OpenVMS Technical Journal V7

Faking it with OpenVMS Shareable Images

John Gillings, HP Customer Support Centre, Sydney Australia

Overview

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

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

Caveats

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

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

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

What is a Shareable Image?

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

How Shareable Images Work

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

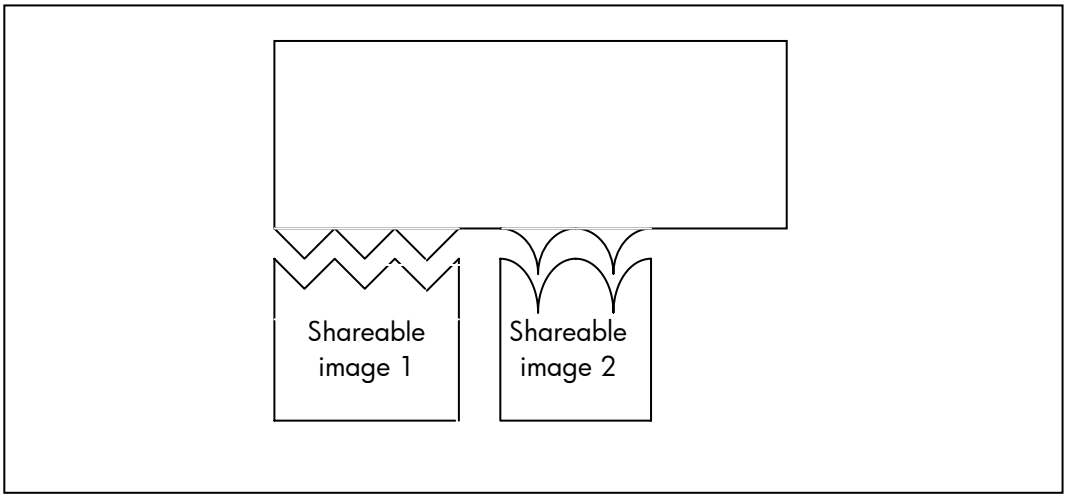


Figure 1 – Matching Interfaces

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

Symbol Vectors and GSMATCH

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

Image Activation

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

Upwards Compatibility

Using GSMATCH=LEQUAL, it is easy to create shareable images that support upwards compatibility. If changes are made to the implementation of a shareable image, the new image will be compatible

(plug interchangeable) with the old image provided there are no changes to the symbol vector. However if new entries are added to the end of the symbol vector, the new symbol vector will be longer. So images linked against the old image can execute against old or new shareable images, but images linked against the new shareable image cannot execute against the old version (because they might reference the new symbol vector entries, which do not exist in the older image's symbol vector). By convention, the minor ID of the GSMATCH values for the new image should be incremented, so that the GSMATCH=LEQUAL match criteria detects the difference in the images. For example:

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

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

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

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

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

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

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

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

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

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

Dynamic Activation

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

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

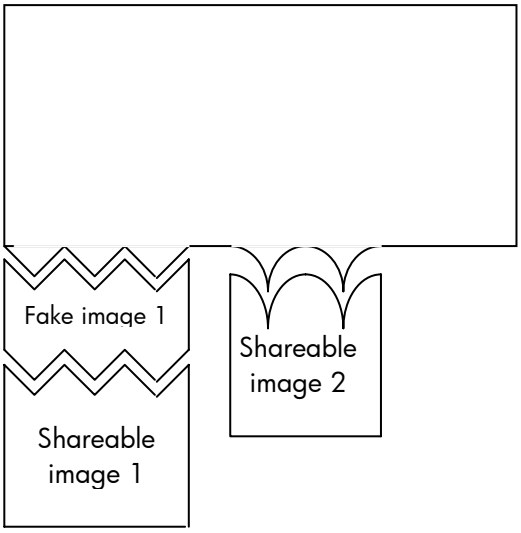


Figure 2 - Call Through a Fake Image

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

Will the Real Image Please Stand Up?

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

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

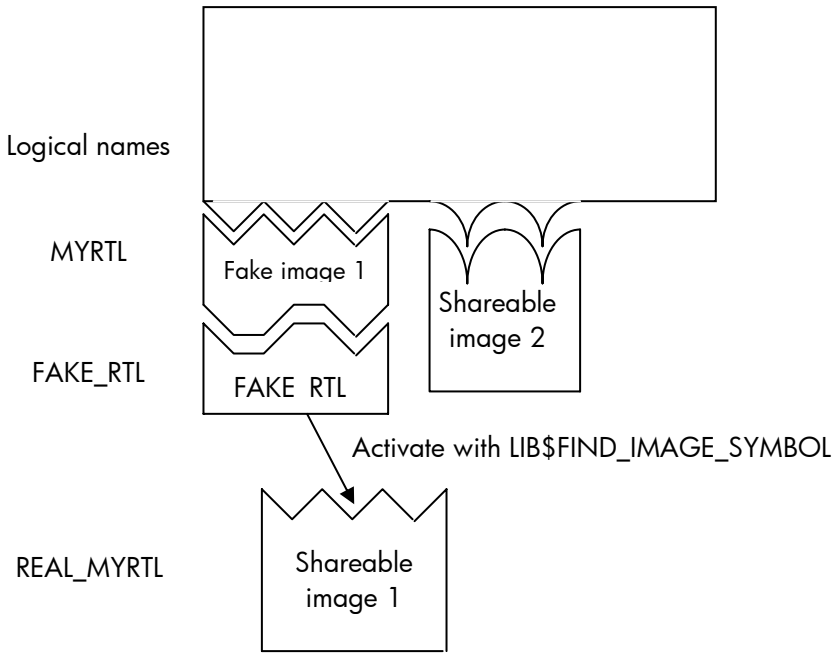


Figure 3 – FAKE_RTL Support Routines

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

Resolving Self References

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

But Why Would I Want to Do That?

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

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

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

How it's Done

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

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

Why MACRO?

