-based minimerge



Write bitmaps

HBMM depends on Write Bitmap (WBM) to perform the merge operation. For every LBN modified, the corresponding bit in the WBM is set. Each bit in the WBM corresponds to 127 blocks. The HBMM uses this WBM and merges only the LBNs set in the WBM. Bitmap entry is recorded before writing to disk.

On OpenVMS Alpha systems, bitmaps are created in S2 space. The memory required for the bitmap is calculated based on the size of the shadow set volume. For every gigabyte of storage of a shadow set mounted on a system, 2 KB of bitmap memory is required on that system for each bitmap.

Local and master bitmaps

Bitmaps can be local bitmaps or master bitmaps. A master WBM contains a record of all the writes to the shadow set from every node in the cluster that has the shadow set mounted. A minimerge operation can occur only on a system with a master bitmap.

A local WBM tracks all the writes that the local node issues to a shadow set. If a node with a local bitmap writes to the same LBN of a shadow set more than once, only the LBN of the first write is sent to the master WBM.

If the same block in a file is modified many times in a shadow set, the local bitmap helps to avoid the many messages to be sent to the master bitmap node to set the already-set bit in the master bitmap. This results in a noticeable performance improvement.

When a node that has the local bitmap and Virtual Unit VU mounted, modifies a LBN, then the "set bit" request is sent to the master bitmap node. The local bitmap bit is set only after the corresponding bit in the master bitmap is set.

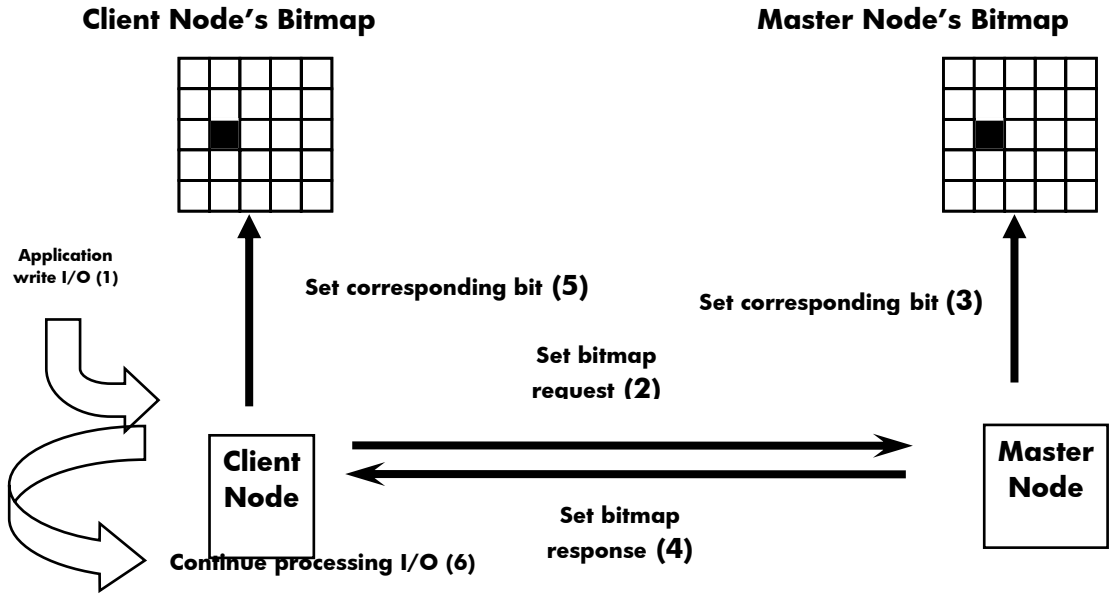
Multiple bitmaps

Existence, state, and master node of a bitmap are coordinated and communicated clusterwide via a per-bitmap lock and its value block. If only one master bitmap exists for the shadow set, and the system with the master bitmap fails or shut down, then the bitmap will not be there. Local bitmaps cannot be used for recovery. This results in a full merge operation. To avoid making the master bit map a single point of failure, HBMM provides the option of specifying more than one node on which master bitmaps are created. Multiple master bitmaps increase the likelihood of an HBMM operation rather than a full merge in the event of a system failure.

Single-message and buffered-message mode

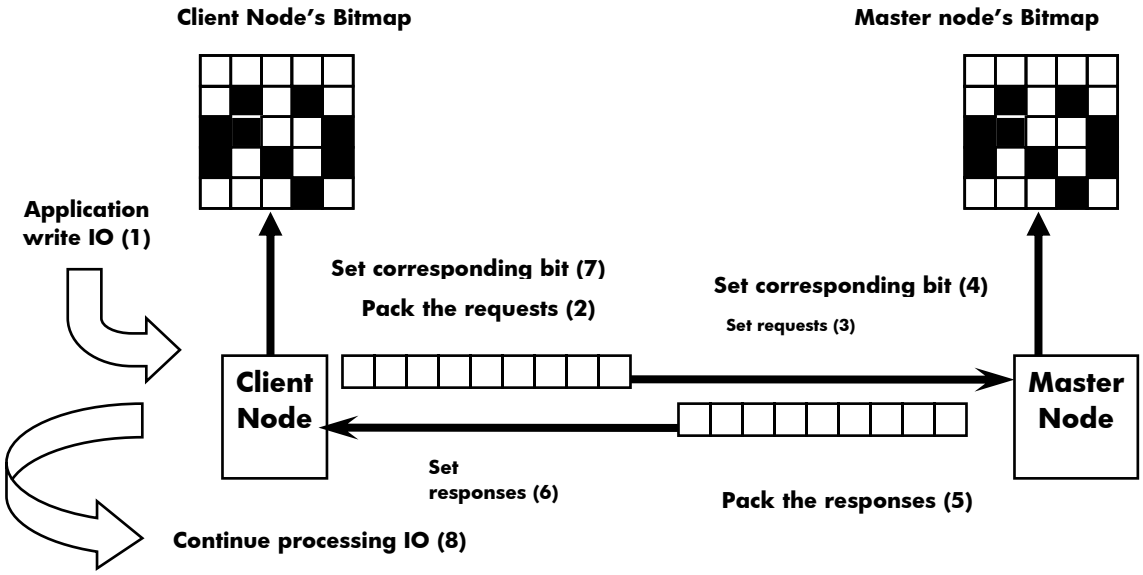
Node-to-node set-bit operations use System Communications Services (SCS)messages. [1] The SCS messages can be sent in single-message mode or buffered-message mode. In single-message mode, only one logical WBM write request per SCS message is sent to the master bitmap node. Figure 3 illustrates single-message mode. In buffered-message mode, shown in Figure 4, up to nine logical WBM write requests per SCS message can be sent. The messages are dispatched to the remote node when the message buffer is full or when the WBM internal timer expires. The WBM internal timer is calculated based on the WBM_MSG_INT system parameter, which specifies the maximum time a message waits before it is sent.

Figure 3. Set-bit Write request to master



1. SCS – System Communication Services provides basic connection management and communication services, implemented as a logical path, between system applications on nodes in an OpenVMS Cluster system.

Figure 4. Buffered Set- Bit write request to master bitmap



HBMM policy

HBMM policies are defined to implement the decisions regarding master bitmaps. HBMM policy specifies the following attributes for one or more shadow sets:

- **MASTER_LIST** – Names of systems that are eligible to host a master bitmap.
- **COUNT** – Number of systems that will host a master bitmap (not to exceed six). If this number is omitted, the first six available systems of those you specified are selected.
- **RESET_THRESHOLD** – Threshold (in 512-byte blocks) at which the bitmaps are reset. If omitted, the threshold defaults are applied. In OpenVMS Version 8.3 and higher, the default is 1,00,000 blocks. For prior versions, the default is 50,000 blocks.
- **MULTIUSE** – Turns on AMCVP.

Writes that need to set bits in the bitmap are slightly slower than writes to areas that are already marked as having been written. Therefore, if many of the writes to a particular shadow set are concentrated in certain ''hot'' files, then the reset threshold should be made large enough so that the same bits are not constantly set and then cleared. On the other hand, if the reset threshold is too large, then the advantages of HBMM are reduced. For example, if 50% of the bitmap is populated (that is, 50% of the shadow set has been written to since the last reset), then the HBMM merge will take approximately 50% of the time of a full merge.

Using the SET SHADOW/POLICY command is the only method for specifying HBMM policies. You use the SET SHADOW/POLICY command with HBMM-specific qualifiers to define, assign, deassign, and delete policies and to enable and disable HBMM on a shadow set.

The following example shows the SET SHADOW/POLICY command and its output:

```
$ SET SHADOW DSA999:/POLICY=HBMM ( -
_$ (MASTER_LIST=(NODE1,NODE2,NODE3), COUNT=2), -
_$ (MASTER_LIST=(NODE4,NODE5), COUNT=1), -
_$ (MASTER_LIST=(NODE6,NODE7,NODE8), COUNT=2), -
_$ RESET_THRESHOLD=500000)

$ SHOW SHADOW DSA999:/POLICY=HBMM
HBMM Policy for device _DSA999:
HBMM Reset Threshold: 500000
HBMM Master lists:
Up to any 2 of the nodes: NODE1, NODE2, NODE3
Any 1 of the nodes: NODE4, NODE5
Up to any 2 of the nodes: NODE6, NODE7, NODE8
```

In this example:

- 5 master bitmaps will be present.
- The following nodes will host master bitmaps:
  Any 2 of NODE1, NODE2, and NODE3 (Site 1)
  Any 1 of NODE4, NODE5 (Site 2)
  Any 2 of NODE6, NODE7, NODE8 (Site 3) nodes.
- A threshold of 500000 blocks must be reached before clearing the bitmap.

HBMM policy attempts to maintain at least one HBMM master bitmap at each site in a multiple-site OpenVMS Cluster system, thereby minimizing the need to perform a full merge.

Some sites might find that a single HBMM policy can effectively implement the decisions. Other sites might need greater granularity and therefore implement multiple policies. The most likely need for multiple policies is when the cluster includes enough high-bandwidth systems that you want to ensure that the merge load is spread out. Multiple HBMM policies are also useful when shadow sets need

different bitmap reset thresholds. The master list can be the same for each policy, but the threshold can differ.

Activating HBMM

HBMM is automatically activated on a shadow set under the following conditions:

- An HBMM policy exists for a given shadow set and that shadow set is then mounted on one or more systems defined in the master list.
- An HBMM policy is created for a mounted shadow set and at least one system that has it mounted is defined in the master list.

You can also activate HBMM with the SET SHADOW/ENABLE=HBMM command, provided a policy exists and the shadow set is mounted on a system defined in the master list of the shadow set policy, and the count has not been exceeded.

Hierarchy of Transient State Operations and Shadow Set Priority

A shadow set is in a **steady state** when none of the following operations is pending or active:

- Minimerge
- Minicopy
- Full copy
- Full merge

A shadow set is in a **transient state** if it has one or more of these operations pending, or one operation active. Although a combination of these transient states is valid, only one operation at a time can be performed.

Shadow set operations for a specific shadow set are performed in the following order:

1. Minimerge
2. Copy (either minicopy or full copy)
3. Full merge

You can assign a unique priority to every mounted shadow set on a per-system basis using the `SET SHADOW/PRIORITY=`*`n`*` DSA`*`n`* command. The priority assigned to a shadow set does not affect the hierarchy of transient state operations.

For example, if HBMM is not enabled after a device is added to a shadow set, it is marked as being in a full-copy transient state. If the system on which this shadow set is mounted fails, the shadow set is further marked as being in a full-merge state. In this case, the full copy operation is performed before the full merge is started.

A merge transient state is an event that cannot be predicted. The management of merge activity, on a specific system for multiple shadow sets, can be predicted if the priority level settings for the shadow sets differ.

In the following example, there are four shadow sets, and the SYSGEN parameter SHADOW_MAX_COPY on this system is equal to 1. A value of 1 means that only one merge or copy operation can occur at the same time. This example illustrates how the priority level is used to select shadow sets when only merge operations are involved.

Two shadow sets are assigned a priority level and two have the default priority level of 5000. The four shadow sets DSA1, DSA20, DSA22, and DSA42, are mounted on two systems. DSA20 and DSA42 are minimerge enabled.

```
$ SET SHADOW/PRIORITY=7000 DSA1:
$ SET SHADOW/PRIORITY=3000 DSA42:
 ! DSA20: and DSA22: are at the default priority level of 5000
```

In this example, when one system fails, all shadow sets are put into a merge-required state. The SHADOW_REC_DLY system parameter specifies the length of time a system waits before it attempts to manage recovery operations on shadow sets that are mounted on the system. After the delay resulting from recovery of this significant event, this system evaluates the shadow sets and the operations are performed in the following order:

1. A minimerge operation starts on DSA20, even though its priority of 5000 is lower than DSA1's priority of 7000. A minimerge operation always takes precedence over other operations. DSA20 and DSA42 are both minimerge enabled, but DSA20's higher priority causes its minimerge operation to start first.

2. A minimerge operation starts on DSA42. Its priority of 3000 is the lowest of all the shadow sets, but a minimerge operation takes precedence over other operations.

3. Because there are no other minimerge capable units, DSA1, with a priority level of 7000, is selected to start a merge operation, and it runs to completion.

4. A merge operation starts on DSA22, the one remaining shadow set whose priority is the default value of 5000, and runs to completion.

AMCVP – Multiuse bitmap

Automatic Minicopy on Volume Processing (AMCVP) means that an existing HBMM bitmap is made available to function as a minicopy bitmap. Specifically, the minimerge bitmap is made "Multiuse" bitmap. Before the introduction of automatic bitmap creation on volume processing, returning expelled members to a shadow set, after connectivity was restored, was a lengthy process. The expelled members could be returned only by undergoing a full copy. The availability of a multiuse bitmap enables the use of a minicopy operation, which takes considerably less time than a full copy operation. To enable AMCVP, you need to establish an HBMM policy for the shadow sets, and include the new MULTIUSE keyword in the policy.

The conversion of HBMM bitmap to minicopy bitmap happens automatically when connectivity to one or more shadow set members is lost and is not restored during the member's timeout period. When such connectivity is lost, the shadow set is paused for volume processing—that is, writes and reads are temporarily suspended until connectivity is restored or until the timeout period (established by the value of the SHADOW_MBR_TMO parameter) expires, whichever comes first.

If connectivity is not restored by the end of the timeout period, the lost members are removed from the shadow set, read and write I/O to the remaining member resumes, and the bitmap keeps track of the writes. The bitmap functions as a minicopy bitmap for the members that are removed. When the removed member is reinstated, then minicopy is performed.

While one or two members are expelled and after all members are restored to membership in the shadow set, the HBMM bitmap functionality remains in effect. The HBMM bitmap functionality is useful in the case of an expelled member only when the shadow set has three members and one member is expelled.

The following example shows how the HBMM bitmap is used as a multiuse bitmap and results in a minicopy operation. Shadow set DSA1: has two members 1DKA100: (connected to NODE1, available to others via MSCP) and 2DKB200: (connected to NODE2, available to others via MSCP). The shadow set policy for DSA1: is set to use HBMM bitmap as a multiuse bitmap, thereby enabling AMCVP. The master WBM is available on nodes NODE1 and NODE2.

```
$ SET SHADOW DSA1:/POLICY=HBMM ( -
_$ (MASTER_LIST= (NODEA), COUNT=1, MULTIUSE=1), -
_$ (MASTER_LIST= (NODEB), COUNT=1, MULTIUSE=1))
```

```
$ SHOW SHAD DSA1

_DSA1:    Volume Label: SHAD2
  Virtual Unit State:   Steady State
  Enhanced Shadowing Features in use:
        Dissimilar Device Shadowing (DDS)
        Host-Based Minimerge (HBMM)
        Automatic Minicopy (AMCVP)

  VU Timeout Value      3600    VU Site Value           0
  Copy/Merge Priority   5000    Mini Merge        Enabled
  Recovery Delay Per Served Member                     30
  Merge Delay Factor     200    Delay Threshold       200

  HBMM Policy
    HBMM Reset Threshold: 1000000
    HBMM Master lists:
      Any 1 of the nodes: NODE1 Multiuse: 1
      Any 1 of the nodes: NODE2 Multiuse: 1
    HBMM bitmaps are active on NODE2,NODE1
    Modified blocks since bitmap creation: 254

  Device $1$DKA100              Master Member
    Read Cost              2    Site 0
    Member Timeout       120

  Device $2$DKB200
    Read Cost            501    Site 0
    Member Timeout       120
```

The following command displays information such as the type of Bitmap, master write bitmap nodes, bitmap name, and so on:

```
$ SHOW DEVICE /BIT/FULL
Device         BitMap     Size     Percent     Type of   Master  Active  Creation
Cluster   Local Delete   Bitmap
 Name            ID      (Bytes)  Populated    Bitmap     Node           Date/Time
Size     Set  Pending  Name
DSA1:      0030007     17496      0.01%   Minimerge  NODE2    Yes  18-JUL-2007 02:08:22.34
127     0.01%    No    HBMC$DSA0001HBMM00010007
                00050008     17496      0.01%   Minimerge  NODE1    Yes  18-JUL-2007
02:12:18.55  127     0.01%    No    HBMC$DSA0001HBMM00010005
```

The network cable to NODE2 was pulled off for a period of time greater than the value of the SHADOW_MBT_TMO parameter. Now connectivity to shadow set member 2DKB200: (connected to NODE2) is lost by the shadow set. The connection cannot be restored until a time period equal to the value of the SHADOW_MBT_TMO parameter has elapsed. Hence, 2DKB200: is removed from

shadow set DSA1. At this point, the minimerge bitmap becomes a multiuse bitmap and keeps track of writes to the shadow set, as shown in the following example:

```
$ SHOW SHADOW DSA1/BIT/FULL
Device          BitMap     Size     Percent     Type of   Master  Active  Creation
Cluster   Local Delete   Bitmap
 Name            ID      (Bytes)  Populated    Bitmap     Node           Date/Time
Size     Set  Pending  Name
DSA1:          00050008     17496      0.01%   Multiuse   NODE1    Yes  18-JUL-2007
02:12:18.55  127     0.01%    No    HBMC$SHAD20000 18-JUL-2007 02:12:19.27
```

The network cable of NODE2 is now plugged in again. The node crashed (CLUEXITed) and then rebooted because of the long delay. When shadow set member 2DKB200 is brought back to the shadow set, as shown in the following example, the minicopy takes place. In the preceding example, the bitmap for NODE2 disappeared because NODE2 was rebooted.