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

Vector Entries

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

- SPARE

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

```
SYMBOL_VECTOR=(SPARE)
```

  No MACRO32 code is necessary:

- CONSTANT

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

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

  MACRO32 code is a global symbol definition:

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

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

- PROCEDURE

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

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

  MACRO32 code is a call to a macro:

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

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

- DATA CELL

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

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


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

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

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

Walkthrough FAKE_RTL.COM

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

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

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

`$ @FAKE_RTL`

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

`$ @FAKE_RTL SMGSHR`

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

`$ @FAKE_RTL SMGSHR COMPILE`

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

`$ @FAKE_RTL SMGSHR GENERATE DEBUG`

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

`$ @FAKE_RTL LIBRTL VECTOR FORCE`

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

FAKE_RTL.COM Phases in Detail

The following list describes FAKE_RTL.COM phases:

- **DEF - Define fake RTL environment**
  This phase defines the logical name FAKE_DIR to point to the directory containing FAKE_RTL routines and procedures. If the shareable image FAKE_RTL.EXE does not exist, it is created. FAKE_DIR is then scanned for files called REAL_<name>.EXE. A logical name is defined for each one found.
- **COPY - Make a copy of the original shareable image**
  The copy is called REAL_<imagename> and placed in the directory containing FAKE_RTL
- **VECTOR - Analyze the symbol vector**
  The input image is analyzed using ANALYZE/IMAGE and then filtered using SEARCH to extract the GSMATCH values and symbol vector entries. The GSMATCH information looks like this.

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

It would be transformed into a Linker options definition:

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

The three symbol vector entry types look like this:

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

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

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

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

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

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

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

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

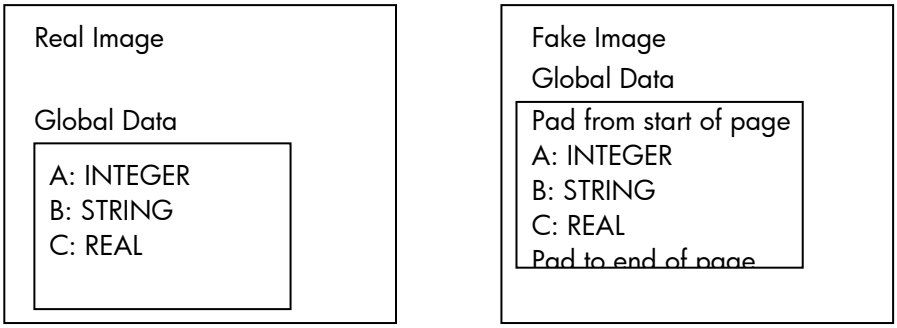
- **GEN - Generate MACRO code**
  This phase uses the ordered vector and data lists from the VECTOR phase to generate the symbol vector (as a linker options file) and MACRO32 code. For each vector entry we generate a symbol vector declaration and some code. Before writing the symbol vector declaration,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

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

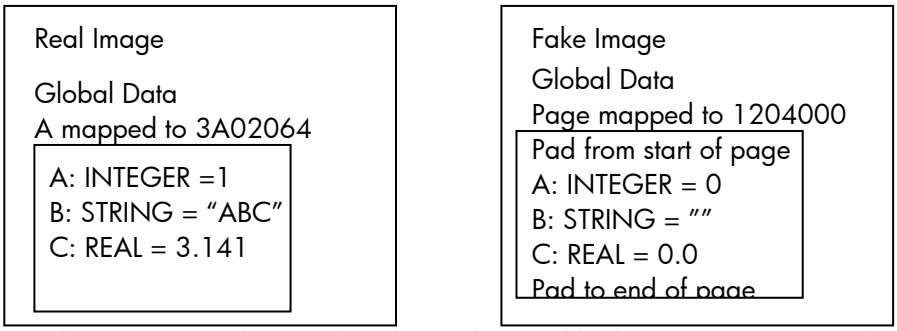
In some cases, the shareable image routine should be called using an OpenVMS VAX style JSB mechanism. In these cases, the FAKE_RTL routine FAKE_LOGJSB should be called instead. The OpenVMS naming convention is to name JSB routines with a trailing _Rn, where n is the highest register number used by the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

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

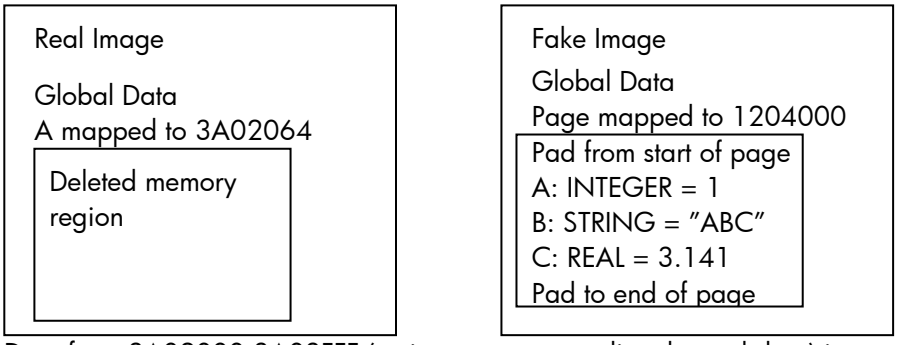
At run time, MapData calls the FAKE_RTL routine FAKE_MAP_DATA for each data block, given the name of the real shareable image, a generated global section name, the name of the first symbol in the block and the size, and start and end addresses of the block. FAKE_MAP_DATA first deletes the virtual memory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



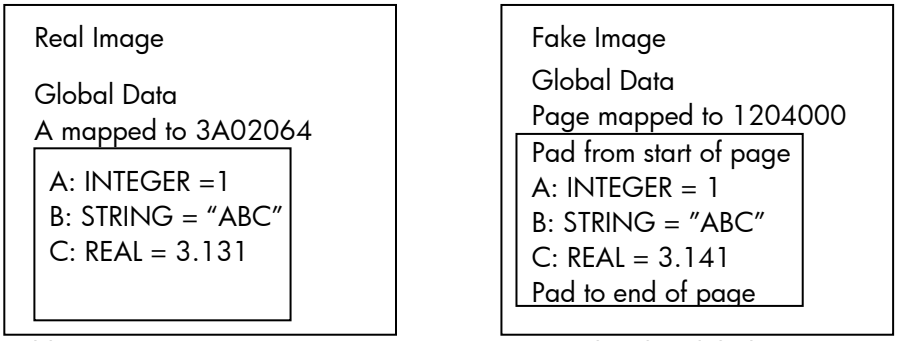
Creation time – Fake image has data declared in whole pages, aligned and allocated to match the data in the real image.