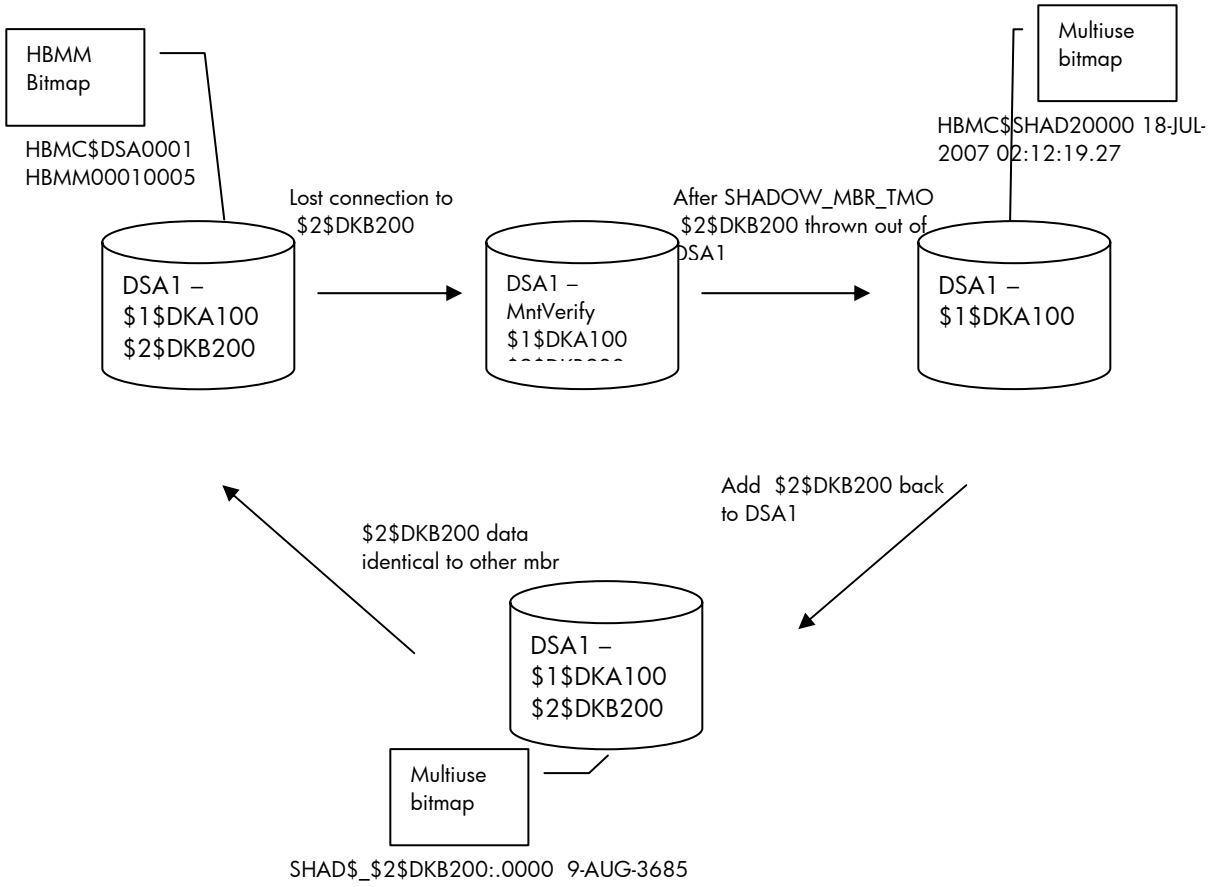
```
$ MOUNT/SYS DSA1:/SHAD=$2$DKB200: SHAD2
%MOUNT-I-MOUNTED, SHAD2 mounted on _DSA1:
%MOUNT-I-SHDWMEMCOPY,_$2$DKB200:(NODE2) added to the shadow set with a
copy operation
%MOUNT-I-ISAMBR, _$1$DKA100: (NODE1) is a member of the shadow set

$ SHOW DEVICE DSA1/BIT/FULL
Device          BitMap     Size     Percent     Type of   Master  Active  Creation
Cluster   Local Delete   Bitmap
 Name            ID      (Bytes)  Populated    Bitmap     Node           Date/Time
Size     Set  Pending  Name
DSA1:          00050008     17496      0.01%   Multiuse   NODE1    Yes  18-JUL-2007
02:12:18.55  127     0.01%    No    SHAD$_$2$DKB200:.0000  9-AUG-3685 22:40:30.34

$ SHOW SHADOW DSA1
_DSA1:    Volume Label: SHAD2
  Virtual Unit State:   MiniCopy Active (90%) on NODE1
  Enhanced Shadowing Features in use:
        Dissimilar Device Shadowing (DDS)
        Host-Based Minimerge (HBMM)
        Automatic Minicopy (AMCVP)

  VU Timeout Value      3600    VU Site Value           0
  Copy/Merge Priority   5000    Mini Merge        Enabled
  Recovery Delay Per Served Member                     30
  Merge Delay Factor     200    Delay Threshold      200

  HBMM Policy
    HBMM Reset Threshold: 1000000
    HBMM Master lists:
      Any 1 of the nodes: NODE1 Multiuse: 1
      Any 1 of the nodes: NODE2 Multiuse: 1
    HBMM bitmaps are active on NODE1
    Modified blocks since bitmap creation: 254

  Device $1$DKA100              Master Member
    Read Cost              2    Site 0
    Member Timeout       120

  Device $2$DKB200              Copy Target (90%)
    Read Cost            501    Site 0
    Member Timeout       120
```

The multiuse bitmap now becomes the normal HBMM bitmap, as shown in the following example:

```
$ SHOW DEV DSA1/BIT/FULL
Device          BitMap     Size     Percent     Type of   Master  Active  Creation
Cluster   Local Delete   Bitmap
 Name            ID      (Bytes)  Populated    Bitmap     Node           Date/Time
Size     Set  Pending  Name
DSA1:          00050008     17496      0.01%   Minimerge  NODE1    Yes  18-JUL-2007
02:12:18.55  127     0.01%    No    HBMC$DSA0001HBMM00010005
```

Figure 5 summarizes the process that occurred in the preceding command examples.

Figure 5 Summary of steps involved in AMCVP



For more information

For more information about HBMM and AMCVP, see the following OpenVMS manuals:

- *HP OpenVMS Version 8.2 New Features and Documentation Overview*
- *HP OpenVMS Version 8.3 New Features and Documentation Overview*
- *HP Volume Shadowing for OpenVMS*

Hints and Tricks When Using Dynamic Volume Expansion (DVE) on OpenVMS Systems

Rob Eulenstein, Multivendor Systems Engineering

Overview

This article provides hints and tricks for using Dynamic Volume Expansion (DVE) on OpenVMS systems. Dynamic volume expansion, which was first available in OpenVMS Alpha Version 7.3-2 and HP OpenVMS for Integrity servers Version 8.2, allows system managers to increase the size (the number of logical blocks) of a mounted volume. Although often associated with OpenVMS Host-Based Volume Shadowing, DVE can be implemented on nonshadowed volumes.

Introduction

The early groundwork for DVE began with OpenVMS Alpha Version 7.2. Prior to that release, the size of the [000000]BITMAP.SYS file was limited to 256 blocks. In Version 7.2, the maximum size of BITMAP.SYS was increased to 65,536 blocks. The initial impact of this change was to allow disks to be initialized with much smaller disk cluster sizes than was previously possible. Prior to Version 7.2, the smallest disk cluster size was approximately equal to the total blocks on the volume divided by 1 million (255 x 4096). For large volumes that contained many small files, this limitation resulted in much wasted space. With the change in Version 7.2, volumes as large as 400 GB could have a disk cluster size of 3 blocks. As we discuss in this article, allowing for a larger BITMAP.SYS file was a key prerequisite for DVE.

At the present time, the maximum volume size supported by OpenVMS is slightly less than 1 TB. The actual number is 2,147,475,456 blocks decimal or 7FFFE000 hexadecimal. The OpenVMS file system (Extended QIO Processor, or XQP) as well as current versions of SYS$DKDRIVER enforce this limit. For the history buffs among us, in OpenVMS VAX Version 5.5-2 and earlier versions, the maximum supported volume size was 16,777,215 blocks decimal or 00FFFFFF hexadecimal. The maximum number of files on an OpenVMS volume is 16,711,679. This value is calculated as follows: 2**24 – 2**16 – 1.

Prior to OpenVMS Alpha Version 7.3-2, when a volume became full, the system manager had a couple of choices. They could delete or purge files on the volume to free up some space, or they could move all the files and directories via BACKUP or COPY to a larger volume. The output of the SHOW DEVICE/FULL command was often used to monitor free space. The output showed the total blocks on the volume plus the number of free blocks on the volume. The total blocks was the value from the unit control block (UCB) field UCB$L_MAXBLOCK. This same value was also stored in the storage control block (SCB) field SCB$L_VOLSIZE. (The SCB is virtual block number, or VBN, 1 of BITMAP.SYS.) Figure 1 is an example of the output of the pre-Version 7.3-2 SHOW/DEVICE/FULL command.

```
$ show device dsa218/full

Disk DSA218:, device type MSCP served SCSI disk, is online, mounted, file-
    oriented device, shareable, available to cluster, error logging is enabled.

    Error count                    0    Operations completed            670148
    Owner process                 ""    Owner UIC                  [SYS,SYSTEM]
    Owner process ID        00000000    Dev Prot        S:RWPL,O:RWPL,G:RW,W:RW
    Reference count                1    Default buffer size                512
    Total blocks             2050353    Sectors per track                   57
    Total cylinders             2570    Tracks per cylinder                 14

    Volume label             "DISK18"    Relative volume number              0
    Cluster size                   3    Transaction count                   1
    Free blocks              1708314    Maximum files allowed          256294
.
.
.
```

Figure 1 – Pre-Version 7.3-2 SHOW DEVICE/FULL Command Output

To help manage DVE, two new items were added to the SHOW DEVICE/FULL output in OpenVMS Alpha Version 7.3-2: Logical Volume Size and Expansion Size Limit. Logical Volume Size is the value from the SCB$L_VOLSIZE field in the SCB. Expansion Size Limit is calculated when the volume is mounted, as shown in Figure 2. The value calculated is stored in the volume control block (VCB) field VCB$L_EXPSIZE.

```
Expansion Size Limit = (blocks allocated to BITMAP.SYS – 1) *
                       disk cluster size * 4096
```

Figure 2 – Calculating Expansion Size Limit

Figure 3 shows an example of the new SHOW DEVICE/FULL output.

```
$ show device/full dsa218:

Disk DSA218:, device type MSCP served SCSI disk, is online, mounted, file-
    oriented device, shareable, available to cluster, error logging is enabled,
    device supports bitmaps (no bitmaps active).

    Error count                    0    Operations completed               228
    Owner process                 ""    Owner UIC               [FIELD,SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                1    Default buffer size                512
    Total blocks             2050353    Sectors per track                   57
    Total cylinders             2570    Tracks per cylinder                 14
    Logical Volume Size      2050353    Expansion Size Limit            2052096

    Volume label             "DISK18"   Relative volume number                0
    Cluster size                   3    Transaction count                     1
    Free blocks              1708314    Maximum files allowed            256294
.
.
.
```

Figure 3 – Version 7.3-2 SHOW DEVICE/FULL Output

In this version of the command output, the Total blocks value is still displayed and continues to represent UCB$L_MAXBLOCK from the UCB. The UCB$L_MAXBLOCK value represents the number of logical blocks presented to OpenVMS by the storage controller. Logical Volume Size is the number of logical blocks on the volume currently in use by the OpenVMS file system (the XQP). In Figure 3, the Total blocks value equals the Logical Volume Size value, which is often the case on disks that cannot be expanded (local SCSI disks), or on disks that have already completed DVE. Expansion Size Limit represents the largest number of logical blocks that can be mapped by the current size of the BITMAP.SYS file. In Figure 3, Expansion Size Limit is slightly higher than Total blocks and Logical Volume Size for two reasons:

- The number of blocks allocated to BITMAP.SYS must be divisible by the disk cluster size.
- Logical Volume Size might not be evenly divisible by 4096 (512 bytes x 8 bits per byte).

Figure 4 shows the relationship between Expansion Size Limit and the size of the BITMAP.SYS file.

```
$ dir/size=all dsa218:[000000]bitmap.sys

Directory DSA218:[000000]
```

```
BITMAP.SYS;1              168/168

Total of 1 file, 168/168 blocks.

$ expsize = (168 - 1) * 3 * 4096

$ show symbol expsize
  EXPSIZE = 2052096
```

Figure 4 – Relationship between Expansion Size Limit and size of BITMAP.SYS

In addition to the new items added to the SHOW DEVICE/FULL output, new qualifiers were added for the DCL commands INITIALIZE and SET VOLUME. The INIT/LIMIT, INIT/SIZE, SET VOLUME/LIMIT, and SET VOLUME/SIZE commands were added to implement DVE. For both commands, the /LIMIT qualifier affects the size of BITMAP.SYS and, ultimately, the Expansion Size Limit value for the volume. The /SIZE qualifier sets or increases (you can not decrease) the Logical Volume Size. The Logical Volume Size value can never exceed the Expansion Size Limit for the volume.

How DVE works

Now that we have some background, let's see how DVE works. If you are initializing a new disk from an OpenVMS system, preparing for a future expansion of this volume is easy. Simply add the /LIMIT qualifier to the INIT command. This results in preallocating sufficient blocks to BITMAP.SYS to map up to the current 1TB volume-size limit. The actual size of the BITMAP.SYS file created depends on the disk cluster size on the volume. In Figure 5, OpenVMS defaults to a disk cluster size of 8 blocks because 8 blocks is the smallest disk cluster size possible for a 1TB volume.

```
$ init/limit $1$dga150: scratch
%INIT-I-DEFCLUSTER, value for /CLUSTER defaulted to 8

$ mount/system $1$dga150: scratch
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$DGA150: (HSV2AL)

$ show device/full $1$dga150:

Disk $1$DGA150: (HSV2AL), device type HSV210, is online, mounted, file-oriented
    device, shareable, available to cluster, device has multiple I/O paths,
    error logging is enabled.

    Error count                    0    Operations completed              1274
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                1    Default buffer size                512
    Current preferred CPU Id       0    Fastpath                             1
    WWID   01000010:6005-08B4-0010-3F6B-0000-9000-052E-0000
    Total blocks            75497472    Sectors per track                  128
    Total cylinders             4608    Tracks per cylinder                128
    Logical Volume Size     75497472    Expansion Size Limit         2147450880
    Allocation class               1

    Volume label           "SCRATCH"    Relative volume number               0
    Cluster size                   8    Transaction count                    1
    Free blocks             75427792    Maximum files allowed         16711679
.
.
.

$ dir/size=all $1$dga150:[000000]bitmap.sys

Directory $1$DGA150:[000000]

BITMAP.SYS;1           2305/65536
```

```
Total of 1 file, 2305/65536 blocks.

$ expsize = (65536 - 1) * 4096 * 8

$ show symbol expsize
  EXPSIZE = 2147450880   Hex = 7FFF8000
```

Figure 5 – INIT/LIMIT Command Example

In Figure 5, if 1DGA150: becomes full, the volume can be expanded without dismounting. The system manager must first increase the size of 1DGA150: from the storage controller. Because 1DGA150: is an EVA-based disk (LUN), the system manager can use the StorageWorks utility Command View EVA to increase the size of 1DGA150:. In this example the size was increased from 36 GB to 72 GB.

Next, the system manager enters the SET VOLUME/SIZE command on the OpenVMS system. If the volume is mounted by multiple cluster nodes, it is necessary to enter the SET VOLUME/SIZE command from only one node. The other cluster nodes acknowledge the larger volume size the next time an I/O involving the XQP occurs. In Figure 6, the blocks allocated to the BITMAP.SYS file do not increase; only the "used" blocks increase.

```
At this point $1$DGA150: was grown from 36 GB to 72 GB at the controller.

$ set volume/size $1$dga150:

$ show device/full $1$dga150:

Disk $1$DGA150: (HSV2AL), device type HSV210, is online, mounted, file-oriented
    device, shareable, available to cluster, device has multiple I/O paths,
    error logging is enabled.

    Error count                    0    Operations completed              8045
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                1    Default buffer size                512
    Current preferred CPU Id       0    Fastpath                             1
    WWID   01000010:6005-08B4-0010-3F6B-0000-9000-052E-0000
    Total blocks           150994944    Sectors per track                  128
    Total cylinders             9216    Tracks per cylinder                128
    Logical Volume Size    150994944    Expansion Size Limit         2147450880
    Allocation class               1

    Volume label           "SCRATCH"    Relative volume number               0
    Cluster size                   8    Transaction count                    1
    Free blocks            150925264    Maximum files allowed         16711679
.
.
.

$ dir/size=all $1$dga150:[000000]bitmap.sys

Directory $1$DGA150:[000000]

BITMAP.SYS;1            4609/65536

Total of 1 file, 4609/65536 blocks.
```

Figure 6 – SET VOLUME/SIZE Command Example

As easy and straightforward as this example is, most system managers do not have the luxury of being able to start with a brand new disk. Rather, they must implement DVE on existing volumes that

already contain data. In these cases, the SET VOLUME/LIMIT command must be used instead of the INIT/LIMIT command to extend the existing BITMAP.SYS file. The one significant restriction when using SET VOLUME/LIMIT is that **the volume must be mounted privately.** Failure to do so results in the following error:

```
$ set volume/limit $1$dga150:
%SET-E-NOTMOD, $1$DGA150: not modified
-SET-W-NOTPRIVATE, device must be mounted privately
```

This restriction is by far the biggest hurdle that system managers face when using DVE. It is the one aspect of DVE that is not truly dynamic. However, once BITMAP.SYS has been extended, the Logical Volume Size value can be increased again and again by using SET VOLUME/SIZE commands while the volume is mounted using the /SYSTEM or /CLUSTER qualifier, as shown in Figure 7.

```
$ init $1$dga150: scratch     <----  Notice That /LIMIT Was Not Specified And
                                     Disk Cluster Size Defaulted to 145 Blocks

$ mount/system $1$dga150: scratch
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$DGA150: (HSV2AL)

$ show device/full $1$dga150:

Disk $1$DGA150: (HSV2AL), device type HSV210, is online, mounted, file-oriented
    device, shareable, available to cluster, device has multiple I/O paths,
    error logging is enabled.

    Error count                    0    Operations completed              8962
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                1    Default buffer size                512
    Current preferred CPU Id       0    Fastpath                             1
    WWID   01000010:6005-08B4-0010-3F6B-0000-9000-052E-0000
    Total blocks           150994944    Sectors per track                  128
    Total cylinders             9216    Tracks per cylinder                128
    Logical Volume Size    150994944    Expansion Size Limit          171642880
    Allocation class               1

    Volume label           "SCRATCH"    Relative volume number               0
    Cluster size                 145    Transaction count                    1
    Free blocks            150993575    Maximum files allowed           517105
.
.
.

At this point $1$DGA150: was grown from 72 GB to 144 GB at the controller.

$ set volume/limit $1$dga150:
%SET-E-NOTMOD, $1$DGA150: not modified
-SET-W-NOTPRIVATE, device must be mounted privately

$ set volume/size $1$dga150:

$ show device/full $1$dga150:

Disk $1$DGA150: (HSV2AL), device type HSV210, is online, mounted, file-oriented
    device, shareable, available to cluster, device has multiple I/O paths,
    error logging is enabled.

    Error count                    0    Operations completed              9121
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                1    Default buffer size                512
    Current preferred CPU Id       0    Fastpath                             1
    WWID   01000010:6005-08B4-0010-3F6B-0000-9000-052E-0000
    Total blocks           301989888    Sectors per track                  128
```

```
    Total cylinders            18432    Tracks per cylinder                128
    Logical Volume Size    171642880    Expansion Size Limit          171642880
    Allocation class               1

    Volume label           "SCRATCH"    Relative volume number              0
    Cluster size                 145    Transaction count                   1
    Free blocks            171641430    Maximum files allowed          517105
.
.
.

Notice that the above SET VOLUME/LIMIT command failed; however, the SET
VOLUME/SIZE command did increase the "Logical Volume Size" up to maximum number
of blocks that the current size of BITMAP.SYS could map.

$ dismount $1$dga150:

$ mount/over=id $1$dga150:
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$DGA150: (HSV2AL)

$ set volume/limit $1$dga150:  <----  Command Works; Volume Mounted Privately

$ set volume/size $1$dga150:           <----  This Bug Will Be Fixed In The
%SET-E-NOTSET, error modifying $1$DGA150:     Next Round Of F11X Remedial Kits;
-SYSTEM-F-BADPARAM, bad parameter value       Workaround Is To DISMOUNT And
                                              Then MOUNT
$ dismount $1$dga150:

$ mount/system $1$dga150: scratch
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$DGA150: (HSV2AL)

$ set volume/size $1$dga150:

$ show device/full $1$dga150:

Disk $1$DGA150: (HSV2AL), device type HSV210, is online, mounted, file-oriented
    device, shareable, available to cluster, device has multiple I/O paths,
    error logging is enabled.

    Error count                    0    Operations completed               9788
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                1    Default buffer size                 512
    Current preferred CPU Id       0    Fastpath                              1
    WWID   01000010:6005-08B4-0010-3F6B-0000-9000-052E-0000
    Total blocks           301989888    Sectors per track                   128
    Total cylinders            18432    Tracks per cylinder                 128
    Logical Volume Size    301989888    Expansion Size Limit         2152366080
    Allocation class               1

    Volume label           "SCRATCH"    Relative volume number              0
    Cluster size                 145    Transaction count                   1
    Free blocks            301984975    Maximum files allowed          517105
.
```

Figure 7 – Expanding an Existing Volume

Dynamic volume expansion also works in conjunction with another new feature in OpenVMS Alpha Version 7.3-2: dissimilar device shadowing (DDS). Prior to Version 7.3-2, all members of a host-based shadow set (DSA*nnnn*: *device*) were required to have the same number of logical blocks. Dissimilar device shadowing allows a larger disk to be added to an existing shadow set. Once the full copy operation completes, the smaller shadow members can be removed from the shadow set. Dynamic volume expansion can then be used on the virtual unit (the DSA*nnnn*: *device*) to increase the Logical Volume Size value.