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



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



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

Figure 4 – Mapping Shared Data

COMPILE - Compile MACRO code

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

LINK - Link fake RTL

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

USE - Define logical names to use image

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

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

You can redefine this point to the original image:

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

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

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

and:

```
$ REAL_imagename==”$DEASSIGN imagename”
```

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

Does it work?

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

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

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

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

Case-sensitive symbol names can't be defined in MACRO32, so any shareable images that export lowercase symbols can cause trouble. For example, DECC$SHR exports each symbol in both uppercase and lowercase forms. Without some modification, this would generate duplicate symbols in MACRO32. The GEN phase identifies lowercase symbols and defines them using a prefix L_,

truncating the symbol name if necessary. Because there is no reason to LINK against the FAKE_imagename image, this does not cause any trouble for a caller activating the image normally. But an attempt to activate the image with LIB$FIND_IMAGE_SYMBOL using a lowercase, case-sensitive symbol name fails because the expected symbol does not exist.

Examples

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

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

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

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

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

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

```
$ mail

You have 1 new message.

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

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

Postscript – Port to OpenVMS Integrity Server

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

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

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

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

The FAKE_RTL command procedure is located at the following URL:

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

Appendix A: FAKE_RTL Routines

The following are routines exported by FAKE_RTL:

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

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

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

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

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

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

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

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

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

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

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

Appendix B: Logging Argument Lists

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

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

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

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

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

Figure 5 – Sample Logged Call and Argument List

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

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

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

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

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

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

  (Calls to other LIB$ routines omitted)

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

Figure 6 – Sample Logged Call Showing Nesting

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

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

Appendix C: Sample Run

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

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

```
$ @fake_rtl
Defining FAKE_RTL environment
%CREATE-I-CREATED, USER$TSC:[GILLINGS.FAKE_RTL]FAKE_RTL.MAR;1 created
%CREATE-I-CREATED, USER$TSC:[GILLINGS.FAKE_RTL]FAKE_RTL.OPT;1 created
Compiling FAKE_RTL.MAR
Linking FAKE_RTL.OBJ
$
$ @fake_rtl smgshr
%COPY-S-COPIED, SYS$COMMON:[SYSLIB]SMGSHR.EXE;1 copied to
USER$TSC:[GILLINGS.FAKE_RTL]REAL_SMGSHR.EXE;1 (593 blocks)
Generating symbol vector for SMGSHR
%ANALYZE-I-ERRORS, USER$TSC:[GILLINGS.FAKE_RTL]REAL_SMGSHR.EXE;1 0 errors
Generating MACRO code for SMGSHR
Compiling FAKE_SMGSHR
Linking FAKE_SMGSHR
$
$ @fake_rtl librtl
Generating symbol vector for LIBRTL
%ANALYZE-I-ERRORS, USER$TSC:[GILLINGS.FAKE_RTL]REAL_LIBRTL.EXE;1 0 errors
Generating MACRO code for LIBRTL
Assumed JSB routine LIB$ANALYZE_SDESC_R2
Warning - vector hole before LIB$CVT_DTB. Expecting 272, got 288
Warning - vector hole before LIB$DELETE_FILE. Expecting 336, got 352
Warning - vector hole before LIB$EXTV. Expecting 432, got 448
Warning - vector hole before LIB$GET_COMMAND. Expecting 544, got 576
Warning - vector hole before LIB$INDEX. Expecting 608, got 624
Warning - vector hole before LIB$LOCC. Expecting 656, got 672
Warning - vector hole before LIB$SCANC. Expecting 848, got 864
Assumed JSB routine LIB$SGET1_DD_R6
Warning - vector hole before LIB$TRA_ASC_EBC. Expecting 1168, got 1184
Assumed JSB routine OTS$$CVT_D_T_R8
Assumed JSB routine OTS$$CVT_F_T_R8
Assumed JSB routine OTS$$CVT_G_T_R8
Assumed JSB routine OTS$$CVT_H_T_R8
Assumed JSB routine OTS$MOVE3_R5
Assumed JSB routine OTS$MOVE5_R5
Assumed JSB routine OTS$SGET1_DD_R6
Assumed JSB routine STR$ANALYZE_SDESC_R1
Assumed JSB routine STR$COPY_DX_R8
Assumed JSB routine STR$COPY_R_R8
Assumed JSB routine STR$FREE1_DX_R4
Assumed JSB routine STR$GET1_DX_R4
Assumed JSB routine STR$LEFT_R8
Assumed JSB routine STR$LEN_EXTR_R8
Assumed JSB routine STR$POSITION_R6
Assumed JSB routine STR$POS_EXTR_R8
Assumed JSB routine STR$REPLACE_R8
Assumed JSB routine STR$RIGHT_R8
Assumed JSB routine OTS$$RET_A_CVT_TAB_R1
Warning - vector hole before LIB$EMUL. Expecting 2896, got 2912
```

```
Warning - vector hole before LIB$REMQHI. Expecting 3104, got 3120
Warning - vector hole before STR$ADD. Expecting 3392, got 3680
Assumed JSB routine STR$ELEMENT_R8
Warning - vector hole before LIB$TABLE_PARSE. Expecting 4496, got 4576
Warning - vector hole before LIB$FIND_VM_ZONE. Expecting 4608, got 4640
Start of data block 1 00000000000D6EB0 LIB$AB_ASC_EBC
Compiling FAKE_LIBRTL
Linking FAKE_LIBRTL
$
$ @fake_rtl decc$shr
%COPY-S-COPIED, SYS$COMMON:[SYSLIB]DECC$SHR.EXE;1 copied to
USER$TSC:[GILLINGS.FAKE_RTL]REAL_DECC$SHR.EXE;1 (4483 blocks)
Generating symbol vector for DECC$SHR
%ANALYZE-I-ERRORS, USER$TSC:[GILLINGS.FAKE_RTL]REAL_DECC$SHR.EXE;1 0
errors
Generating MACRO code for DECC$SHR
Warning - vector hole before decc$bsd__cputchar. Expecting 22784, got
22800
Warning - vector hole before DECC$CONFSTR. Expecting 34736, got 34800
Warning - vector hole before decc$readdir_r. Expecting 46128, got 46224
Warning - vector hole before decc$poll. Expecting 49776, got 50000
Start of data block 1 00000000001F0000 DECC$GA_BSD_AE
Hole in data exceeded threshold at 00000000001F4E18 DECC$$GA_IO_BLOCK
Start of data block 2 00000000001F4E18 DECC$$GA_IO_BLOCK
Compiling FAKE_DECC$SHR
Linking FAKE_DECC$SHR
$
```

Appendix D: Sample Use

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

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

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

Appendix E: Sample Output

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