Note: At some point the virtual unit must be mounted privately to extend BITMAP.SYS. Removing a member, performing DVE on this removed member, and then adding this removed member back into

the shadow set accomplishes nothing because the removed member becomes a full copy target when it is re-added to the shadow set. The following scenario illustrates the recommended way to migrate the data on an existing host-based shadow set to a larger volume.

A system manager has an existing 2-member shadow set, DSA100:, which is running low on free blocks. The system manager wants to use DDS and DVE to migrate the data on this shadow set to a larger volume. During the migration, down time must be kept to a minimum and availability of the data on this shadow set must be kept to a maximum. The current physical members of DSA100: are 1DGA160: and 1DGA170:. Both of these disks (LUNs) are 18 GB in size. At the storage controller, the system manager has created two new 36GB disks -- 1DGA180: and 1DGA190: -- and has presented these new disks to OpenVMS. Figure 8 shows the initial setup in this scenario.

```
$ show device dsa100:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label        Blocks Count Cnt
DSA100:                 Mounted              0  PROD_DATA     1248084   323   3
$1$DGA160:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA170:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)

$ show device/full dsa100:

Disk DSA100:, device type Generic SCSI disk, is online, mounted, file-oriented
    device, shareable, available to cluster, error logging is enabled, device
    supports bitmaps (no bitmaps active).

    Error count                    0    Operations completed           57833416
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count              322    Default buffer size                512
    Total blocks            37748736    Sectors per track                  128
    Total cylinders             2304    Tracks per cylinder                128
    Logical Volume Size     37748736    Expansion Size Limit           39100416

    Volume label         "PROD_DATA"    Relative volume number                0
    Cluster size                  37    Transaction count                   325
    Free blocks              1248084    Maximum files allowed            496693
.
.
.

$ init $1$dga180: scratch
$ init $1$dga190: scratch

$ mount/over=id $1$dga180:
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$DGA180: (HSV2AL)
$ mount/over=id $1$dga190:
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$DGA190: (HSV2AL)

$ show device $1$dga180:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label        Blocks Count Cnt
$1$DGA180:    (HSV2AL)  Mounted alloc        0  SCRATCH      75496527     1   1

$ show device $1$dga190:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label        Blocks Count Cnt
$1$DGA190:    (HSV2AL)  Mounted alloc        0  SCRATCH      75496527     1   1

$ dismount $1$dga180:
$ dismount $1$dga190:
```

Figure 8 – Using DDS and DVE on an Existing Shadow Set – Initial Setup

Step 1: The first step is to add one of the larger, 36GB disks to the DSA100: shadow set, and then to allow the full copy operation to complete. The shadow set now contains three physical members, 1DGA160:, 1DGA170:, and 1DGA180:, as shown in Figure 9.

```
$ mount/system dsa100:/shadow=$1$dga180: prod_data
%MOUNT-I-MOUNTED, PROD_DATA mounted on _DSA100:
%MOUNT-I-SHDWMEMCOPY, _$1$DGA180: (HSV2AL) added to the shadow set with a copy
operation
%MOUNT-I-ISAMBR, _$1$DGA160: (HSV2AL) is a member of the shadow set
%MOUNT-I-ISAMBR, _$1$DGA170: (HSV2AL) is a member of the shadow set


$ show device dsa100:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label        Blocks Count Cnt
DSA100:                 Mounted              0  PROD_DATA     1248084   336   3
$1$DGA160:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA170:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA180:    (HSV2AL)  ShadowCopying        0  (copy trgt DSA100:   8% copied)

.
.
.

$ show device dsa100:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label        Blocks Count Cnt
DSA100:                 Mounted              0  PROD_DATA     1248084   318   3
$1$DGA160:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA170:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA180:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
```

Figure 9 – Using DDS and DVE on an Existing Shadow Set – Step One

Step 2: At this point, one of the original, smaller disks (1DGA160:) can be removed from the shadow set and the second, larger disk (1DGA190:) can be added into the shadow set. As before, allow the full copy operation to complete. Figure 10 illustrates this step.

```
$ dismount $1$DGA160:

$ mount/system dsa100:/shadow=$1$dga190 prod_data
%MOUNT-I-MOUNTED, PROD_DATA mounted on _DSA100:
%MOUNT-I-SHDWMEMCOPY, _$1$DGA190: (HSV2AL) added to the shadow set with a copy
operation
%MOUNT-I-ISAMBR, _$1$DGA170: (HSV2AL) is a member of the shadow set
%MOUNT-I-ISAMBR, _$1$DGA180: (HSV2AL) is a member of the shadow set

$ show device dsa100:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label        Blocks Count Cnt
DSA100:                 Mounted              0  PROD_DATA     1248084   297   3
$1$DGA170:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA180:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA190:    (HSV2AL)  ShadowCopying        0  (copy trgt DSA100:  27% copied)

.
.

$ show device dsa100:

Device                  Device           Error    Volume         Free  Trans Mnt
```

```
 Name                  Status           Count     Label       Blocks Count Cnt
DSA100:                Mounted              0  PROD_DATA     1248084   302   3
$1$DGA170:    (HSV2AL) ShadowSetMember      0  (member of DSA100:)
$1$DGA180:    (HSV2AL) ShadowSetMember      0  (member of DSA100:)
$1$DGA190:    (HSV2AL) ShadowSetMember      0  (member of DSA100:)
```

Figure 10 – Using DDS and DVE on an Existing Shadow Set – Step Two

Step 3: The third step is to remove the one remaining small disk (1DGA170:) from the shadow set. Disk DSA100: now has just 1DGA180: and 1DGA190: as its physical members; however, there is still only 18GB of usable space on the volume. The Total blocks value has increased to 36 GB, but the Logical Volume Size value is still 18 GB, as shown in Figure 11.

```
$ dismount $1$DGA170:

$ show device dsa100:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label       Blocks Count Cnt
DSA100:                 Mounted              0  PROD_DATA     1248084   291   3
$1$DGA180:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA190:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)

$ show device/full dsa100:

Disk DSA100:, device type Generic SCSI disk, is online, mounted, file-oriented
    device, shareable, available to cluster, error logging is enabled, device
    supports bitmaps (no bitmaps active).

    Error count                    0    Operations completed             892037
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count              282    Default buffer size                 512
    Total blocks            75497472    Sectors per track                   128
    Total cylinders             4608    Tracks per cylinder                 128
    Logical Volume Size     37748736    Expansion Size Limit           39100416

    Volume label         "PROD_DATA"    Relative volume number                0
    Cluster size                  37    Transaction count                   285
    Free blocks              1248084    Maximum files allowed            496693
.
.
.
```

Figure 11 – Using DDS and DVE on an Existing Shadow Set – Step Three

Step 4: We have maintained data availability during the migration by ensuring that there are at least two members in the shadow set at all times. This final step requires a short period of down time because the volume was not originally initialized with the /LIMIT qualifier. We have to use the SET VOLUME/LIMIT command to extend BITMAP.SYS, and this command requires that the volume be mounted privately. On each cluster node, the system manager must reduce the transaction count to 1 prior to dismounting DSA100:, as shown in Figure 12.

```
$ show device dsa100:

Device                  Device           Error    Volume         Free  Trans Mnt
 Name                   Status           Count     Label       Blocks Count Cnt
DSA100:                 Mounted              0  PROD_DATA     1248084     1   3
$1$DGA180:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)
$1$DGA190:    (HSV2AL)  ShadowSetMember      0  (member of DSA100:)

$ dismount/cluster dsa100:
```

```
$ mount/over=id DSA100:/shadow=($1$DGA180:,$1$DGA190:)
%MOUNT-I-MOUNTED, PROD_DATA mounted on _DSA100:
%MOUNT-I-SHDWMEMSUCC, _$1$DGA180: (HSV2AL) is now a valid member of the shadow
set
%MOUNT-I-SHDWMEMSUCC, _$1$DGA190: (HSV2AL) is now a valid member of the shadow
set

$ set volume/limit dsa100:

$ dismount dsa100:

$ mount/cluster DSA100:/shadow=($1$DGA180:,$1$DGA190:) prod_data
%MOUNT-I-MOUNTED, PROD_DATA mounted on _DSA100:
%MOUNT-I-SHDWMEMSUCC, _$1$DGA180: (HSV2AL) is now a valid member of the shadow
set
%MOUNT-I-SHDWMEMSUCC, _$1$DGA190: (HSV2AL) is now a valid member of the shadow
set

$ set volume/size dsa100:

$ show device/full dsa100:

Disk DSA100:, device type Generic SCSI disk, is online, mounted, file-oriented
    device, shareable, available to cluster, error logging is enabled, device
    supports bitmaps (no bitmaps active).

    Error count                    0    Operations completed             22573
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count               47    Default buffer size                512
    Total blocks            75497472    Sectors per track                  128
    Total cylinders             4608    Tracks per cylinder                128
    Logical Volume Size     75497472    Expansion Size Limit        2147491840

    Volume label         "PROD_DATA"    Relative volume number              0
    Cluster size                  37    Transaction count                  50
    Free blocks             38982904    Maximum files allowed          496693
.
.
.
```

Figure 12 – Using DDS and DVE on an Existing Shadow Set – Final Step

In Figure 12, notice that the Expansion Size Limit for DSA100: is now 1 TB; therefore, future dynamic volume expansions on this shadow set can be done without mounting privately. Disk DSA100: can be grown to 72 GB, 144 GB, or even larger while the volume is mounted and in use clusterwide.

A Closer Look at BITMAP.SYS

The preceding scenario makes multiple references to the BITMAP.SYS file and to the SCB, which is VBN number 1 in the BITMAP.SYS file. The following discussion takes a more in-depth look at some of the fields in a SCB and at the contents of an actual bitmap block. The SCB shown in Figure 13 is from the DSA100: shadow set, which we used in the previous DDS/DVE example. I have identified some of the more important and interesting fields.

```
$ set default dsa100:[000000]

$ dump/block=count=1 bitmap.sys

Dump of file DSA100:[000000]BITMAP.SYS;1 on 30-AUG-2007 07:54:05.47
```

```
File ID (2,2,0)   End of file block 500 / Allocated 14171

Virtual block number 1 (00000001), 512 (0200) bytes

 00000080 00000001 04800000 00250201 ................ 000000
                   ^^^^^^^^ <------ SCB$L_VOLSIZE Field - 75497472 Decimal
 00000000 0000000E 00001200 00000080 ................ 000010
 21772020 20415441 445F444F 52500001 ..PROD_DATA   .. 000020
 ^^^^
 00A6CDB6 BAE76216 000000A6 CDB6B5A9 ................ 000030
 |||||||| ||||||||     ^^^^ ^^^^^^^^ <------ SCB$Q_MOUNTTIME Field
 ^^^^^^^^ ^^^^^^^^ <------ SCB$Q_GENERNUM Field
 00000000 A0A00001 12610064 00000000 ................ 000040
                   ^^^^^^^^ ^^^^^^^^ <------ SCB$Q_UNIT_ID Field - Virtual Unit
 10E100BE 00000001 10E100B4 00000001 ................ 000050
 |||||||| |||||||| ^^^^^^^^ ^^^^^^^^ <------ SCB$Q_MEMBER_IDS (Index 0)
 ^^^^^^^^ ^^^^^^^^ <------ SCB$Q_MEMBER_IDS (Index 1)
 00000202 022CF4A8 00000000 00000000 .........,..... 000060
                   ^^^^^^^^ ^^^^^^^^ <------ SCB$Q_MEMBER_IDS (Index 2)
 00000000 00000000 00000000 00000000 ................ 000070
 00000000 00000000 00000000 00000000 ................ 000080
 00000000 00000000 00000000 00000000 ................ 000090
 00000000 00000000 00000000 00000000 ................ 0000A0
 00000000 00000000 00000000 00000000 ................ 0000B0
 00000000 00000000 00000000 00000000 ................ 0000C0
 00000000 00000000 00000000 00000000 ................ 0000D0
 00000000 00000000 00000000 00000000 ................ 0000E0
 00000000 00000000 00000000 00000000 ................ 0000F0
 00000000 00000000 00000000 00000000 ................ 000100
 00000000 00000000 00000000 00000000 ................ 000110
 00000000 00000000 00000000 00000000 ................ 000120
 00000000 00000000 00000000 00000000 ................ 000130
 00000000 00000000 00000000 00000000 ................ 000140
 00000000 00000000 00000000 00000000 ................ 000150
 00000000 00000000 00000000 00000000 ................ 000160
 00000000 00000000 00000000 00000000 ................ 000170
 00000000 00000000 00000000 00000000 ................ 000180
 00000000 00000000 00000000 00000000 ................ 000190
 00000000 00000000 00000000 00000000 ................ 0001A0
 00000000 00000000 00000000 00000000 ................ 0001B0
 00000000 00000000 00000000 00000000 ................ 0001C0
 00000000 00000000 00000000 00000000 ................ 0001D0
 00000000 00000000 00000000 00000000 ................ 0001E0
 E9A10004 00000000 00000000 00000000 ..............!i 0001F0
 ||||^^^^ <------ SCB$W_SHADOWING_STATUS Field
 ^^^^ <------ SCB$W_CHECKSUM Field
```

Figure 13 – Example Storage Control Block (SCB)

The SCB$Q_UNIT_ID field and the SCB$Q_MEMBER_IDS fields have a unique format. In each of these quadword fields, the low-order longword contains the allocation class, the low-order word in the high-order longword is the unit number, and the high-order word in the high-order longword contains a special 5-bit encoding scheme for the controller designation. Let's decode the SCB$Q_UNIT_ID field and the SCB$Q_MEMBER_IDS fields.

The SCB$Q_UNIT_ID field contains 12610064 00000000. The allocation class is 0 (low-order longword), and the unit number is 100 decimal (low-order word in the high-order longword). The 1261 hex is 0001 0010 0110 0001 in binary. Starting with bit 0 on the right, group these bits into groups of 5 bits: 0 00100 10011 00001. Ignore the lone 0 on the left and translate the 5-bit groups into decimal to yield 4, 19 and 1. The corresponding letters of the alphabet are D (4th letter), S (19th letter), and A (1st letter). This yields DSA as the controller designation. The complete device name is DSA100:.

The SCB$Q_MEMBER_IDS field for index 0 contains 10E100B4 00000001. The allocation class is 1 (low-order longword), and the unit number is 180 decimal (low-order word in the high-order longword). The 10E1 hex is 0001 0000 1110 0001 in binary. Starting with bit 0 on the right, group these bits into groups of 5 bits: 0 00100 00111 00001. Ignore the lone 0 on the left and translate the 5-bit groups into decimal to yield 4, 7, and 1. The corresponding letters of the alphabet are D (4th letter), G (7th letter), and A (1st letter). This yields DGA as the controller designation. The complete device name is 1DGA180:.

The SCB$Q_MEMBER_IDS field for index 1 contains 10E100BE 00000001. The allocation class is 1 (low-order longword), and the unit number is 190 decimal (low-order word in the high-order longword). The 10E1 hex is 0001 0000 1110 0001 in binary. Starting with bit 0 on the right, group these bits into groups of 5 bits: 0 00100 00111 00001. Ignore the lone 0 on the left and translate the 5-bit groups into decimal to yield 4, 7, and 1. The corresponding letters of the alphabet are D (4th letter), G (7th letter), and A (1st letter). This yields DGA as the controller designation. The complete device name is 1DGA190:.

The SCB$Q_MEMBER_IDS field for index 2 contains 00000000 00000000 because there is no third member in this shadow set.

Figure 14 provides a closer look at the contents of an actual bitmap block. In a bitmap block, a clear bit (0) means the corresponding disk cluster is allocated. A set bit (1) means the corresponding disk cluster is free.

```
$ dump/block=(start=2,count=1) bitmap.sys

Dump of file DSA100:[000000]BITMAP.SYS;1 on 30-AUG-2007 10:50:33.45
File ID (2,2,0)   End of file block 500 / Allocated 14171

Virtual block number 2 (00000002), 512 (0200) bytes

 00000000 00000000 00000000 00000000 ................ 000000
 00000000 00000000 00000000 00000000 ................ 000010
 00000000 00000000 00000000 00000000 ................ 000020
 00000000 00000000 00000000 00000000 ................ 000030
 00000000 00000000 00000000 00000000 ................ 000040
 00000000 00000000 00000000 00000000 ................ 000050
 00000000 00000000 00000000 00000000 ................ 000060
 00000000 00000000 00000000 00000000 ................ 000070
 00000000 00000000 00000000 00000000 ................ 000080
 00000000 00000000 00000000 00000000 ................ 000090
 00000000 00000000 00000000 00000000 ................ 0000A0
 00000000 00000000 00000000 00000000 ................ 0000B0
 00000000 00000000 00000000 00000000 ................ 0000C0
 00000000 00000000 00000000 00000000 ................ 0000D0
 00000000 00000000 00000000 00000000 ................ 0000E0
 00000000 00000000 00000000 00000000 ................ 0000F0
 00000000 00000000 00000000 00000000 ................ 000100
 00000000 00000000 00000000 00000000 ................ 000110
 00000000 00000000 00000000 00000000 ................ 000120
 00000000 00000000 00000000 00000000 ................ 000130
 00000000 00000000 00000000 00000000 ................ 000140
 00000000 00000000 00000000 00000000 ................ 000150
 00000000 00000000 00000000 00000000 ................ 000160
 00000000 00000000 00000000 00000000 ................ 000170
 00000000 00000000 00000000 00000000 ................ 000180
 00000000 00000000 00000000 00000000 ................ 000190
 00000000 00000000 00000000 00000000 ................ 0001A0
 00000000 00000000 00000000 00000000 ................ 0001B0
 00000000 00000000 00000000 00000000 ................ 0001C0
 00000000 00000000 00000000 00000000 ................ 0001D0
 00000000 00000000 00000000 00000000 ................ 0001E0
 00000000 00000000 00000000 00000000 ................ 0001F0
```

Figure 14 – Example Bitmap Block – All Disk Clusters Allocated

The bitmap block in Figure 14 is not very interesting. All the bits are clear, which means the first 4096 disk clusters (the first 151,552 blocks) on DSA100: are allocated. The 151,552 number was calculated by multiplying 4096 times 37, the disk cluster size. What would this same bitmap block look like if a large file on DSA100: was deleted, as shown in Figure 15?

```
$ dir

Directory DSA100:[000000]

000000.DIR;1        BACKUP.SYS;1        BADBLK.SYS;1        BADLOG.SYS;1
BIG_FILE.DAT;1      BITMAP.SYS;1        CONTIN.SYS;1        CORIMG.SYS;1
INDEXF.SYS;1        SECURITY.SYS;1      VOLSET.SYS;1

Total of 11 files.

$ delete/log BIG_FILE.DAT;1
%DELETE-I-FILDEL, DSA100:[000000]BIG_FILE.DAT;1 deleted (36500019 blocks)

$ dump/block=(start=2,count=1) bitmap.sys

Dump of file DSA100:[000000]BITMAP.SYS;1 on 30-AUG-2007 10:52:56.69
File ID (2,2,0)   End of file block 500 / Allocated 14171

Virtual block number 2 (00000002), 512 (0200) bytes
 FFFFFFFF FFFFFFFF FFFFFFFF F7FFFFFC ................ 000000
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000010
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000020
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000030
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000040
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000050
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000060
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000070
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000080
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000090
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0000A0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0000B0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0000C0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0000D0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0000E0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0000F0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000100
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000110
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000120
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000130
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000140
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000150
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000160
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000170
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000180
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 000190
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0001A0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0001B0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0001C0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0001D0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0001E0
 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................ 0001F0
```

Figure 15 – Example Bitmap Block – Most Disk Clusters Free

This reveals quite a different picture. Most of the disk clusters are now free. The very first longword in this block contains F7FFFFFC. All of the other longwords contain FFFFFFFF. In F7FFFFFC, bits 0, 1,

and 27 are clear. This means that out of the 151,552 blocks mapped by this bitmap block, only blocks (LBNs) 0 through 73 and 999 through 1035 are allocated.