```
$ type/page argdump.log
Arg tracing started at 17-MAY-2005 15:33:46.62
Mapped Data LIBRTL1_DATA real:004EC000:004EFFFF => fake:00332000:00335FFF
Mapped Data DECC$SHR1_DATA real:00A5E000:00A5FFFF =>
fake:00604000:00605FFF
Mapped Data DECC$SHR2_DATA real:00A62000:00A63FFF =>
fake:00606000:00607FFF

DECC$MAIN at 15:33:46.65
R0:7FFCF87C R1:00002000 R2:00010830 R3:7AF08EB2 R4 :7FFCF814 R5: 7FFCF934
R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28 R11:7FFCDBE8
DECC$MAIN called with 9 args
   1 7FFCF884 => 00010830
              =/0/
   2 7AE63EC8 => 00303089
   3 7FFCF814 => 00000003
   4 7FFCF934 => 001C0048
              =/H/
   5 00000028
   6 00000000
   7 7AD49B78 => 7AEDB914
   8 7AD49B74 => 00000000
   9 7AD49B70 => 00000000

  LIB$GET_SYMBOL at 15:33:46.65
  R0:00000000 R1:00000001 R2:00A27A18 R3:00A13990 R4 :00000004 R5:
00000001
  R6:00000000 R7:00AFC008 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_SYMBOL called with 4 args
     1 7AD48F58 => 010E0004
                   00A13990 => PATH
     2 7AD48F28 => 010E0401
                   7AD48F70 =>
ÐeM     8       T•G     ÐeM     8'Ôz    ¸ L
     3 7AD48F60 => 00000000
     4 00000000
  LIB$GET_SYMBOL returning 4 args
     1 7AD48F58 => 010E0004
                   00A13990 => PATH
     2 7AD48F28 => 010E0401
                   7AD48F70 => sys$login
     3 7AD48F60 => 00000009
     4 00000000
  LIB$GET_SYMBOL returned: 00000001 at 15:33:46.66
  R0:00000001 R1:00000009 R2:00A27A18 R3:00A13990 R4 :00000004 R5:
00000001
  R6:00000000 R7:00AFC008 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$INSERT_TREE at 15:33:46.66
  R0:00000000 R1:00000001 R2:00A3F2D8 R3:00AFC008 R4 :00000001 R5:
00AA0280
```

```
  R6:7AD48F70 R7:00AA0280 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$INSERT_TREE called with 7 args
     1 00A67B40 => 00000000
     2 7AD48EC8 => 00A13990
     3 7AD48ED8 => 00000000
     4 00A3F170 => 0116300A
     5 00A3F288 => 00083089
     6 7AD48EE0 => 00A0FF10
     7 00000000

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

    LIB$VM_MALLOC at 15:33:46.66
    R0:00B2C008 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008
    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000
    LIB$VM_MALLOC called with 1 arg
       1 00000005
    LIB$VM_MALLOC returning 1 arg
       1 00000005
    LIB$VM_MALLOC returned: 00B2C030 at 15:33:46.66
    R0:00B2C030 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008
    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000

    LIB$VM_MALLOC at 15:33:46.66
    R0:00B2C030 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008
    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000
    LIB$VM_MALLOC called with 1 arg
       1 0000000A
    LIB$VM_MALLOC returning 1 arg
       1 0000000A
    LIB$VM_MALLOC returned: 00B2C040 at 15:33:46.66
    R0:00B2C040 R1:00000001 R2:00A1A260 R3:7AD48EC8 R4 :7AD48B30 R5:
00B2C008
    R6:00158001 R7:7AD48EE0 R8:00A3F288 R9:00A3F170 R10:7AD48EC8
R11:00000000

  LIB$INSERT_TREE returning 7 args
     1 00A67B40 => 00B2C008
     2 7AD48EC8 => 00A13990
     3 7AD48ED8 => 00000000
     4 00A3F170 => 0116300A
```

```
     5 00A3F288 => 00083089
     6 7AD48EE0 => 00B2C008
     7 00000000
  LIB$INSERT_TREE returned: 00158001 at 15:33:46.66
  R0:00158001 R1:00158214 R2:00A3F2D8 R3:00AFC008 R4 :00000001 R5:
00AA0280
  R6:7AD48F70 R7:00AA0280 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_SYMBOL at 15:33:46.66
  R0:00000000 R1:00000001 R2:00A27A18 R3:00A139F0 R4 :00000009 R5:
00000001
  R6:00000000 R7:00AFC008 R8:00A9E2D7 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_SYMBOL called with 4 args
     1 7AD48F58 => 010E0009
                   00A139F0 => VAXC$PATH
     2 7AD48F28 => 010E0401
                   7AD48F70 => sys$login
     3 7AD48F60 => 00000000
     4 00000000
  LIB$GET_SYMBOL returning 4 args
     1 7AD48F58 => 010E0009
                   00A139F0 => VAXC$PATH
     2 7AD48F28 => 010E0401
                   7AD48F70 => sys$login
     3 7AD48F60 => 00000000
     4 00000000
  LIB$GET_SYMBOL returned: 00158364 at 15:33:46.66
  R0:00158364 R1:FFFFFFFF R2:00A27A18 R3:00A139F0 R4 :00000009 R5:
00000001
  R6:00000000 R7:00AFC008 R8:00A9E2D7 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$INSERT_TREE at 15:33:46.66
  R0:00000000 R1:00000001 R2:00A3F2D8 R3:00AFC008 R4 :00000001 R5:
0000000E
  R6:7AD49030 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$INSERT_TREE called with 7 args
     1 00A67B40 => 00B2C008
     2 7AD48F88 => 00A13CF0
     3 7AD48F98 => 00000000
     4 00A3F170 => 0116300A
     5 00A3F288 => 00083089
     6 7AD48FA0 => 20202020
                    =/        ti/
     7 00000000