Actual DDS/DVE Cases

The topics discussed in the next few paragraphs were taken directly from calls about DVE and DDS that were logged to the Customer Support Center. In most cases, these problems and pitfalls have been reported on more than one occasion.

When a customer implements DVE, one error they might encounter if they run ANALYZE/DISK is: %ANALDISK-I-SHORTBITMAP, the storage bitmap on RVN 1, does not cover the entire device. Although this informational error appears ominous, in almost every case it can be safely ignored. To understand this SHORTBITMAP error, we need to examine the conditions under which it occurs.

A word-length field at offset 508 decimal (1FC hex) in the SCB is used to hold status flags. The name of this field is SCB$W_SHADOWING_STATUS. As of OpenVMS Version 7.3-2, three bits are defined in this field. Bit 0 is the SCB$V_INIT_NO_ERASE bit. This bit is set when the INIT/SHADOW/NOERASE command was used to initialize the volume. Bit 1 is the SCB$V_DVE_ENABLED bit. This bit is set when either the Expansion Size Limit or the Logical Volume Size value does not equal the Total blocks value when the volume is initialized. Bit 2 is the SCB$V_HBVS_MEMBERS_MAY_DIFFER bit. Figure 16 shows the SCB$W_SHADOWING_STATUS field in the SCB.

```
$ dump/block=count=1 dsa600:[000000]bitmap.sys

Dump of file DSA600:[000000]BITMAP.SYS;1 on  8-OCT-2006 09:17:28.70
File ID (2,2,0)   End of file block 1087 / Allocated 65536

Virtual block number 1 (00000001), 512 (0200) bytes

 00000020 00000001 021EAE18 00080201 .....Z...... ... 000000
 00000000 00000006 000087AC 00000020  ...Z........... 000010
 79212020 20314B53 49445245 53550001 ..USERDISK1   !y 000020
 00A59093 1A8A34FA 000000A5 8D85D4E3 cT..#...z4....#. 000030
 .
 .
 .
 00000000 00000000 00000000 00000000 ................ 0001D0
 00000000 00000000 00000000 00000000 ................ 0001E0
 A5120002 00000000 00000000 00000000 ...............% 0001F0
     ^^^^<---- SCB$W_SHADOWING_STATUS Field, Bits 0 & 2 Are Clear, Bit 1 Is Set
```

Figure 16 - SCB$W_SHADOWING_STATUS Field in an SCB

The SHORTBITMAP error occurs only if the SCB$V_DVE_ENABLED bit (bit 1) is clear in the SCB$W_SHADOWING_STATUS field, and if the Total blocks value on the volume is greater than the current Logical Volume Size. Initializing a disk device with either the /SIZE or the /LIMIT qualifier sets the SCB$V_DVE_ENABLED bit. Using the SET VOLUME/LIMIT command does not set the SCB$V_DVE_ENABLED bit.

```
$ show device/full $6$dka200:

Disk $6$DKA200: (WSC236), device type COMPAQ BF01885A34, is online, mounted,
    file-oriented device, shareable, served to cluster via MSCP Server, error
    logging is enabled.

    Error count                    0    Operations completed            814899
    Owner process                 ""    Owner UIC                      [SYSTEM]
```

```
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                1    Default buffer size                 512
    Current preferred CPU Id       0    Fastpath                              1
    Total blocks            35565080    Sectors per track                    32
    Total cylinders            34732    Tracks per cylinder                  32
    Logical Volume Size       5000000   Expansion Size Limit          2147450880
    Allocation class               6
    .
    .
    .



$dump/block=count=1 $6$dka200:[000000]bitmap.sys

Dump of file $6$DKA200:[000000]BITMAP.SYS;1 on  8-OCT-2006 09:56:23.13
File ID (2,2,0)   End of file block 154 / Allocated 65536

Virtual block number 1 (00000001), 512 (0200) bytes

 0000002F 00000001 004C4B40 00080201 ....@KL...../... 000000
 00000000 0000000E 000008D8 0000002F /...X........... 000010
 FF822020 20202048 43544152 43530001 ..SCRATCH     .. 000020
 00000000 00000000 000000A5 90A480AB +.Z.#........... 000030
 .
 .
 .
 00000000 00000000 00000000 00000000 ................ 0001D0
 00000000 00000000 00000000 00000000 ................ 0001E0
 90D00000 00000000 00000000 00000000 ..............Z. 0001F0
     ^^^^ <------  SCB$W_SHADOWING_STATUS Field, SCB$V_DVE_ENABLED Bit Is Clear


$ analyze/disk $6$dka200:
Analyze/Disk_Structure for _$6$DKA200: started on  8-OCT-2006 09:56:37.97
%ANALDISK-I-SHORTBITMAP, storage bitmap on RVN 1 does not cover the entire device
%ANALDISK-I-OPENQUOTA, error opening QUOTA.SYS
-SYSTEM-W-NOSUCHFILE, no such file
```

Figure 17 – Example of the %ANALDISK-I-SHORTBITMAP Error

There is no corruption on this volume. The conditions described are likely to occur if the system manager has grown the volume from the storage controller and has used the SET VOLUME/LIMIT command, but has not yet used the SET VOLUME/SIZE command.

Another pitfall when attempting to expand a volume is the dreaded SYSTEM-W-DEVICEFULL error. Often the volume is being expanded because free space is insufficient. Why, then, do you get this error (shown in Figure 18), which essentially tells you what you already know?

```
$ set volume/limit $1$dga300:
%SET-E-NOTSET, error modifying $1$DGA300:
-SYSTEM-W-DEVICEFULL, device full; allocation failure
```

Figure 18 – Example SYSTEM-W-DEVICEFULL Error

The "device full" error occurs when there is insufficient contiguous free space on the volume to create the new, larger BITMAP.SYS file. If the volume in question is totally full, you have no choice but to free up some disk space and then re-enter the SET VOLUME/LIMIT command. If there is some free space on the volume, try a small increase in volume size first. Then, using this newly created space (which, by definition, has to be contiguous), enter the SET VOLUME/LIMIT command again to extend BITMAP.SYS to map up to 1 TB. In Figure 19, the volume in question was originally 36 GB and was nearly out of free space. From the storage controller, the system manager had grown the volume to 72 GB.

```
$ mount/over=id $1$dga300:
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$dga300: (HSV2AL)

$ set volume/limit $1$dga300:
%SET-E-NOTSET, error modifying $1$DGA300:
-SYSTEM-W-DEVICEFULL, device full; allocation failure

$ set volume/limit=80000000 $1$dga300:

$ dismount $1$dga300:

$ mount/over=id $1$dga300:
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$dga300: (HSV2AL)

$ set volume/size $1$dga300:

$ set volume/limit $1$dga300:

$ dismount $1$dga300:

$ mount/system $1$dga300: scratch
%MOUNT-I-MOUNTED, SCRATCH mounted on _$1$dga300: (HSV2AL)

$ set volume/size $1$dga300:
```

Figure 19 – Workaround to the SYSTEM-W-DEVICEFULL Error

Do not try to expand a volume by just a few blocks. If the expansion requested in the SET VOLUME/SIZE command is less than 256 times the disk cluster size, the expansion requested is ignored; however, no error is returned.

Note: The SET VOLUME/SIZE command cannot be used to decrease the size of a volume. If the new Logical Volume Size is less than the present Logical Volume Size, an SS$_UNSUPPORTED error is returned, as shown in Figure 20.

```
$ show device/full dsa100:

Disk DSA100:, device type Generic SCSI disk, is online, mounted, file-oriented
    device, shareable, available to cluster, error logging is enabled, device
    supports bitmaps (no bitmaps active).

    Error count                    0    Operations completed            297277
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count              251    Default buffer size                512
    Total blocks             37748736    Sectors per track                  128
    Total cylinders              2304    Tracks per cylinder                128
    Logical Volume Size      37748736    Expansion Size Limit          39100416

    Volume label            "SCRATCH"    Relative volume number               0
    Cluster size                   37    Transaction count                  254
    Free blocks              37748103    Maximum files allowed           496693
.
.
.

$ set volume/size=17000000 dsa100:
%SET-E-NOTSET, error modifying _DSA100:
-SYSTEM-E-UNSUPPORTED, unsupported operation or function
```

Figure 20 – SS$_UNSUPPORTED Error

Finally, the exact number displayed for the Expansion Size Limit in the output of the SHOW DEVICE/FULL command depends on the version of the VMSxxx_MOUNT96 patch kit that is installed

on the system. As mentioned previously the Expansion Size Limit for a volume is calculated as follows:

```
Expansion Size Limit = (blocks allocated to BITMAP.SYS – 1) *
                       disk cluster size * 4096
```

Based on this formula, systems with older or no MOUNT96 kit installed might display a number larger than 2,147,475,456 decimal for Expansion Size Limit. The display of this errant number is only cosmetic. The latest versions of VMS*xxx*_MOUNT96 kits properly limit the number displayed to 2,147,475,456 decimal blocks.

Summary

Dynamic volume expansion and dissimilar device shadowing are two of the most useful new features in OpenVMS Alpha Version 7.3-2 and HP OpenVMS for Integrity servers Version 8.2. When using the INIT or SET VOLUME commands, the /LIMIT qualifier is used to preallocate space to or extend BITMAP.SYS. The /SIZE qualifier is used to set or to increase the Logical Volume Size. Remember that when you use the SET VOLUME/LIMIT command, the volume in question must be mounted privately.

WebSphere MQ and OpenVMS Failover Sets

John Edelmann, Technology Consultant

Overview

WebSphere MQ is a messaging middleware product whose primary function is the transfer of a datagram from one application to another on one computer system, or from one application to an application running on another computer system.

There are many advantages to such a messaging model, the primary ones being related to ease of use from the application standpoint; the benefit of secured, guaranteed, and trusted delivery of an individual message; and the availability of a robust array of recovery features.

As with most such messaging products on the market today, MQ is only as robust as the underlying operating system and platform integrity permits. Thus, it is both proper and fortunate that IBM exploited the unique active-active clustering features of the OpenVMS operating system, when it designed the unique failover capabilities embodied in the 5.3 release.

Because the features of OpenVMS Clusters are well known to the OpenVMS technical community, this document focuses on the WebSphere MQ capabilities designed in the OpenVMS implementation.

WebSphere MQ Clusters

This report is not intended to replace the MQ Clustering Manual or other basic IBM WebSphere MQ product documentation. Please refer to these documents for a more in depth review of the product set.

WebSphere MQ includes a clustering capability (not to be confused with OpenVMS Clusters) that allows a number of queue managers, running on the same or different computer servers or platforms, to share the same named queue object to any application that must write a message to the queue. An application writing a message from the context of a queue manager that actually "hosts" the queue in question always puts messages to the queue of the local queue manager. This is not the real intent behind clustering, however.

An application putting a message to a queue from a queue manager that is a member of the cluster, but that *does not have a local instance of the queue*, puts the message to the queue hosted by other members of the cluster, in a round-robin methodology and with a certain level of failover provided. This assumes that processes reading the messages from the cluster queues can process them in a similar round-robin fashion. If messages need to be processed in a certain order (called message affinity), other clustering features (such as bind options) need to be incorporated in the application model.

WebSphere MQ Clusters are most useful for multiple-site operations, where the various sites are connected by a WAN or LAN (thus, single or multiple "customers" put messages to a multiple-site operation for workload distribution and high availability), or where multiple sites with many application queues and independent servers need to share queues among themselves. In both cases, the administration of sender/receiver channel pairs is drastically reduced, as are the number and complexity of remote queue definitions and other, more typical WebSphere MQ connectivity requirements.

In spite of the inherent availability and workload balancing that MQ Clusters provide, there is nevertheless the opportunity for system crashes and other unplanned outages. When such events occur, applications reading the messages from the clustered queues hosted by the failed machine are no longer available for processing. Queue managers that were connected to the failed machine likewise are no longer able to communicate with it.

Because the reliance on standard interprocess communication heartbeat and keep-alive features, which may or may not adequately stop a running channel to a failed system (and hence, its queue manager), messages can inadvertently be left uncommitted in a channel until the channel is activated

again. This recovery can take a while, and thus the messages in transit will become unavailable for a time.

For these reasons, although an MQ cluster is a valuable asset, it is not the most robust implementation that the industry has to offer.

OpenVMS Clusters and WebSphere MQ Failover

In OpenVMS, in spite of the robustness of the cluster locking mechanism, the WebSphere MQ product requires that a given queue manager can run on only one node of an OpenVMS Cluster at a time. (A typical nonclustered server environment has only one instance of a given queue manager running as well on a single server.) However, as with other WebSphere MQ implementations, each system can run one or more *different* queue managers.

This feature becomes especially valuable with multiple queue managers running on separate nodes in the cluster, all being members of a WebSphere MQ Cluster. In such a configuration, OpenVMS clustered applications can read messages being written to MQ cluster shared queues, hosted by queue managers distributed throughout the OpenVMS Cluster. This arrangement provides great processing power among applications that require shared memory or other resources, while maximizing availability to members of the MQ Cluster on other distributed systems, whether or not they are running OpenVMS.

An even greater development was deployed in WebSphere MQ version 5.3. This new feature, called failover sets, enables a queue manager to be restarted automatically on another OpenVMS Cluster node if a node failure occurs. Although the feature can be used with or without MQ Clusters, it certainly enhances the overall efficiency and power of the solution.

A failover set is a single queue manager and the nodes across which it can run. There can be multiple failover sets on a given OpenVMS Cluster or standalone system, but each failover set can control only a single queue manager. The "set" component of the names refers to the set of OpenVMS Cluster nodes on which the queue manager can run. However, one or more failover sets can be configured into an MQ Cluster, which, in my opinion, provides the most robust configuration.

There are several important issues with regard to MQ 5.3 on OpenVMS that must be considered in advance.

Failover Set Details

Failover sets are intended to permit a given queue manager to be restarted on another node in the OpenVMS Cluster if a failure occurs on the server on which it is running. From a data standpoint, this is accomplished by virtue of the inherent clusterwide access afforded by the lock manager in a cluster. From a network connectivity standpoint, it is accomplished by incorporating alias IP addressing on a given internet interface. (For more information, see the manuals for various TCP/IP product variants.)

Each failover set has a failover *monitor* that keeps track of the status of the failover set to which it is assigned. As long as this monitor process is active, only the provided FAILOVER commands can be used to stop, start, or inquire about the status of the queue manager failover set. (The use of standard MQ commands like `strmqm`, `endmqm`, and so on, is not permitted.)

After a failure is detected and a queue manager is moved to another node in the cluster, the queue manager is started, and the designated IP address for the failover set (which is specifically not the same as the IP address of the node/server itself) is added as an alias to one of the network interfaces on the machine using standard IP `ifconfig` commands.

Migrating the IP address for the failover set permits reconnection by distributed MQ clients and servers using a fixed IP address to the same queue manager channels, on a different node in the cluster. Distributed MQ connections that were active experience only a channel interruption that clears either by manual restart or automatic retry.

A failover set consists of an initialization file and three associated command procedures (START_QM.COM, TIDY_QM.COM, and END_QM.COM). To establish a failover set, the template FAILOVER.TEMPLATE in MQS_EXAMPLES: is copied to FAILOVER.INI and is then modified for the required changes. Required logicals are defined and the other template procedures are customized.

After the systemwide MQS_STARTUP.COM procedure runs, the `runmqfm` command is issued to start the failover monitor on the correct node for the designated failover set. Then the `failover` command starts the queue manager, the associated listener, and any dependent applications (runs the START_QM.COM procedure).

Changes to any of the template files coded prior to starting the failover monitor requires the monitor to be restarted. Thus, the failover monitor also monitors the integrity of the failover set command environment.

A Simple Example

The following example will illustrate the failover operation using a two node OpenVMS cluster supporting two WMQ queue managers, with a distributed HP-UX queue manager in the same WMQ cluster.

In Figure 1, Node A and Node B are OpenVMS Cluster Members. Node A and B each support a distinct queue manager failover set. Each set is a member of an MQ Cluster (MQ_CL), which includes an HP-UX server, running queue manager QM_HP.

Applications running on QM_HP put messages to the cluster queues, C_Q, hosted by QM_1 and QM_2, in a round-robin manner. The naming convention used for queue managers and node names is specifically decoupled to reduce unnecessary linkage between a failover set and the node on which it runs.

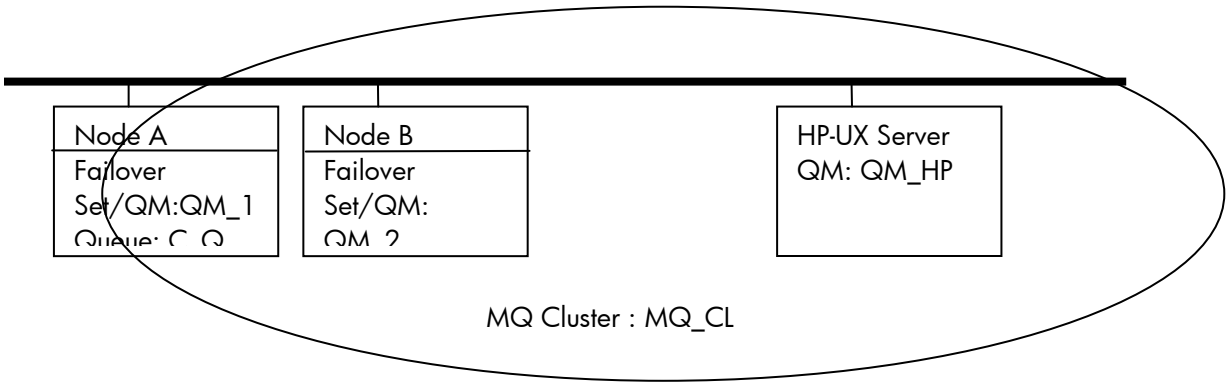


Figure 1 – Normal Configuration

In Figure 2, Node A has failed or is otherwise unavailable. The failover monitor running on Node B determines that failover set QM_1 is no longer running. Because the failover set is configured to run on Node A at priority 1 and on Node B at priority 2, the queue manager QM_1 now starts on Node B using the START_QM procedure specified for QM_1. This procedure ensures that the proper IP address alias is established so that listener programs respond to channel start request appropriately. Thus, both OpenVMS WebSphere MQ Cluster queue managers are now running on Node B.