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

```
    R6:00158001 R7:7AD48FA0 R8:00A3F288 R9:00A3F170 R10:7AD48F88
R11:00000000

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

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

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

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

```
  R0:00B2C0C0 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65780
  R6:7FF9DEA7 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

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

  LIB$VM_CALLOC at 15:33:46.66
  R0:00B2C300 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65778
  R6:00B2C300 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$VM_CALLOC called with 2 args
     1 00000001
     2 00000421
  LIB$VM_CALLOC returning 2 args
     1 00000001
     2 00000421
  LIB$VM_CALLOC returned: 00B2C780 at 15:33:46.66
  R0:00B2C780 R1:00000000 R2:00A1A280 R3:7FFCF934 R4 :00000000 R5:
00A65778
  R6:00B2C300 R7:7FFA0ED0 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_FOREIGN at 15:33:46.66
  R0:00000000 R1:00000000 R2:00A13DB8 R3:7FFCF934 R4 :00000000 R5:
00A65778
  R6:00B2C300 R7:00B2C780 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_FOREIGN called with 3 args
     1 7AD497D8 => 01000421
                   =/!/
     2 00000000
     3 7AD497E0 => 00000000
  LIB$GET_FOREIGN returning 3 args
     1 7AD497D8 => 01000421
                   =/!/
     2 00000000
     3 7AD497E0 => 00000000
  LIB$GET_FOREIGN returned: 00000001 at 15:33:46.66
  R0:00000001 R1:00000000 R2:00A13DB8 R3:7FFCF934 R4 :00000000 R5:
00A65778
  R6:00B2C300 R7:00B2C780 R8:7FF9CDE8 R9:7FF9DDF0 R10:7FFA4F28
R11:7FFCDBE8

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

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

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

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

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

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

```
   6 7AD49A50 => 00606000
   7 00000000

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

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

  LIB$GET_VM at 15:33:46.66
  R0:00128069 R1:0000014C R2:00B9A500 R3:00BC8020 R4 :00BA8860 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
```

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

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

  LIB$GET_VM at 15:33:46.66
  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_VM called with 3 args
     1 7AD49250 => 00004380
     2 007D2968 => 00000000
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD49250 => 00004380
     2 007D2968 => 007D29A8
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.66
  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM at 15:33:46.66
  R0:007D2960 R1:0000004A R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
0000FFFF
```

```
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_VM called with 3 args
     1 7AD49250 => 0000004A
                   =/J/
     2 007D298C => 00000000
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD49250 => 0000004A
                   =/J/
     2 007D298C => 007D6D38
                   =/8m}/
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.67
  R0:00000001 R1:09110000 R2:00B9A4E0 R3:7AD493D0 R4 :00BC8020 R5:
0000FFFF
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM at 15:33:46.68
  R0:00000001 R1:007D2808 R2:00B9A500 R3:00BC8020 R4 :00BA8860 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_VM called with 3 args
     1 7AD492A0 => 00000200
     2 007D2878 => 00000000
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD492A0 => 00000200
     2 007D2878 => 007D6D90
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B9A500 R3:00BC8020 R4 :00BA8860 R5:
7AD49670
  R6:00128069 R7:00BC80D0 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM at 15:33:46.68
  R0:007D2908 R1:007D290C R2:00B9BA80 R3:007D2808 R4 :00BA8860 R5:
7AD49670
  R6:000000FF R7:007D2908 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8
  LIB$GET_VM called with 3 args
     1 007D2908 => 000000FF
     2 007D290C => 00000000
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 007D2908 => 000000FF
     2 007D290C => 007D6FA0
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B9BA80 R3:007D2808 R4 :00BA8860 R5:
7AD49670
  R6:000000FF R7:007D2908 R8:00000000 R9:00BA8750 R10:7FFA4F28
R11:7FFCDBE8

  LIB$GET_VM at 15:33:46.68
  R0:00000000 R1:00000000 R2:00B9AA00 R3:00000007 R4 :00124B60 R5:
00090020
```

```
  R6:0000000A R7:00BC8024 R8:00000000 R9:00BA8750 R10:00000000
R11:7FFCDBE8
  LIB$GET_VM called with 3 args
     1 7AD496B0 => 0000000A
     2 007D28C0 => 00000000
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD496B0 => 0000000A
     2 007D28C0 => 007D70B0
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B9AA00 R3:00000007 R4 :00124B60 R5:
00090020
  R6:0000000A R7:00BC8024 R8:00000000 R9:00BA8750 R10:00000000
R11:7FFCDBE8

SMG$CREATE_PASTEBOARD returning 7 args
   1 00124B40 => 00000000
   2 00000000
   3 00124B60 => 00000024
              =/$/
   4 00090020 => 00000050
              =/P/
   5 00020000 => 00000001
   6 7AD49A50 => 00000006
   7 00000000
SMG$CREATE_PASTEBOARD returned: 00000001 at 15:33:46.68
R0:00000001 R1:00BC8024 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:7FFA4F28 R11:7FFCDBE8

DECC$GETENV at 15:33:46.68
R0:00000001 R1:00BC8024 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:7FFA4F28 R11:7FFCDBE8
DECC$GETENV called with 1 arg
   1 00010790 => 24554644
              =/DFU$NOSMG/

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

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

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

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

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

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

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

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

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

  LIB$GET_VM at 15:33:46.68
  R0:00128069 R1:00128069 R2:00B992F0 R3:00000000 R4 :00000050 R5:
000001A0
  R6:00BA855C R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080
  LIB$GET_VM called with 3 args
     1 7AD49210 => 000001E8
     2 7AD491E0 => 818D55A0
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD49210 => 000001E8
     2 7AD491E0 => 007D72D0
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B992F0 R3:00000000 R4 :00000050 R5:
000001A0
  R6:00BA855C R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080

  LIB$GET_VM at 15:33:46.68
  R0:00000001 R1:007D72D0 R2:00B992F0 R3:00000000 R4 :00000000 R5:
00000050
  R6:00BA8330 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080
  LIB$GET_VM called with 3 args
     1 7AD49208 => 000007A0
```