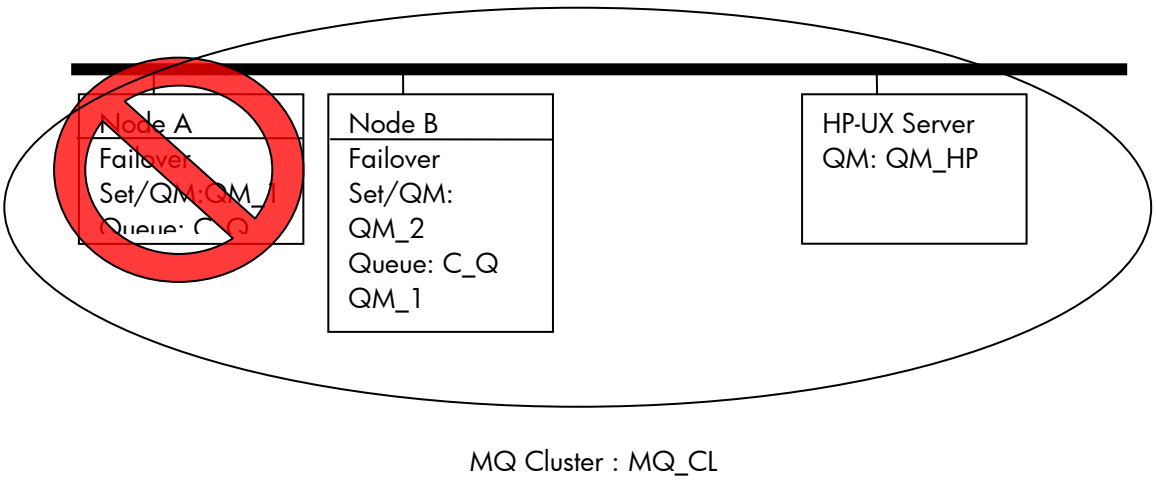


Figure 2 – Failover Configuration

The following aspects of WebSphere MQ 5.3 require more explanation than what is provided in the product documentation:

It is critical that the latest WebSphere MQ 5.3 patch be applied.

In an OpenVMS Cluster, it is critical that a common SYSUAF and rightslist be used across all nodes of the cluster. If this is not possible, the MQM identifier and user name, the MQS_SERVER user name, and their respective UICs must be identical across all systems. This commonality must be in place before any WebSphere MQ is installed on *another node* in the cluster. If this is not done, the new installation will not have the ownership it requires in the common MQS_ROOT:[MQM] directory.

It is recommended to use `runmqlsr` to initiate channel listening programs, specifically in the START_QM.COM procedure by, uncommenting the provided commands after the queue manager starts. This creates a <queue-manager_LS> process. If this feature is used, the `endmqlsr` command must also be uncommented in the END_QM.COM procedure. In version 5.3, MQ on OpenVMS has adopted the same syntax as other distributed platforms, namely the use of the ampersand (&) to initiate the listener processes as detached processes. When IBM releases WebSphere MQ Version 6.0, listener objects will likely be supported. At that time, the starting and stopping of listeners will be coordinated with the corresponding actions by the queue manager, and use of these commands in the start and end scripts will no longer be necessary.

When configuring the port used for multiple queue managers (failover sets) running in a single OpenVMS Cluster environment, it is important to assign different ports to each failover set, especially if the same LAN is used for all such connections. During a failover condition, when the potential exists for having more than one queue manager running on a single system, different port assignments must be used to ensure separation between receiver channels using IP aliases to the same interface.

Summary

In a distributed WebSphere MQ clustered environment, one might be led to believe that the round-robin and failover capability inherent in that configuration is adequate for high-availability solutions. However, if a processing latency exists for messages delivered to a queue on a node in the OpenVMS Cluster that subsequently fails, those messages are unavailable until the node rejoins the cluster. With failover sets, such messages can be accessible as soon as the queue manager on the failed server is restarted on another node in the OpenVMS Cluster. In such an arrangement, the application processes that utilize such messages must be cluster aware and able to migrate to the node where the queue manager is being restarted.

For this reason, it is best to start such applications from within the START_QM.COM procedure itself (and end them in the accompanying END_QM.COM).

Again, much more detailed concept descriptions and configuration guides are available in the *WebSphere MQ for HP OpenVMS System Administration Guide, Version 5 Release 3*, but it is my hope that this paper is a useful starting point for those new to the failover capabilities of WebSphere MQ on OpenVMS.

For more information

John Edelmann can be reached via email at john.edelmann@hp.com.

For additional information, see the following website and select the Platform Specific section, for HP OpenVMS:

http://www-306.ibm.com/software/integration/wmq/library/library6x.html

F$GETQUI to the Rescue

Bruce Claremont, Senior Consultant

Overview

The DCL lexical F$GETQUI is a powerful function that can be a bit confusing to deploy. This article demonstrates its usage via a useful DCL procedure designed to selectively delete entries from a queue. This article presumes a working knowledge of OpenVMS and DCL on the part of the reader.

F$GETQUI Lexical Function

Lexical functions provide easy access to a great deal of system information from the DCL command level. One of the most potent of these is the F$GETQUI function. It provides a way to interrogate the system for queue information, including batch entries, print entries, queue characteristics, and more.

Unlike most lexicals, which need only be called once to obtain information, the F$GETQUI function is iterative. Initial calls create a context that defines the information subsequent calls provide. Using the function may be a bit daunting at first, but mastering it provides a powerful tool for queue management and useful insights into the related system service calls that can be made from within programs.

Deleting Queue Entries

Anyone that has spent much time managing an OpenVMS system has run into the situation where they need to delete a large number of entries from an active queue. DCL provides the DELETE /ENTRY command to accomplish this. Unfortunately, DCL does not provide a means to specify a range of entries. Thus, to delete the following jobs,

```
Generic printer queue CPS_ANSI
Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
     13  MIGRPG_LIST     CLAREMONT        42  Pending
     21  KIT             CLAREMONT        19  Pending
     23  CVTFILE         CLAREMONT        19  Pending
     27  CBL_RPT         CLAREMONT        19  Pending
```

one needs to enter the following command:

```
DELETE /ENTRY=(13,21,23,27)
```

This isn't too bad when there are only a few jobs, but when a print queue has been down for several hours or a batch job has failed repeatedly and had its entries retained using the /RETAIN=ERROR option, the list can get long. I've know managers that have deleted and recreated queues to clear them in these situations. Using F$GETQUI we can avoid such drastic measures, create a procedure that automates the process, and learn how F$GETQUI works.

CLEARQ

CLEARQ.COM is a procedure I created to automate clearing multiply entries from a queue. It allows the user to select a queue, then displays each entry on that queue in turn, and asks the user if they would like to delete the entry. It provides information on each entry to help the user determine if the job is a candidate for removal.

CLEARQ relies on F$GETQUI. The procedure is comprised of two main components, CLEARQ.COM and SELECTQ.COM. Both procedures appear at the end of this article. I will discuss CLEARQ in detail. SELECTQ also takes advantage of F$GETQUI to display a list of available queues to the user. It is similar in concept to CLEARQ. Determining the details of how it works is left as an exercise for the reader.

Procedure Initialization

```
$ ON ERROR THEN GOTO ERROR_TRAP           !Error trap.
$ ON CONTROL_Y THEN GOTO ERROR_TRAP       !User abort trap.
$ STATUS = 1                              !Default exit status value.
$ !Set terminal & process attributes.
$ @SYS$UTILITIES:SET_ATTRIBUTES.COM "''F$ENVIRONMENT("PROCEDURE")'"
$!
$ P1 = F$EDIT(P1, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
$ IF F$EXTRACT(0, 3, P1) .EQS. "ALL" THEN P1 = "ALL_JOBS"
$ P2 = F$EDIT(P2, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
```

```
$ ERRMSG = ""
$ FOUND = ""
$ _QUEUE = ""
```

Figure 1: Procedure initialization code

The block of code in *Figure 1* initializes the environment. It does the following:

- Creates and initializes variables.
- Edits optional user parameters P1 and P2.

The procedure SET_ATTRIBUTES.COM and my philosophy on code development are discussed in detail in the article *Simplification Thru Symbols*, available in OpenVMS Technical Journal V10.

Queue Selection

```
$! Obtain or validate queue name.  Validated queue name is returned in
$! global symbol _QUEUE.
$!
$ @SYS$UTILITIES:SELECTQ.COM 'P2'
$ IF $STATUS .EQ. 3 THEN GOTO CANCEL_PROCEDURE
$ IF _QUEUE .EQS. ""
$   THEN
$         SAY "Queue not found: ", _QUEUE
$         GOTO ERROR_TRAP
$  ENDIF
```

Figure 2: Queue selection call.

Having initialized our variables, we need to select the queue upon which we wish to act. The SELECTQ.COM procedure fulfills two roles in this regard. If the user provides a queue name via P2, SELECTQ verifies the entry is a valid, available queue. If P2 is invalid or the user does not provide a queue name, SELECTQ presents a list of available queues from which the user can choose.

SELECTQ takes full advantage of F$GETQUI functionality. The procedure appears in its entirety at the end of this article. The lessons imparted in our discussion of CLEARQ will help you understand how SELECTQ works.

Tip: SELECTQ passes back the queue name in the global symbol _QUEUE. If a local symbol called _QUEUE is also defined, it will take precedence and interfere with the operation of SELECTQ and CLEARQ (guess how I know this). This is one of those simple, subtle problems that can be difficult to identify. Adding code to check for and address conflicting local symbols would resolve the issue. Sounds like material for another article.

Setting Context

```
$!      Cancel any outstanding wildcard contexts for F$GETQUI service.
$!
$ TEMP = F$GETQUI("CANCEL_OPERATION")
```

Figure 3: Clearing context.

Setting context is key to using F$GETQUI. Before context is set, the CANCEL_OPERATIONS call ensures context is cleared from any prior F$GETQUI calls.

Setting context is not required for all F$GETQUI calls. For example, JOB_ENTRY calls do not require context to be set. However, for CLEARQ, we need context established to ensure subsequent F$GETQUI calls return information specific to the queue selected.

```
$ IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_CLOSED", _QUEUE)
$   THEN
$     QSTATUS = "Closed"
$   ELSE
```

```
$     IF   F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_RESETTING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STALLED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_UNAVAILABLE", _QUEUE)
$        THEN
$          QSTATUS = "Offline"
$        ELSE
$          QSTATUS = "Online"
$        ENDIF
$  ENDIF
$!
$!      Context for following F$GETQUI commands is set here!!!
$!
$ QDESC = F$GETQUI("DISPLAY_QUEUE", "QUEUE_DESCRIPTION", _QUEUE,
"WILDCARD")
```

Figure 4: Setting context.

The DISPLAY_QUEUE calls in the IF-THEN-ELSE block do not establish context. That means a subsequent DISPLAY_JOB or DISPLAY_FILE call may not display information from the desired queue or may issue an error. The purpose of these calls is to establish the status of the queue.

The DISPLAY_QUEUE/WILDCARD combination at the end of *Figure 4* establishes context. Subsequent F$GETQUI calls will return information within the context of the queue we have selected, which is identified by _QUEUE.

```
$ JOB_LOOP:
$!
$ ENTRY = F$GETQUI("DISPLAY_JOB", "ENTRY_NUMBER",, "''P1'")
$ IF ENTRY .EQS. "" THEN GOTO END_PROCEDURE
$ FOUND = 1
```

Figure 5: Listing entries.

Having established context, the procedure will use it to return information about each entry present on the selected queue. The DISPLAY_JOB/ENTRY_NUMBER call shown in *Figure 5* returns the entry number of the first job listed on the queue. Each pass through JOB_LOOP will pick up the next entry on the queue. Note there is no need to specify the queue name at this point.

The P1 parameter can be blank or specify the parameter ALL_JOBS. ALL_JOBS is only relevant to accounts with OPER privilege. By default, F$GETQUI will only return information on entries belonging to the UIC of the calling process. This is a handy feature. It allows unprivileged users to use CLEARQ without danger of them deleting entries that do not belong to them. If a privileged user runs CLEARQ without specifying ALL_JOBS, they too only see the entries that belong to them. Specifying ALL_JOBS allows a privileged user to see all entries on the queue.

```
$ JOB_STATUS = ""
$ IF F$GETQUI("DISPLAY_JOB", "JOB_ABORTING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Aborting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_EXECUTING",, "FREEZE_CONTEXT") THEN
-
       JOB_STATUS = JOB_STATUS + "Executing, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_HOLDING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Holding, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_INACCESSIBLE",, "FREEZE_CONTEXT")
THEN -
       JOB_STATUS = JOB_STATUS + "Inaccessible, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_PENDING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Pending, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_REFUSED",, "FREEZE_CONTEXT") THEN –
```

```
       JOB_STATUS = JOB_STATUS + "Refused, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_RETAINED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Retained, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STALLED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Stalled, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STARTING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Starting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_SUSPENDED",, "FREEZE_CONTEXT") THEN
-
       JOB_STATUS = JOB_STATUS + "Suspended, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_TIMED_RELEASE",, "FREEZE_CONTEXT")
THEN -
       JOB_STATUS = JOB_STATUS + "Timed Release, "
$!      !Get rid of trailing comma.
$ JOB_STATUS = F$EXTRACT(0, F$LENGTH(JOB_STATUS) - 2, JOB_STATUS)
$!
$ SAY _ESC, "[3;1f"
$ SAY "  Job:       ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "JOB_NAME",,"FREEZE_CONTEXT"), _CANCEL
$ SAY "  Entry:     ", _CEOL, _BOLD, ENTRY, _CANCEL
$ SAY "  Owner:     ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "USERNAME",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  Submitted: ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "SUBMISSION_TIME",, "FREEZE_CONTEXT"),
_CANCEL
$ SAY "  File:      ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_FILE", "FILE_SPECIFICATION",, "FREEZE_CONTEXT"),
_CANCEL
$ SAY "  Status:    ", _CEOL, _BOLD, JOB_STATUS, _CANCEL
```

Figure 6: Obtaining entry (job) information.

Figure 6 demonstrates F$GETQUI DISPLAY_JOB and DISPLAY_FILE calls obtaining entry information that will be displayed to the user. The FREEZE_CONTEXT parameter is important. It prevents each DISPLAY call from jumping to the next entry on the queue. Note that the first DISPLAY call (*Figure 5*) does not freeze context. All subsequent DISPLAY calls must freeze context to keep F$GETQUI positioned on the same entry.

```
$ SELECTION = ""
$ RETRY_SELECTION:
$!
$ SAY _BOLD, _ESC, "[12;1f", ERRMSG, _CANCEL
$ SAY _ESC, "[11;1f", _CEOL              !Clear selection line.
$ SAY _ESC, "[11;1fRemove Entry (Y/N) [N]: ", _CANCEL, SELECTION, -
   ESC, "[11;29f", _BOLD, "<Ctrl^Z>", _CANCEL, " Quit"
$ ASK "''_ESC'[11;25f" SELECTION
$ SAY _ESC, "[12;1f", _CEOS       !Clear to end of screen
$ ERRMSG = ""                     !Clear error message.
$ SELECTION = F$EDIT(SELECTION, "COLLAPSE, UPCASE")
$ IF SELECTION  .EQS. "" THEN SELECTION = "N"
$ IF SELECTION  .NES. "N" .AND. SELECTION .NES. "Y"
$   THEN
$     ERRMSG = "Invalid selection (Y/N)"
$     GOTO RETRY_SELECTION
$  ENDIF
$ IF SELECTION  .EQS. "Y"
$   THEN
$     DELETE /ENTRY='ENTRY'
$!    ERRMSG = "Entry removed: ''ENTRY'"
$  ENDIF
$ GOTO JOB_LOOP
```

Figure 7: To delete or not to delete.

The code in *Figure 6* obtains and displays entry information to the user. The code in *Figure 7* asks the user if the entry should be removed. Once the user has provided a valid response, CLEARQ takes the appropriate action, then loops back to the DISPLAY_JOB call in *Figure 5* and starts the process over again. The lack of the FREEZE_CONTEXT parameter in the F$GETQUI call in *Figure 5* ensures the next entry in the queue is returned. This loop continues until no more entries are found or the user aborts the utility with a *<Ctrl^Z>*.

Conclusion

So there you have it, a simple example that demonstrates the harnessing of a very powerful lexical function, F$GETQUI. I have used this function in many interesting ways, such as:

- Obtaining and acting on queue and job information
- Controlling queues
- Logging job information for audits

Once you understand how to utilize F$GETQUI effectively, you too will find myriad applications for it.