```
     2 7AD491E8 => 00000000
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD49208 => 000007A0
     2 7AD491E8 => 007D74C8
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B992F0 R3:00000000 R4 :00000000 R5:
00000050
  R6:00BA8330 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080

  LIB$GET_VM at 15:33:46.68
  R0:00000001 R1:00000000 R2:00B98B18 R3:00000000 R4 :00000050 R5:
000000A0
  R6:00000014 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080
  LIB$GET_VM called with 3 args
     1 7AD493B8 => 000000A0
     2 007D729C => 00000000
     3 00BC8020 => 007D0800
  LIB$GET_VM returning 3 args
     1 7AD493B8 => 000000A0
     2 007D729C => 007D7C78
                 =/x|}/
     3 00BC8020 => 007D0800
  LIB$GET_VM returned: 00000001 at 15:33:46.68
  R0:00000001 R1:09110000 R2:00B98B18 R3:00000000 R4 :00000050 R5:
000000A0
  R6:00000014 R7:00BC8020 R8:0000003E R9:007D7120 R10:00124270
R11:00090080

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

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

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

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

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

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

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

  LIB$INSERT_TREE at 15:33:46.68
  R0:00000001 R1:007D7D6C R2:00B98D28 R3:00000006 R4 :007D7D28 R5:
001262C0
  R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080
  LIB$INSERT_TREE called with 7 args
     1 007D7D28 => 00000000
     2 007D7D58 => 010E000A
                   007D7D64 => DEFAULT  (
     3 7AD49668 => 00000000
     4 00B98F60 => 00183089
     5 00B99028 => 00283089
     6 7AD49658 => 00000000
     7 00000000

    LIB$GET_VM at 15:33:46.68
    R0:0045C834 R1:00000054 R2:004C6FB0 R3:004C6F78 R4 :00000000 R5:
00000000
```

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

  LIB$INSERT_TREE returning 7 args
     1 007D7D28 => 007D7D98
     2 007D7D58 => 010E000A
                   007D7D64 => DEFAULT  (
     3 7AD49668 => 00000000
     4 00B98F60 => 00183089
     5 00B99028 => 00283089
     6 7AD49658 => 007D7D98
     7 00000000
  LIB$INSERT_TREE returned: 00158001 at 15:33:46.68
  R0:00158001 R1:00158214 R2:00B98D28 R3:00000006 R4 :007D7D28 R5:
001262C0
  R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00090080

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

…
---Many lines omitted---
…
DECC$GXVSPRINTF at 15:33:46.96
R0:00000031 R1:7AD49998 R2:00010EC0 R3:00124930 R4 :00070004 R5: 001262C0
```