Code

CLEAR1.COM

```
$! CLEARQ.COM
$!
$!      This procedure allows a user to clear entries from a queue.
The
$!      procedure presents queue entries one at a time and allows the
user
$!      to remove the entry from the queue via a Y/N query.
$!
$!      If a queue name is passed to the procedure, it is validated.
If no
$!      queue name is passed, it is prompted for.
$!
$!      On Entry:
$!       P1 - ALL_JOBS   (Optional: Valid for privileged user only)
$!                       (          Can be abbreviated to ALL.    )
```

```
$!       P2 - Queue name (Optional)
$!
$ ON ERROR THEN GOTO ERROR_TRAP           !Error trap.
$ ON CONTROL_Y THEN GOTO ERROR_TRAP       !User abort trap.
$ STATUS = 1                              !Default exit status value.
$ !Set terminal & process attributes.
$ @SYS$UTILITIES:SET_ATTRIBUTES.COM "''F$ENVIRONMENT("PROCEDURE")'"
$!
$ P1 = F$EDIT(P1, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
$ IF F$EXTRACT(0, 3, P1) .EQS. "ALL" THEN P1 = "ALL_JOBS"
$ P2 = F$EDIT(P2, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
$ ERRMSG = ""
$ FOUND = ""
$ _QUEUE = ""
$!
$! Obtain or validate queue name.  Validated queue name is returned in
$! global symbol _QUEUE.
$!
$ @SYS$UTILITIES:SELECTQ.COM 'P2'
$ IF $STATUS .EQ. 3 THEN GOTO CANCEL_PROCEDURE
$ IF _QUEUE .EQS. ""
$   THEN
$         SAY "Queue not found: ", _QUEUE
$         GOTO ERROR_TRAP
$  ENDIF
$!
$!      Cancel any outstanding wildcard contexts for F$GETQUI service.
$!
$ TEMP = F$GETQUI("CANCEL_OPERATION")
$!
$!      Get queue information and set context for obtaining job
information.
$!
$ IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_CLOSED", _QUEUE)
$   THEN
$     QSTATUS = "Closed"
$   ELSE
$     IF   F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_RESETTING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STALLED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_UNAVAILABLE", _QUEUE)
$        THEN
$          QSTATUS = "Offline"
$        ELSE
$          QSTATUS = "Online"
$        ENDIF
$  ENDIF
$!
$!      Context for following F$GETQUI commands is set here!!!
$!
$ QDESC = F$GETQUI("DISPLAY_QUEUE", "QUEUE_DESCRIPTION", _QUEUE,
"WILDCARD")
$!
$ SAY _BOLD, _REVERSE, _ESC, "[1;1fREMOVE ENTRIES UTILITY", _CANCEL, "
-
    (Provided by the wonderful people at MSI)"
$ SAY "Queue: ", _UNDERLINE, _QUEUE, _CANCEL, " <", QDESC, "> ",
QSTATUS
$!
$!      This section locates all of the entries on the specified queue
```

```
$!      and displays them one at a time.  The user is prompted to
remove
$!      each entry.
$!
$ JOB_LOOP:
$!
$ ENTRY = F$GETQUI("DISPLAY_JOB", "ENTRY_NUMBER",, "''P1'")
$ IF ENTRY .EQS. "" THEN GOTO END_PROCEDURE
$ FOUND = 1
$!
$ JOB_STATUS = ""
$ IF F$GETQUI("DISPLAY_JOB", "JOB_ABORTING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Aborting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_EXECUTING",, "FREEZE_CONTEXT") THEN
-
       JOB_STATUS = JOB_STATUS + "Executing, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_HOLDING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Holding, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_INACCESSIBLE",, "FREEZE_CONTEXT")
THEN -
       JOB_STATUS = JOB_STATUS + "Inaccessible, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_PENDING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Pending, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_REFUSED",, "FREEZE_CONTEXT") THEN –
       JOB_STATUS = JOB_STATUS + "Refused, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_RETAINED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Retained, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STALLED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Stalled, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STARTING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Starting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_SUSPENDED",, "FREEZE_CONTEXT") THEN
-
       JOB_STATUS = JOB_STATUS + "Suspended, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_TIMED_RELEASE",, "FREEZE_CONTEXT")
THEN -
       JOB_STATUS = JOB_STATUS + "Timed Release, "
$!      !Get rid of trailing comma.
$ JOB_STATUS = F$EXTRACT(0, F$LENGTH(JOB_STATUS) - 2, JOB_STATUS)
$!
$ SAY _ESC, "[3;1f"
$ SAY "  Job:       ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "JOB_NAME",,"FREEZE_CONTEXT"), _CANCEL
$ SAY "  Entry:     ", _CEOL, _BOLD, ENTRY, _CANCEL
$ SAY "  Owner:     ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "USERNAME",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  Submitted: ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "SUBMISSION_TIME",, "FREEZE_CONTEXT"),
_CANCEL
$ SAY "  File:      ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_FILE", "FILE_SPECIFICATION",, "FREEZE_CONTEXT"),
_CANCEL
$ SAY "  Status:    ", _CEOL, _BOLD, JOB_STATUS, _CANCEL
$!
$ SELECTION = ""
$ RETRY_SELECTION:
$!
$ SAY _BOLD, _ESC, "[12;1f", ERRMSG, _CANCEL
$ SAY _ESC, "[11;1f", _CEOL                !Clear selection line.
$ SAY _ESC, "[11;1fRemove Entry (Y/N) [N]: ", _CANCEL, SELECTION, -
   ESC, "[11;29f", _BOLD, "<Ctrl^Z>", _CANCEL, " Quit"
$ ASK "''_ESC'[11;25f" SELECTION
$ SAY _ESC, "[12;1f", _CEOS       !Clear to end of screen
```

```
$ ERRMSG = ""                       !Clear error message.
$ SELECTION = F$EDIT(SELECTION, "COLLAPSE, UPCASE")
$ IF SELECTION  .EQS. "" THEN SELECTION = "N"
$ IF SELECTION  .NES. "N" .AND. SELECTION .NES. "Y"
$   THEN
$     ERRMSG = "Invalid selection (Y/N)"
$     GOTO RETRY_SELECTION
$  ENDIF
$ IF SELECTION  .EQS. "Y"
$   THEN
$     DELETE /ENTRY='ENTRY'
$!    ERRMSG = "Entry removed: ''ENTRY'"
$  ENDIF
$ GOTO JOB_LOOP
$!
$ END_PROCEDURE:
$
$ SAY _ESC, "[1;1f", _CEOS
$ IF (.NOT. FOUND .AND. _QUEUE .NES. "") -
   THEN SAY "No entries found in queue ", _QUEUE
$ EXIT 'STATUS'
$!
$ ERROR_TRAP:
$
$ SAY _BELL
$ SAY "An error or <Ctrl^Y> has aborted this procedure."
$ CHECK_POINT:
$ IF F$MODE().NES."BATCH"
$   THEN
$         ASK "''_BELL'Enter 0 to exit the procedure: " CHK
$         IF CHK.NES."0" THEN GOTO CHECK_POINT
$  ENDIF
$!
$ CANCEL_PROCEDURE:
$
$ STATUS = 3
$ IF F$MODE().NES."BATCH" THEN SET TERMINAL/LINE_EDITING
$ GOTO END_PROCEDURE
```

SELECTQ.COM

```
$! SELECTQ.COM
$!-------------
$!      This procedure is used to allow users to select a queue.  The
$!      queue name is returned in the global symbol _QUEUE.
$!
$!      If a queue name is passed to the procedure, it is validated.
$!
$!      On Entry:
$!              P1 - Queue name (Optional)
$!
$       ON ERROR THEN GOTO ERROR_TRAP           !Error trap.
$       ON CONTROL_Y THEN GOTO ERROR_TRAP       !User abort trap.
$       STATUS = 1                              !Default exit status
value.
$!      !Set terminal & process attributes.
$       @SYS$UTILITIES:SET_ATTRIBUTES.COM
"''F$ENVIRONMENT("PROCEDURE")'"
$!
$       P1 = F$EDIT(P1, "COLLAPSE, UPCASE")
$       CNT = 1
$       DEFAULT = ""
```

```
$       ERRMSG = ""
$       IF F$TYPE(_QUEUE) .EQS. "" THEN _QUEUE == ""
$!
$!      Cancel any outstanding wildcard contexts for F$GETQUI service.
$!
$       TEMP = F$GETQUI("CANCEL_OPERATION")
$!
$!      If the user has not specified a queue name in P1, then display
$!      the menu heading information.
$!
$       IF P1 .EQS. ""
$         THEN
$               GOSUB HEADING
$               SAY _REVERSE, _BLINK, _ESC, -
                    "[12;4fGathering Queue Information - Please wait -
",-
                    F$TIME(), _CANCEL
$        ENDIF
$       NP1 = P1
$!
$ NAME_LOOP:
$!      This section locates all of the queues currently
$!      defined on the system and records their names in symbols.  If
the user
$!      entered a queue name in P1, it is validated here.
$!
$       QNAME'CNT' = -
          F$GETQUI("DISPLAY_QUEUE", "QUEUE_NAME", "*")
$       IF QNAME'CNT' .EQS. "" THEN GOTO GET_INFO
$       IF P1 .NES. ""                          !Validate passed name?
$         THEN                                  !Yes
$               IF P1 .EQS. QNAME'CNT'              !Name match?
$                 THEN                              !Yes
$                       GOSUB GET_QINFO
$                       IF QSTATUS'CNT' .NES. "Closed"  !Queue
available?
$                         THEN                        !Yes
$                               SELECTION = CNT       !Select it.
$                               GOTO SETQ
$                         ELSE                        !No
$                               P1 = ""               !Reset
variables and
$                               CNT = 0               !proceed as
though no
$                               TEMP = F$GETQUI("CANCEL_OPERATION")
$                        ENDIF                        !entry was
made.
$                ENDIF
$        ENDIF
$       CNT = CNT + 1
$       GOTO NAME_LOOP
$!
$ GET_INFO:
$!      If the procedure has reached this point with a value in P1,
then an
$!      invalid queue name was entered by the user or the specified
queue was
$!      closed.  The menu header and an appropriate error message are
$!      displayed.
$!
$       IF NP1 .NES. ""
$         THEN
$               GOSUB HEADING
```

```
$               IF P1 .NES. ""
$                 THEN
$                       ERRMSG = "Queue specified <''NP1> does not
exist"
$                 ELSE
$                       ERRMSG = "Queue specified <''NP1'>is closed"
$                ENDIF
$        ENDIF
$!
$!      If the first queue name picked up is null, then no queues
$!      exist.  A message to that effect is displayed and the
procedure
$!      exits.
$!
$       IF QNAME1 .EQS. ""
$         THEN
$               SAY ""
$               SAY _BELL, _BELL, _BELL, _BOLD, "No queues available
on ", -
                    "this system at this time", _CANCEL
$               GOTO CHECK_POINT
$        ENDIF
$!
$!      Get the description and current status of the queues found in
the
$!      previous section.
$!
$       TEMP = F$GETQUI("CANCEL_OPERATION")
$       CNT = 1
$!
$ INFO_LOOP:
$       IF QNAME'CNT' .EQS. "" THEN GOTO DISPLAY_QUEUES
$       GOSUB GET_QINFO
$       CNT = CNT + 1
$       GOTO INFO_LOOP
$!
$ DISPLAY_QUEUES:
$!      Display information collected on queus.
$!
$       CNT = 1
$       ROW = 4
$       SAY _ESC, "[", ROW, ";1f", _CEOS        !Clear remainder of
screen.
$!
$ DISPLAY_LOOP:
$       IF QNAME'CNT' .EQS. "" THEN GOTO GET_SELECTION
$       DCNT = F$FAO("!3SL", CNT)
$       SAY _ESC, "[", ROW, ";1f", _BOLD, DCNT, ". ", _CANCEL, -
            F$EXTRACT(0, 15, QNAME'CNT'), _ESC, "[", ROW, ";22f", -
            F$EXTRACT(0, 51, QDESC'CNT'), _ESC, "[", ROW, ";74f", -
            QSTATUS'CNT'
$       CNT = CNT + 1
$       ROW = ROW + 1
$!
$!      If there are more than 13 queues to display, pause the display
at
$!      the bottom of the screen and give the user the opportunity to
$!      Continue, Redisplay, make a Selection, or quit.
$!
$       IF ROW .LE. 22 THEN GOTO DISPLAY_LOOP
$       IF QNAME'CNT' .EQS. "" THEN GOTO GET_SELECTION
$!
$!      Prompt for queue name and then verify that it was correctly
```

```
entered.
$!
$ GET_SELECTION:
$       SELECTION = ""
$!$ RETRY_SELECTION:
$       SAY _BOLD, _ESC, "[24;1f", ERRMSG, _CANCEL
$       SAY _ESC, "[23;1f", _CEOL               !Clear remainder of
line.
$       SAY "''_BLINK'''_ESC'[23;1fSelection>''_CANCEL' ", SELECTION,
-
            _ESC, "[23;20f", _BOLD, "<Enter>", _CANCEL, " More, ", -
            _BOLD, "<R>", _CANCEL, " Redisplay, ", -
            _BOLD, "<Ctrl^Z>", _CANCEL, " Quit"
$       ASK "''_ESC'[23;12f" SELECTION
$       SAY _ESC, "[24;1f", _CEOL       !Clear error line.
$       ERRMSG = ""                     !Clear error message.
$       SELECTION = F$EDIT(SELECTION, "COLLAPSE, UPCASE")
$       IF SELECTION  .EQS. "R"         !Redisplay queue list.
$         THEN
$              GOSUB HEADING
$              GOTO DISPLAY_QUEUES
$        ENDIF
$       IF SELECTION  .EQS. ""          !Display more queues if
available,
$         THEN                          !otherwise start list over.
$           IF QNAME'CNT' .EQS. "" THEN GOTO DISPLAY_QUEUES
$           ROW = 4
$           SAY _ESC, "[", ROW, ";1f", _CEOS        !Clear remainder
of screen.
$           GOTO DISPLAY_LOOP
$          ENDIF
$!
$!      Verify that a valid queue selection has been entered.  If not,
$!      display an error message and prompt for another selection.
$!
$       IF F$TYPE(QNAME'SELECTION') .NES. "STRING"
$         THEN
$              ERRMSG = "Invalid Entry"
$              GOTO RETRY_SELECTION
$        ENDIF
$       IF QSTATUS'SELECTION' .EQS. "Closed"
$         THEN
$              ERRMSG = "Queue is Closed"
$              GOTO RETRY_SELECTION
$        ENDIF
$!
$ SETQ:
$       _QUEUE == QNAME'SELECTION'
$!
$ END_PROCEDURE:
$       SAY _ESC, "[1;1f", _CEOS
$       EXIT 'STATUS'
$ ERROR_TRAP:
$       SAY _BELL
$       SAY "An error or <Ctrl^Y> has aborted this procedure."
$ CHECK_POINT:
$       IF F$MODE().NES."BATCH"
$         THEN
$              ASK "''_BELL'Enter 0 to exit the procedure: " CHK
$              IF CHK.NES."0" THEN GOTO CHECK_POINT
$        ENDIF
$ CANCEL_PROCEDURE:
$       STATUS = 3
```

```
$       IF F$MODE().NES."BATCH" THEN SET TERMINAL/LINE_EDITING
$       GOTO END_PROCEDURE
$!
$!*****************************************************************
**********
$!               S U B R O U T I N E   S E C T I O N
$!*****************************************************************
**********
$!
$ HEADING:
$!      Display selection menu heading and text.
$!
$       SAY _CLEAR, _BOLD, _ESC, "#3", _ESC, "[1;1f''_NODE' QUEUE
SELECTION MENU"
$       SAY               _ESC, "#4", _ESC, "[2;1f''_NODE' QUEUE
SELECTION MENU"
$       SAY _UNDERLINE, _ESC, "[3;34fNode: ", _NODE, _CANCEL
$       SAY _UNDERLINE, -
          _ESC, "[3;3f#", -
          _ESC, "[3;6fQueue Name     ", -
          _ESC, "[3;22fDescription
", -
          _ESC, "[3;73f Status ", _CANCEL
$       RETURN
$!
$!*****************************************************************
**********
$!
$ GET_QINFO:
$!      Get description and current status of queues.
$!      On Return:
$!               QSTATUS'CNT' = "Online"
$!                              "Offline"
$!                              "Closed"
$!                              "Unknown"
$!
$       IF F$TRNLNM(QNAME'CNT') .NES. ""     !Logical name check.
$         THEN
$           QDESC'CNT' = "Queue name also a logical name, unable to
get desc"
$           QSTATUS'CNT' = "Unknown"
$         ELSE
$           QDESC'CNT' = F$GETQUI("DISPLAY_QUEUE",
"QUEUE_DESCRIPTION", QNAME'CNT')
$           IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_CLOSED", QNAME'CNT')
$             THEN
$               QSTATUS'CNT' = CLOSED
$             ELSE
$               IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSED",
QNAME'CNT') -
                  .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSING",
QNAME'CNT') -
                  .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_RESETTING",
QNAME'CNT') -
                  .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STALLED",
QNAME'CNT') -
                  .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED",
QNAME'CNT') -
                  .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPING",
QNAME'CNT') -
                  .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_UNAVAILABLE",
QNAME'CNT')
$                 THEN
```

```
$                   QSTATUS'CNT' = "Offline"
$                 ELSE
$                   QSTATUS'CNT' = "Online"
$               ENDIF
$            ENDIF
$         ENDIF
$       RETURN
```

About the author: Mr. Bruce Claremont has been working with OpenVMS since 1983. Mr. Claremont has extensive programming, project management, and system management experience. He also likes mechanical toys. He founded Migration Specialties in 1992 and continues to deliver OpenVMS and application migration services along with hardware emulation solutions. More information about Migration Specialties products and services can be found at *www.MigrationSpecialties.com*.

For more information

More on DCL development:

- *Simplification Thru Symbols*
  http://h71000.www7.hp.com/openvms/journal/v10/simplificationthrusymbols.html
- *Simplifying Maintenance with DCL*
  http://h71000.www7.hp.com/openvms/journal/v9/simplifying_maintenance_with_dcl.html

For the latest issue of the OpenVMS Technical Journal, go to: http://www.hp.com/go/openvms/journal.

Thought for the day:
If bad code killed its creator, we would have better software applications.

How Flügger Modernized their OpenVMS Applications

Mogens Porsgaard, Modernization Project Leader, Flügger
Roger van Valen, Seagull Software

Overview

Flügger — one of Scandinavia's leading manufacturers, distributors, and dealers of paint products, wallpapers, paint brushes, and accessories — has depended on the reliability of the HP OpenVMS platform for more than 20 years. When Flügger moved to this operating system in 1986, their IT team developed a number of applications that continue to support the business: logistics, customer relationship management, vendor management, financials, and purchase and production applications. Flügger recently undertook a modernization effort for their OpenVMS applications. All applications run on OpenVMS Alpha server systems and are written in COBOL. They have an HP Integrity server in house for development use and expect to migrate to HP Integrity two to three years from now.

Founded in 1890, Flügger is headquartered in Rodovre, Denmark, and employs 1,400 people. Flügger is expanding its business outside Scandinavia into Poland, the Czech Republic, and China. Business management has outlined a number of requirements to support their global operations: business applications need to be internationalized while the host application remains in Danish; applications need to become more intuitive for users; and application screens, especially those used in the retail shops, need to be modernized with an updated look and feel.

Because the reliability of OpenVMS is fundamental to supporting Flügger's business, migrating to a different platform was not an option. But while experimenting with several modernization tools on the market, Flügger's IT team encountered various limitations that prevented them from enhancing their applications. As they were starting to rewrite their applications, they were introduced to Seagull Software through Benny Nielsen, the Danish ambassador in the OpenVMS Ambassadors program.

Going from "Green Screens" to GUI Panels to Leverage and Extend OpenVMS

We installed Seagull Software's LegaSuite GUI on a Windows PC and the client software on our Windows terminal servers. After a five-day training course, we started working on the migration, and the process has run smoothly ever since. Whenever we needed support from Seagull, they were there within a short period of time, and we have been very pleased with their help.

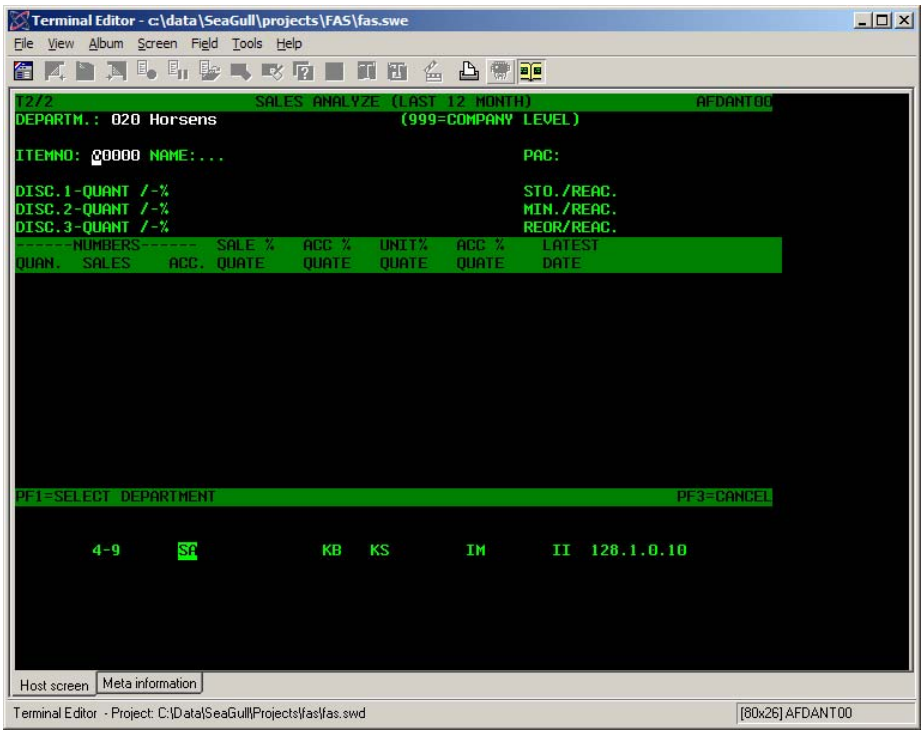
The setup is very simple. No software has to be installed on the OpenVMS system, which means that we don't have to implement any changes to our existing applications at all. This is a great advantage because we can implement the GUI version in a way that is transparent to users. When the IT staff releases an application, users can start it either from the GUI or from the "old- fashioned" version.