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

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

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

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

```
     3 00BA8590 => 000000FF
  LIB$SCANC returned: 00000000 at 15:33:46.96
  R0:00000000 R1:00124961 R2:00B99660 R3:00000007 R4 :007D80F8 R5:
00000000
  R6:00000031 R7:00124930 R8:00000000 R9:00000000 R10:00000000
R11:00000000

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

SMG$END_PASTEBOARD_UPDATE at 15:33:46.96
R0:00000000 R1:000900F0 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00000001
SMG$END_PASTEBOARD_UPDATE called with 1 arg
   1 00124B40 => 00000000
SMG$END_PASTEBOARD_UPDATE returning 1 arg
   1 00124B40 => 00000000
SMG$END_PASTEBOARD_UPDATE returned: 00000001 at 15:33:46.97
R0:00000001 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00000001

DECC$GETENV at 15:33:46.97
R0:00000001 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00126380
DECC$GETENV called with 1 arg
   1 000105B0 => 24554644
                 =/DFU$TRACE/

  LIB$GET_SYMBOL at 15:33:46.97
  R0:00000000 R1:00000001 R2:00A27A18 R3:000105B0 R4 :00000009 R5:
00000001
  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$GET_SYMBOL called with 4 args
     1 7AD492F8 => 010E0009
                   000105B0 => DFU$TRACE
     2 7AD492C8 => 010E0401
                   7AD49310 =>
        a       q               °€  ÿÿÿÿ°Ð‹
     3 7AD49300 => 00000000
     4 00000000
  LIB$GET_SYMBOL returning 4 args
     1 7AD492F8 => 010E0009
                   000105B0 => DFU$TRACE
     2 7AD492C8 => 010E0401
                   7AD49310 =>
        a       q               °€  ÿÿÿÿ°Ð‹
     3 7AD49300 => 00000000
     4 00000000
```

```
  LIB$GET_SYMBOL returned: 00158364 at 15:33:46.97
  R0:00158364 R1:FFFFFFFF R2:00A27A18 R3:000105B0 R4 :00000009 R5:
00000001
  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380

DECC$GETENV returning 1 arg
   1 000105B0 => 24554644
              =/DFU$TRACE/
DECC$GETENV returned: 00000000 at 15:33:46.97
R0:00000000 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00126380

LIB$CREATE_VM_ZONE at 15:33:46.97
R0:00126400 R1:000204A0 R2:00010E00 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00126380
LIB$CREATE_VM_ZONE called with 13 args
   1 000204A0 => 00000000
   2 7AD499A8 => 00000002
   3 7AD499A0 => 00000080
   4 7AD49998 => 00000028
              =/(/
   5 7AD49990 => 00000008
   6 00000000
   7 00000000
   8 00000000
   9 00000000
  10 00000000
  11 7AD49988 => 010E0008
                 00010E30 => dfu_zone
  12 00000000
  13 00000000
LIB$CREATE_VM_ZONE returning 13 args
   1 000204A0 => 007D0858
              =/X/
   2 7AD499A8 => 00000002
   3 7AD499A0 => 00000080
   4 7AD49998 => 00000028
              =/(/
   5 7AD49990 => 00000008
   6 00000000
   7 00000000
   8 00000000
   9 00000000
  10 00000000
  11 7AD49988 => 010E0008
                 00010E30 => dfu_zone
  12 00000000
  13 00000000
LIB$CREATE_VM_ZONE returned: 00000001 at 15:33:46.97
R0:00000001 R1:0000C3A5 R2:00010E00 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00126380

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

  LIB$GET_SYMBOL at 15:33:46.97
```

```
  R0:00000000 R1:00000001 R2:00A27A18 R3:00013D40 R4 :00000011 R5:
00000001
  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$GET_SYMBOL called with 4 args
     1 7AD492F8 => 010E0011
                   00013D40 => DFU$DISABLE_CHECK
     2 7AD492C8 => 010E0401
                   7AD49310 =>
        a       q               °€  ÿÿÿÿ°Ð‹
     3 7AD49300 => 00000000
     4 00000000
  LIB$GET_SYMBOL returning 4 args
     1 7AD492F8 => 010E0011
                   00013D40 => DFU$DISABLE_CHECK
     2 7AD492C8 => 010E0401
                   7AD49310 =>
        a       q               °€  ÿÿÿÿ°Ð‹
     3 7AD49300 => 00000000
     4 00000000
  LIB$GET_SYMBOL returned: 00158364 at 15:33:46.97
  R0:00158364 R1:FFFFFFFF R2:00A27A18 R3:00013D40 R4 :00000011 R5:
00000001
  R6:00000000 R7:00AFC008 R8:00020000 R9:00090020 R10:00124270
R11:00126380

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

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

DECC$STRNCMP at 15:33:46.98
R0:00000001 R1:7AD40008 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA800 R8:00000001 R9:00000000 R10:FFFFFFFF R11:00126380
DECC$STRNCMP called with 3 args
   1 00070140 => 4C4C4947
                 =/GILLINGS                                          ...
   2 000104E0 => 5F554644
                 =/DFU_ALLPRIV/
   3 00000008
```

```
DECC$STRNCMP returning 3 args
   1 00070140 => 4C4C4947
                 =/GILLINGS                                          ...
   2 000104E0 => 5F554644
                 =/DFU_ALLPRIV/
   3 00000008
DECC$STRNCMP returned: 00000001 at 15:33:46.98
R0:00000001 R1:00000000 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA800 R8:00000001 R9:00000000 R10:FFFFFFFF R11:00126380

DECC$STRNCMP at 15:33:46.98
R0:00000001 R1:7AD4000B R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA808 R8:00000002 R9:00000000 R10:FFFFFFFF R11:00126380
DECC$STRNCMP called with 3 args
   1 00070140 => 45544E49
                 =/INTERACTIVE                                       ...
   2 000104E0 => 5F554644
                 =/DFU_ALLPRIV/
   3 0000000B
DECC$STRNCMP returning 3 args
   1 00070140 => 45544E49
                 =/INTERACTIVE                                       ...
   2 000104E0 => 5F554644
                 =/DFU_ALLPRIV/
   3 0000000B
DECC$STRNCMP returned: 00000001 at 15:33:46.98
R0:00000001 R1:00000000 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA808 R8:00000002 R9:00000000 R10:FFFFFFFF R11:00126380

DECC$STRNCMP at 15:33:46.98
R0:00000001 R1:7AD40005 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA810 R8:00000003 R9:00000000 R10:FFFFFFFF R11:00126380
DECC$STRNCMP called with 3 args
   1 00070140 => 41434F4C
                 =/LOCAL                                             ...
   2 000104E0 => 5F554644
                 =/DFU_ALLPRIV/
   3 00000005
DECC$STRNCMP returning 3 args
   1 00070140 => 41434F4C
                 =/LOCAL                                             ...
   2 000104E0 => 5F554644
                 =/DFU_ALLPRIV/
   3 00000005
DECC$STRNCMP returned: 00000008 at 15:33:46.98
R0:00000008 R1:00000007 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA810 R8:00000003 R9:00000000 R10:FFFFFFFF R11:00126380

DECC$STRNCMP at 15:33:46.98
R0:00002224 R1:7AD40005 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA818 R8:00000004 R9:00000000 R10:FFFFFFFF R11:00126380
DECC$STRNCMP called with 3 args
   1 00070140 => 41434F4C
                 =/LOCAL                                             ...
   2 000104E0 => 5F554644
                 =/DFU_ALLPRIV/
   3 00000005
DECC$STRNCMP returning 3 args
   1 00070140 => 41434F4C
                 =/LOCAL                                             ...
   2 000104E0 => 5F554644
```

```
                =/DFU_ALLPRIV/
   3 00000005
DECC$STRNCMP returned: 00000008 at 15:33:46.98
R0:00000008 R1:00000007 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA818 R8:00000004 R9:00000000 R10:FFFFFFFF R11:00126380

DECC$STRNCMP at 15:33:46.98
R0:00002224 R1:7AD40005 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA820 R8:00000005 R9:00000000 R10:FFFFFFFF R11:00126380
DECC$STRNCMP called with 3 args
   1 00070140 => 41434F4C
                =/LOCAL                                              ...
   2 000104E0 => 5F554644
                =/DFU_ALLPRIV/
   3 00000005
DECC$STRNCMP returning 3 args
   1 00070140 => 41434F4C
                =/LOCAL                                              ...
   2 000104E0 => 5F554644
                =/DFU_ALLPRIV/
   3 00000005
DECC$STRNCMP returned: 00000008 at 15:33:46.98
R0:00000008 R1:00000007 R2:00010468 R3:7AD49A70 R4 :00070140 R5: 00070004
R6:00000005 R7:007DA820 R8:00000005 R9:00000000 R10:FFFFFFFF R11:00126380

SMG$READ_COMPOSED_LINE at 15:33:46.98
R0:00000000 R1:00000001 R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00126380
SMG$READ_COMPOSED_LINE called with 11 args
   1 00124270 => 007D7120
                =/ q}/
   2 00070000 => 007D7D28
                =/(}}/
   3 7AD49A40 => 010E00FF
                 00070248 =>
   4 7AD49A38 => 010E0005
                 00010640 => DFU>
   5 7AD49A80 => 00000000
   6 00090090 => 007D8598
   7 00000000
   8 00000000
   9 00000000
  10 00000000
  11 00000000

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

```
  LIB$GET_VM returned: 00000001 at 15:33:46.98
  R0:00000001 R1:09110000 R2:00B987D0 R3:7AD49414 R4 :00000007 R5:
001262C0
  R6:00128402 R7:00000002 R8:00000001 R9:007D7D28 R10:00000001
R11:00000000