Even though it wasn't necessary, we chose to implement some minor changes in our old applications. The most essential of these changes focused on using a mouse in the GUI panel, so that the OpenVMS application would work essentially as a Windows-based application. Users had to be able to move from one field to another in the OpenVMS application by using the arrow keys. This

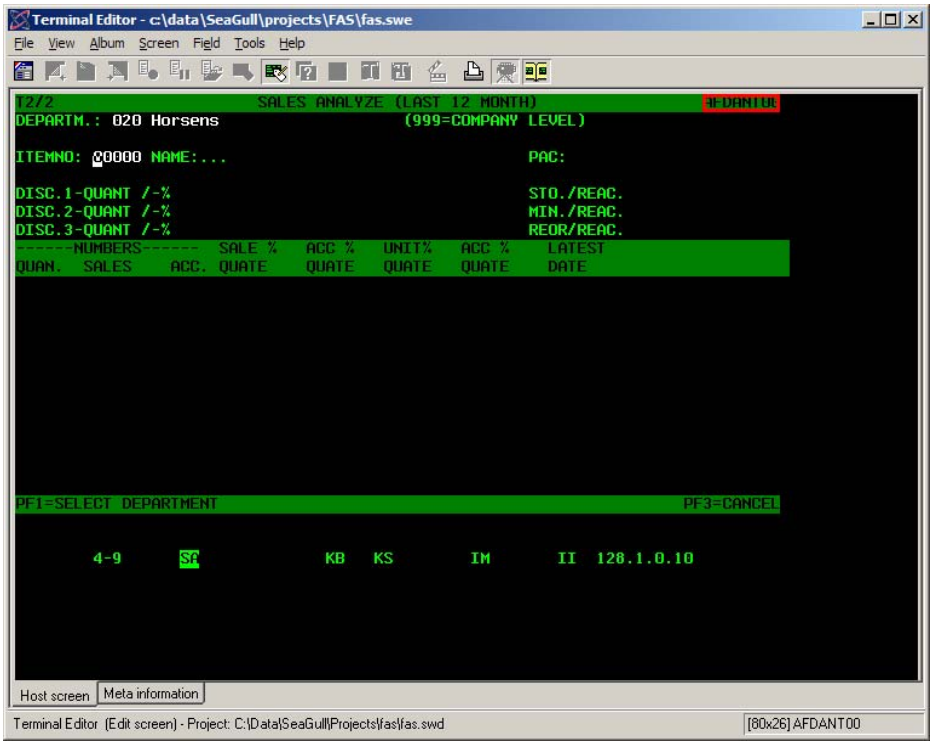
challenge was solved by implementing a panel script that sends the correct number of arrow-key sentences when moving from one field to another using the mouse.

We also took some time to define the look of the graphical panels. We defined different standards for the use of colors, fonts, the look of the buttons, and so on.

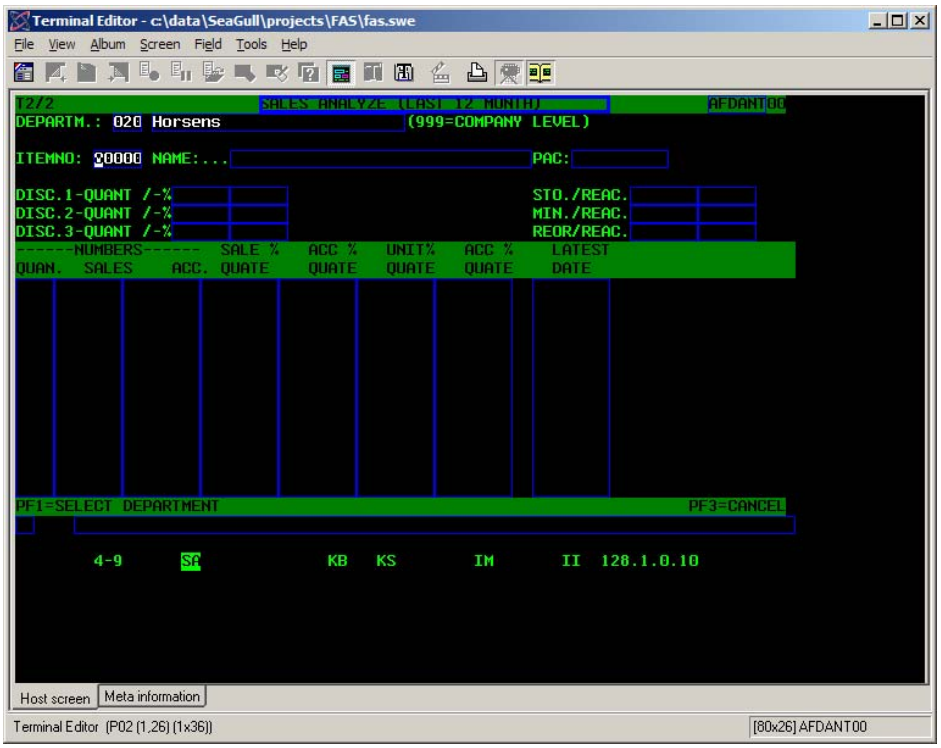
The following screen shows a typical OpenVMS application:



The screen has to be identified to the GUI system. This is done by marking a unique area on the screen, which is shown by the red frame in the upper right area of the following example:



When the application (or screen) is identified, the individual fields on the screen need to be defined, as shown in the blue frames in the following screen:

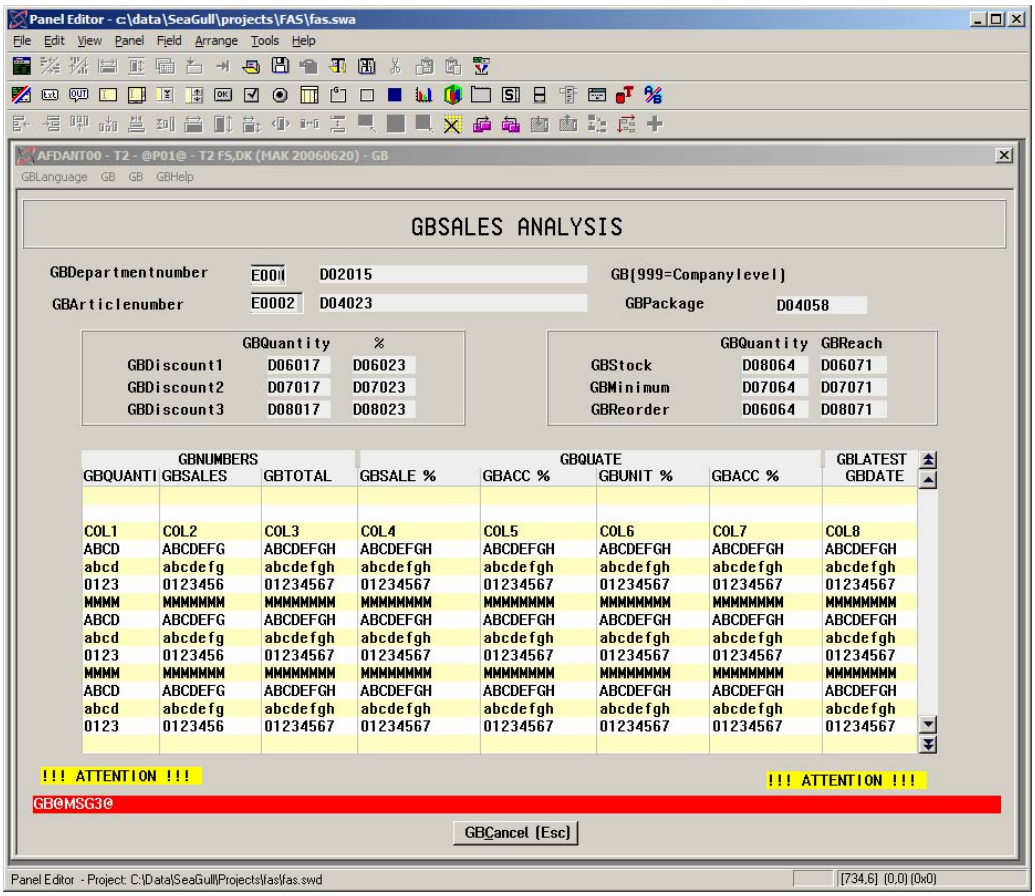


These fields are named uniquely, enabling them to be referenced on the GUI panel.

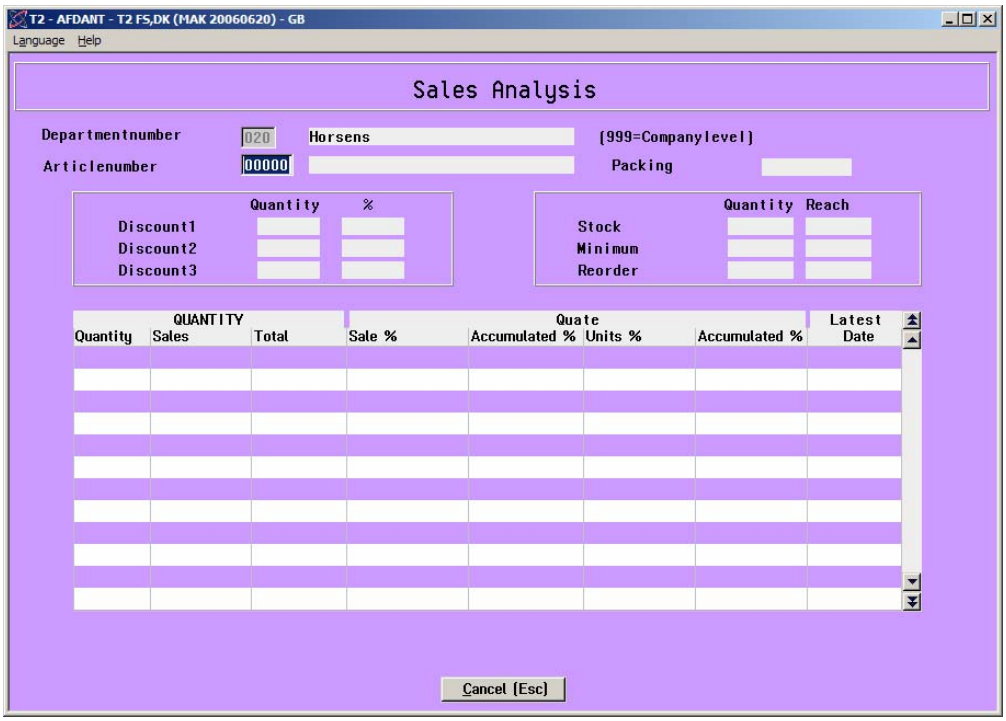
Then we created a panel in the GUI builder tool of LegaSuite GUI, including the definition of the leading text that explains the fields, as well as the definition of the named fields.

Leading text fields are created with a leading country suffix, enabling the text to be translated to the preferred language during execution of the application.

We had a wide choice of colors, frames, pictures, and other characteristics to include on the GUI panel screens. For example:



When the process was complete, the GUI version of our original OpenVMS application looked like this:

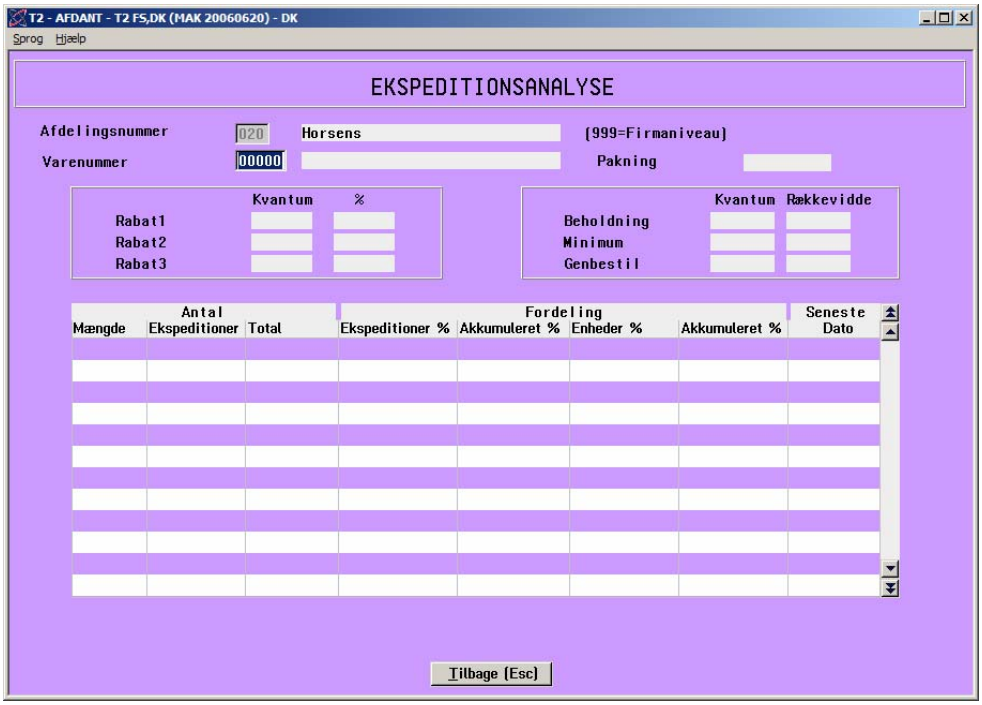


Thanks to the GUI panel menu, we greatly improved the functionality of the application.

We added some features that weren't possible to implement in the original OpenVMS application. For instance, we implemented an HTML document containing a short manual for each application, as well as the option to use different languages.

Each user can change the language as they prefer, any time during the execution. We have already translated, or are about to translate, the system into several different languages, including Danish (of course), Swedish, Norwegian, Polish, Icelandic, and English.

In Danish, the application looks like this:



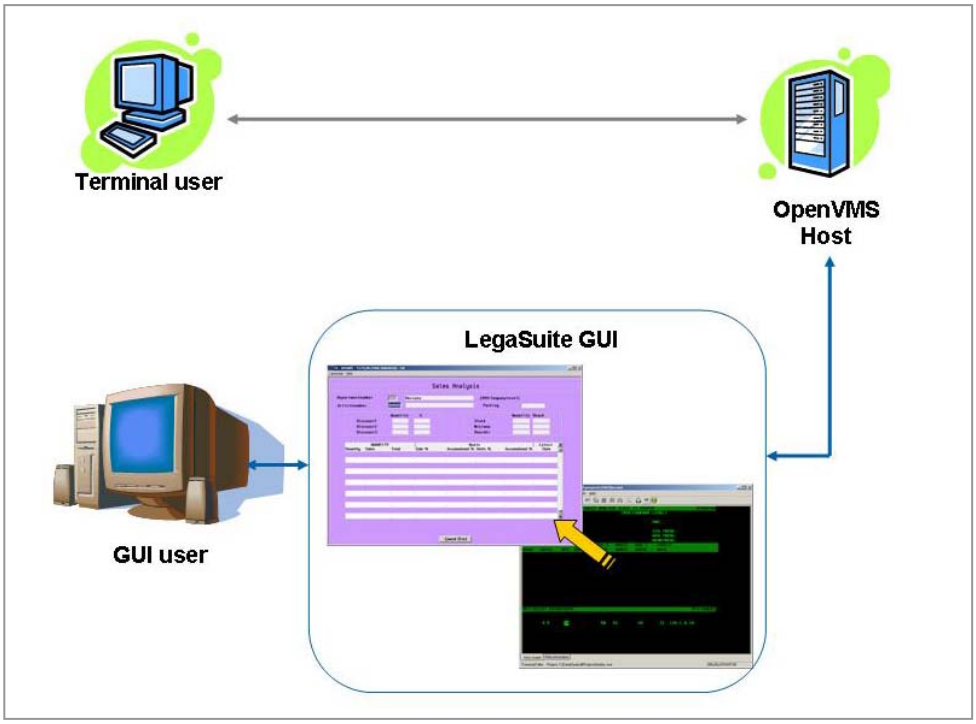
Technical Implementation

From the OpenVMS host perspective, LegaSuite GUI is a terminal emulator application. LegaSuite GUI connects though Telnet (or Secure Shell SSH) to the OpenVMS host and can work as a VT220, Wyse, or other emulator. Since the LegaSuite GUI application simply drives the host as a regular terminal, the host application does not require changes. However, if the host application is still maintained, changes can be incorporated to make certain functionality easier to implement using LegaSuite GUI.

LegaSuite GUI applies two different concepts to match a GUI layer on top of an OpenVMS application: screens and fields. A "LegaSuite GUI screen" can be defined as a transparent mask on top of a specific emulator view, and a "LegaSuite GUI field" is a container for a specific area on that view. Identification information (screen and field definition) is part of the data stream (for example, a VT220 data stream) received by LegaSuite GUI.

After the screen is uniquely identified, LegaSuite GUI can either show a panel or start some automated steps. A panel is a Windows-style representation to the user. This might resemble the screen (for example, having the same objects as the fields on the OpenVMS application), but it can also be built as, say, one Windows representation (panel) of many screens on the Host application.

The following figure shows how this process might occur:



OpenVMS applications can differ in key handling, but LegaSuite GUI is flexible in adapting its behavior to the host application. Two principal functions can help in this process:

- Metadata: The LegaSuite GUI emulator supports metadata settings, which provide a dynamic way of changing the emulator behavior based on user action (for example, through scripting)
- Scripting: Seagull provides its own script language that has syntax similar to Visual Basic.

A big difference in application handling between a Windows and an OpenVMS application is the level of control. In an OpenVMS application, the host is typically in control. Any key pressed by the user is handled by the host, and the host responses by text echoing, cursor movements, screen updates, and so on. When the user wants to supply text in a certain field, they typically have to use keys to move to that field and then type the contents.

In a Windows application, the user is typically in control. When the user clicks a certain control (such as a box in which to enter text), the control is given focus and the user can type the text. LegaSuite GUI supports a wide range of features such as events, emulator properties, and configuration settings and script functions to move easily through the host application and type text in fields.

Although LegaSuite GUI offers numerous features that help you to overlay the host application with GUI controls, building a GUI application is not only a matter of adding a visual interface on top of a "green screen" application. With LegaSuite GUI, you can redesign the application workflow, consolidate data from different applications, use other (desktop) applications (such as Word, Excel, and so on). Toward these ends, LegaSuite GUI scripting is an important feature.

A typical GUI project follows these steps:

1. Host application investigation. During this phase, answer questions such as the following: How can I tune the host application? Can I standardize specific behavior such as key handing, cursor positioning?
2. Screen and field design. At this point, you must build the "identification" for both LegaSuite GUI screens and LegaSuite GUI fields.
3. GUI building: You now build and customize your graphical panels and their related controls, and then map these controls to the fields designed previously.

4. Scripting: Scripting lets you control the behavior of your GUI application mapped against the OpenVMS application. Scripting also lets you add functionality not available in the host application, such as workflow improvements or integration with desktop applications like email, Word, and Excel.
5. Testing: Testing is an essential step of the development process and is performed in the LegaSuite GUI developer before the application is deployed. The panels are tested to verify the look and feel of the application as well as the data exchange process between the panel and host. The scripts supporting the functionality and navigation of the application are tested using the step-by-step execution capabilities that are built into the LegaSuite GUI developer.
6. Deploying: When deploying the GUI, you have a choice of thin-clients or rich-clients. For example, a Windows® client software or Windows browser plugin achieves the performance characteristics of traditional emulators with the benefits of Win32 code executing natively on the Windows platform.

For more information

To contact the authors, please send email to Mogens Porsgaard, mopo@flugger.com, or Roger van Valen, rvanvalen@seagullsoftware.com.

For more information about Seagull Software's OpenVMS solutions, please see the Seagull website at www.seagullsoftware.com, or send email to info@seagullsoftware.com.

RMS Collector for T4 and Friends

Gorazd Kikelj, HP Account Support Manager

Overview

This article provides a technique of expanding T4 and Friends with additional data collectors and with minimal impact to the core T4 procedures and utilizing services provided by them. The example described is uses a MONITOR RMS extension to monitor RMS file statistics.

Introduction

When analyzing a customer's system performance problem, it is common for some hot files to arise as key factors in the solution. Customers usually have no historical data on real file performance. Also, it is difficult to estimate from T4 cumulative data how much performance loss is caused by a specific file.

The RMS collector enables T4 to track RMS file-related statistics that are on line with other system performance data and ready for examination with TlViz. Figure 1 shows that, if relevant data is captured, how file performance is tied to system performance and also can be compared with application performance.

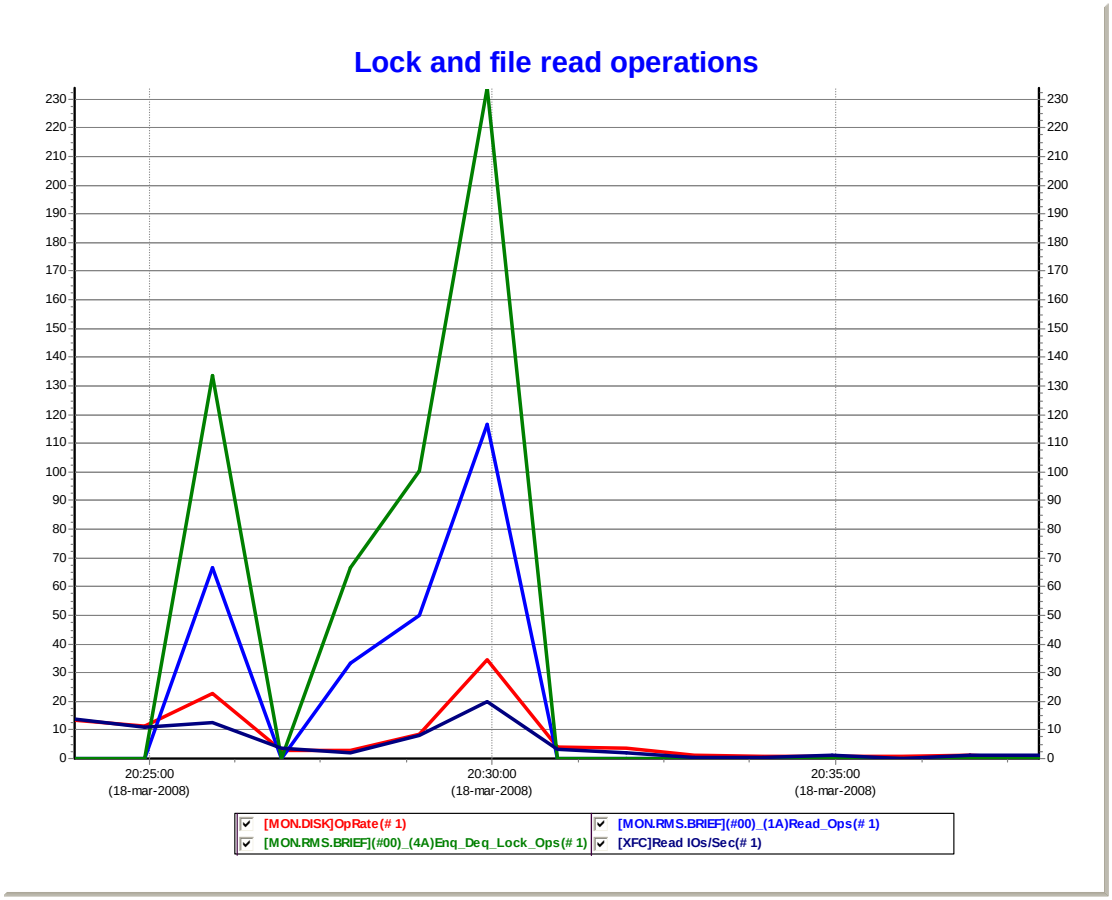


Figure 1 – Comparing file read and lock activity with system I/O operation rate

Trying to add a new collector into the T4 and Friends framework can be time consuming and requires modification of the core T4 command procedures. Also, some reverse engineering is needed to utilize facilities already in the framework. For this article, we used a technique that allows easy addition of new collectors in T4 and Friends framework by utilizing existing T4 data and collection process handling.

Work on this article is heavily based on Pat McConnell's article, "Adding a Friend to T4 and Friends", OpenVMS Technical Journal V4, 2004.

T4 and Friends version V4.2 is used in this article.

Expanding T4$COLLECT.COM to allow new Friends

The main procedure for data collection is the T4$SYS:T4$COLLECT.COM. It provides all necessary functions for dispatching data collection, processing collected performance data, maintaining the file repository, and recovering from errors. It is a complex procedure and is replaced with every new T4 upgrade. Consequently, every modification to the procedure must be reimplemented with any new release of T4 and Friends.

The T4$COLLECT.COM procedure provides many useful functions and symbols that can be used with Friends. The following table lists just some of the available symbols you can use:

Symbol	Description	Format
Today	Date when T4$COLLECT starts	DDMMMYYYY
Start_Time	Date and time when collection starts	DD-MMM-YYYY HH:MM
End_Time	Date and time when collection stops	DD-MMM-YYYY HH:MM
St_Et	Start and end times of data collection	HHMM_HHMM
This_Node	Node name of the T4 main process	Character string
This_Pid	PID of the main T4 process	Character string
Work_Dir	Working directory where data-collection files reside	Directory specification, by default T4$DATA
P6	Collection interval in seconds	Integer

For collected data to be useful, it must be gather in the same time period and at the same time intervals. You can specify these values by the Start_Time, End_Time and P6 symbols in the T4$COLLECT.COM command procedure. Those symbols must be passed to a Friend data-collection procedure and must be used in the collection process.

This article focuses on two sections of the T4$COLLECT.COM procedure that are of interest when adding a Friend: the Start_Data_Collection section and the Post_Process_the_Data section.

The Start_Data_Collection section starts and monitors all data collectors. If you want to add a Friend, add it to this section.

To minimize changes that are required after an upgrade, a user-maintainable procedure is called from this section. The T4$SYSTARTUP procedure includes a set of parameters that enable synchronization of the data-collection schedule and a working directory for data collection. By default, T4$SYSTARTUP.COM resides in the T4$SYS directory or must be pointed to by the logical name T4$SYSTARTUP. Here is an example of the Start_Data_Collection section of the T4$COLLECT.COM procedure.

```
$ Start_Data_Collection:
$! Skipped all lines before Network Monitor starts
$! ...
$!      Start user data collectors – Friends
$       User_Startup = F$Edit( F$Search( F$Parse( "T4$SYSTARTUP",-
                      "T4$SYS:.COM",,,"SYNTAX_ONLY")),"TRIM" )
$       If "''User_Startup'" .Nes. ""
$       Then
$        @'User_Startup "''Start_Time'" "''End_Time'" "''This_Pid'"
-
                           "" "''St_Et'" "''P6'" "''Work_Dir'"
$       EndIf
$
$!      Last, but not least spawn a Network Monitor data collection
event
```

The following parameters are passed to the user-maintainable procedure:

P1 – Start date and time for data collection
P2 – End date and time for data collection
P3 – PID of the master T4 process
P4 – Blank
P5 – Start and end times of data collection (used to form a file name)
P6 – Collection interval
P7 – Data-collection working directory, defined in T4$CONFIGURE.COM

In the Post_Process_the_Data section, all data collectors are already stopped and data is available for post-processing. A user-modifiable procedure is added at the end of standard post-processing procedures, right before consolidating data. For example:

```
$ post_Process_the_Data:
$! Skipped all lines before glue the .CSV files with T4$ApRc
$! ...
$!      User post processing data procedures
$       User_Shutdown = F$Edit( F$Search( F$Parse( "T4$SYSHUTDOWN",-
                        "T4$SYS:.COM",,,"SYNTAX_ONLY")),"TRIM" )
$       If "''User_Shutdown'" .Nes. ""
$       Then
$        @'User_Shutdown "''Start_Time'" "''End_Time'" "''This_Pid'"
-
                           "" "''St_Et'" "" "''Work_Dir'"
$       EndIf
$
$!      Last, but not least, we glue all the .CSV records together
$!      horizontally.  This is done using T4$APRC, which takes two
$!      parameters (P1 and P2), where each record in file P1 is
prefixed
$!      with a comma and then appended to the end of the
$! corresponding
```

The following parameters are passed to the user-maintainable procedure are:

P1 – Start date and time for data collection
P2 – End date and time for data collection
P3 – PID of the master T4 process
P4 – Blank
P5 – Start and end time of data collection (used to form a file name)
P6 – Blank
P7 – Data-collection working directory, defined in T4$CONFIGURE.COM

These two changes are the only ones needed for enabling new Friends to be added to T4.

The following is the syntax for log files to be automatically processed by T4$COLLECT.COM procedure:

```
<Work_Dir>T4_<nodename>_<Today>_<St_Et>_SUBP_<collector>.LOG
```

For example:

T4$DATA:T4_DS10L_26JUN2007_1500_1530_SUBP_R001.LOG

User-modifiable procedures

The only function of the T4$SYSTARTUP and T4$SYSHUTDOWN procedures is to call new T4 Friends (collectors) added by the user.

T4$SYSTARTUP example:

```
$       Set Verify ! T4$SYSTART.COM
$       Set Noon
$       Start_Time = P1
$       End_Time   = P2
$       This_Pid   = P3
$       St_Et      = P5
$       Interval   = P6
$       Work_Dir   = P7
$
$!      Call RMS Collector
$       Rms_Collector = F$Edit( F$Search( F$Parse( "T4$RMS_START",-
                        "T4$SYS:.COM",,,"SYNTAX_ONLY")),"TRIM" )
$       If "''Rms_Collector'" .Nes. ""
$       Then
$        @'Rms_Collector "''Start_Time'" "''End_Time'" "''This_Pid'"
-
                             "" "''St_Et'" "''Interval'" "''Work_Dir'"
$       EndIf
```

T4$SYSHUTDOWN example:

```
$       Set Verify ! T4$SYSHUTDOWN.COM
$       Set Noon
$       Start_Time = P1
$       End_Time   = P2
$       This_Pid   = P3
$       St_Et      = P5
$       Interval   = P6
$       Work_Dir   = P7
$
$!      Call RMS Collector Postprocessor
$       Rms_Collector_End = F$Edit( F$Search( F$Parse(
"T4$RMS_END",-
                       "T4$SYS:.COM",,,"SYNTAX_ONLY")),"TRIM" )
$       If "''Rms_Collector_End'" .Nes. ""
$       Then
$      @'Rms_Collector_End "''Start_Time'" "''End_Time'"
"''This_Pid'" -
                                  "" "''St_Et'" "" "''Work_Dir'"
```

RMS Collector

The RMS collector consists of three command files and a data file. Monitored files need to be entered into T4$SYS:RMS_FILES.DAT text file with your text editor of choice. Each record represents one monitored file specification. For each file specification, add 1 to the Prclm quota in SYSUAF record for the user account that will be used for the data collection.

T4$SYS:RMS_FILES.DAT example:

```
$ type T4$SYS:RMS_FILES.DAT

DISK$$DATA:[APPLICATION]MASTER_DATA.DAT
DISK$$DATA:[APPLICATION]INDEX_DATA.DAT
DISK$$USER:[HOTSPOT]HOTFILE.IDX
```

The T4$RMS_START.COM command procedure reads the T4$SYS:RMS_FILES.DAT file and spawns a subprocess for each record in a file. For example:

```
$! T4$RMS_START.COM
$! Parameters:
$!      P1 - Begin time
$!      P2 - End time
$!      P3 – Pid
$!      P4 -
$!      P5 - StopTime_EndTime
$!      P6 - Sample interval
$!      P7 - Working Directory
$!
$       Set Verify
$       On Error then goto Exit
$       Set Process/Priv=(All,NoBypass)
$       This_Node = F$GetSyi("NodeName")
$       Today = F$CvTime(P1,"ABSOLUTE","DATE")
$       If (F$Length(Today) .Eq. 10)
$       Then
$             Today = "0" + Today
$       EndIf
$
$       Today = Today - "-" - "-"
$       Work_Dir = P7
$       if ( F$Edit(F$Search(“T4$Sys:Rms_Files.dat”),”TRIM”) .Nes. “”
)
$       Then
$         File_Index = 1
$         Open/Read/Share t4_rms t4$sys:rms_files.dat
$ Loop:
$         Read/End=End_Read/Error=End_Read t4_rms rec
$         Idx = "''F$Fao("R!3XL",File_Index)'"
$         Spawn/NoSymbols/NoWait/Process="T4''P3'_''Idx'" -

/OutPut='Work_Dir'T4_'This_Node'_'Today'_'P5'_SUBP_'Idx'.log -
            @T4$Sys:T4$Rms_Mon "''P1'" "''P2'" "''Idx'" "''rec'"
"''P5'" –
            "''P6'" "''P7'"
$
$         File_Index = File_Index + 1
$         goto loop
```

In addition to the Prclm quota, another limiting factor is file-name syntax. The collector part of the file name (that is, the last part) consists of the letter R for RMS and a 3-byte hexadecimal-encoded running sequence from the RMS_FILES.DAT file. This sequence starts from 1 for the first record and progresses until the end of file. This imposes a limit of 4094 (%xFFF-1) files that can be monitored simultaneously.

The T4$RMS_MON.COM command procedure is the data collector. It uses a MONITOR RMS to monitor a file. Data are recorded into Work_Dir. Monitored files must have RMS statistics enabled via the SET FILE/STATISTIC command. To collect data about the RMS file provided in P4, define the Start_Time (P1) and End_Time (P2) parameters and the Sample Interval (P6) value. The process name and the collection file name use the File Index (P3) parameter to uniquely identify a collection. For example, T40009960_R001 is the process that collects statistics for the first file listed, and T4_DS10L_26JUN2007_1500_1530_R001.DAT is the name of the collection data file.

The following is the syntax for the data file name:

```
<Work_Dir>T4_<nodename>_<Today>_<St_Et>_<collector>.DAT
```

For example:

T4$DATA:T4_DS10L_26JUN2007_1500_1530_R001.DAT

This syntax enables T4$COLLECT.COM to automatically manage DAT files.

```
$! T4$RMS_MON.COM
$! Parameters:
$!      P1 - Begin time
$!      P2 - End time
$!      P3 - File Index
$!      P4 - File name
$!      P5 - StopTime_EndTime
$!      P6 - Sample interval
$!
$       Set Verify
$       Set Noon
$       Set Process/Priv=(All,NoBypass)
$       This_Node = F$GetSyi("NodeName")
$       Today = F$CvTime(P1,"ABSOLUTE","DATE")
$       If (F$Length(Today) .Eq. 10)
$       Then
$             Today = "0" + Today
$       EndIf
$
$       Today = Today - "-" - "-"
$
$       write sys$output "File ''P4'"
$       File_Name =
f$edit(f$search(f$parse(P4,,,,"SYNTAX_ONLY")),"TRIM")
$       if "''File_Name'".nes.""
$       then -
          Monitor/Record=T4_'This_Node'_'Today'_'P5'_'P3'.Dat-
          /Interval='P6 -
           /Flush_Interval='P6 -
           /Begin="''P1'" -
           /End="''P2'" -
           /Comment="''This_Node' ''File_Name'" -
           /NoDisplay
```

The T4$RMS_END.COM command procedure is a post-processing step for integrating recorded data into the file-specific comma-separated values (CSV) and composite CSV file. To create a CSV file, the T4$RMS_END.COM procedure utilizes the T4$Mon_Extract utility. For merging a CSV file into a composite CSV file, the procedures utilizes the T4$ApRc utility.

The CSV file syntax is the same as the .DAT file syntax:

```
<Work_Dir>T4_<nodename>_<Today>_<St_Et>_<collector>.CSV
```

For example:

T4$DATA:T4_DS10L_26JUN2007_1500_1530_R001.CSV

This syntax enables the T4$COLLECT.COM procedure to automatically manage CSV files.

```
$! T4$RMS_END.COM
$! Parameters:
$!      P1 - Begin time
$!      P2 - End time
$!      P3 – Pid
$!      P4 -
$!      P5 - StopTime_EndTime
$!      P6 - Sample interval
$!      P7 - Working Directory
$!
$       Set Verify
$       Set NoOn
$       Set Process/Priv=(All,NoBypass)
$       This_Node = F$GetSyi("NodeName")
$       Today = F$CvTime(P1,"ABSOLUTE","DATE")
$       If (F$Length(Today) .Eq. 10)
$       Then
$             Today = "0" + Today
$       EndIf
$
$       Today = Today - "-" - "-"
$       Work_Dir = P7
$       Set Command T4$Sys:T4$Mon_Extract
$       Set Command T4$Sys:T4$ApRc
$
$loop:
$      File_Name = -
       F$Edit(F$Search("T4_''This_Node'_''Today'_''P5'_R%%%.dat",-
              112233),"TRIM")
$       if File_Name .Eqs. "" then goto end_loop
$       File_Csv = F$Parse(File_Name,,,"NAME","SYNTAX_ONLY")
$       T4Extr 'File_Name' /Csv_File='File_Csv'.Csv –
                           /Class=(NoAll,RMS) –
                           /Format=Extended
$       T4Aprc 'Work_Dir''File_Csv'.Csv; -
               T4_'This_Node'_'Today'_'P5'_Comp.csv;
$       goto loop
```

The original CSV files are retained and combined with the composite CSV file in the archive ZIP file that T4$COLLECT provides at the end of the processing. This was done deliberately because there is no other way to find out what file was monitored once the RMS_FILES.DAT is changed.

Summary

This article shows that it is easy to add a Friend to T4 and Friends. This approach adds a user entry point to the T4 core execution process. Adding a simple, additional collector that utilizes existing utilities in OpenVMS and T4 is demonstrated.

For more information

- The following article is the standard reference for understanding the Timeline Collaboration concept, which has driven the development of T4 and Friends. This article provides the background necessary for the development of extensions to the default T4 implementation.

  "Timeline-Driven Collaboration with T4 and Friends: A Timesaving Approach to OpenVMS Performance," OpenVMS Technical Journal, Volume 3, February 2004.

- The following article describes how to integrate a Friend into T4:

  "Adding a Friend to T4 and Friends Incorporating BEA WebLogic Server 8.1 Performance Data, OpenVMS Technical Journal, Volume 4, June 2004.