  LIB$LOOKUP_TREE at 15:33:48.02
  R0:007D7D6D R1:007D7D6C R2:00B98F90 R3:007D8598 R4 :0000001A R5:
00000000
  R6:007D7D58 R7:00000002 R8:00000000 R9:007D7D28 R10:007D7D6C
R11:00000001
  LIB$LOOKUP_TREE called with 4 args
     1 007D7D28 => 007D7EB8
     2 007D7D58 => 010E000A
                   007D7D64 => DEFAULT
     3 00B98F60 => 00183089
     4 7AD49298 => 00B5D8E0
  LIB$LOOKUP_TREE returning 4 args
     1 007D7D28 => 007D7EB8
     2 007D7D58 => 010E000A
                   007D7D64 => DEFAULT
     3 00B98F60 => 00183089
     4 7AD49298 => 00B5D8E0
  LIB$LOOKUP_TREE returned: 001582FC at 15:33:48.02
  R0:001582FC R1:00B98F60 R2:00B98F90 R3:007D8598 R4 :0000001A R5:
00000000
  R6:007D7D58 R7:00000002 R8:00000000 R9:007D7D28 R10:007D7D6C
R11:00000001

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

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

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

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

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

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

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

SMG$READ_COMPOSED_LINE returning 11 args
   1 00124270 => 007D7120
                 =/ q}/
   2 00070000 => 007D7D28
                 =/(}}/
   3 7AD49A40 => 010E00FF
                 00070248 =>
   4 7AD49A38 => 010E0005
                 00010640 => DFU>
   5 7AD49A80 => 00000000
   6 00090090 => 007D8598
   7 00000000
   8 00000000
   9 00000000
  10 00000000
  11 00000000
SMG$READ_COMPOSED_LINE returned: 00128402 at 15:33:48.04
R0:00128402 R1:000000FF R2:000106C0 R3:00070248 R4 :00070004 R5: 001262C0
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00126380

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

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

SMG$DELETE_PASTEBOARD at 15:33:48.04
R0:00020140 R1:00000002 R2:000103F0 R3:00070248 R4 :00070004 R5: 001262C0
```

```
R6:00090020 R7:00124B40 R8:00020000 R9:00090020 R10:00124270 R11:00126380
SMG$DELETE_PASTEBOARD called with 2 args
   1 00124B40 => 00000000
   2 000200F8 => 00000000

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

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

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

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

  LIB$FREE_VM at 15:33:48.04
  R0:007D2960 R1:7FF5E300 R2:00B9A4C0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD49680 => 00004380
     2 007D2968 => 007D29A8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD49680 => 00004380
     2 007D2968 => 007D29A8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.04
  R0:00000001 R1:00000001 R2:00B9A4C0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:00000000 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380

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

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

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

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

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

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

```
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B9A8D0 R3:00124B40 R4 :007D2808 R5:
00BC80A4
  R6:007D2808 R7:00124B40 R8:00020000 R9:00090020 R10:00124270
R11:00126380

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

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

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

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

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

```
  R0:00000000 R1:007D72B8 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8
  R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD495C8 => 0000000C
     2 7AD495B8 => 007D72B8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD495C8 => 0000000C
     2 7AD495B8 => 007D72B8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D7120 R5:
007D70C8
  R6:00000001 R7:7FF87FC0 R8:00020000 R9:00090020 R10:00124270
R11:00126380

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

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

  LIB$FREE_VM at 15:33:48.06
  R0:00000000 R1:000000A0 R2:00B984D0 R3:007D7120 R4 :00000013 R5:
00000014
  R6:000000A0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380
```

```
  LIB$FREE_VM called with 3 args
     1 7AD495C8 => 000000A0
     2 007D729C => 007D7C78
                   =/x|}/
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD495C8 => 000000A0
     2 007D729C => 007D7C78
                   =/x|}/
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :00000013 R5:
00000014
  R6:000000A0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380

  LIB$FREE_VM at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D70E8 R5:
00000014
  R6:000001C0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380
  LIB$FREE_VM called with 3 args
     1 7AD495C8 => 000001C0
     2 7AD495D0 => 007D70E8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returning 3 args
     1 7AD495C8 => 000001C0
     2 7AD495D0 => 007D70E8
     3 00BC8020 => 007D0800
  LIB$FREE_VM returned: 00000001 at 15:33:48.06
  R0:00000001 R1:00000001 R2:00B984D0 R3:007D7120 R4 :007D70E8 R5:
00000014
  R6:000001C0 R7:00BC8020 R8:00020000 R9:00090020 R10:00124270
R11:00126380

  exit status: 00000001
```

Appendix F: Fake RTL Command Procedure

This appendix contains the Fake RTL command procedure.

```
$! Creates a fake RTL from an input shareable image
$!
$! Alpha and I64 version
$!
$! Author:
$!                   John Gillings
$!                   Software Systems Consultant, OpenVMS Ambassador
$!                   Hewlett-Packard Pty Limited
$!                   OpenVMS Group, Customer Support Centre
$!                   Sydney, Australia
$!
$! @FAKE_RTL [name] [start-phase] [options]
$!
$! Phases are:
$!
$!   DEF     - Define fake rtl environment
$!   COPY    - Make a copy of the original shareable image
$!   VECTOR  - Analyze the symbol vector
$!   GEN     - Generate MACRO code
$!   COMPILE - Compile MACRO code
$!   LINK    - Link fake RTL
$!   USE     - Define logical names to use image
$!
$! Options are:
$!   DEBUG   - Compile and link with /DEBUG
$!   FORCE   - Force creation of files, even if apparently unnecessary
$!
$!
$ exp=p2.NES.""
$ IF .NOT.exp THEN p2="DEF"
$ IF p1.EQS.""
$ THEN
$   endphase="DEF"
$ ELSE
$   endphase="USE"
$   img=F$SEARCH(F$PARSE(p1,"SYS$SHARE:",".EXE"))
$   IF img.EQS."" THEN EXIT
$   name=F$PARSE(img,,,"NAME")
$   newimg="[]REAL_''name'.EXE;"
$ ENDIF
$ dbg=(p3.NES.""".AND.F$LOCATE("DEBUG",p3).LT.F$LENGTH(p3)).OR.-
       F$TRNLNM("FAKE_OPTION_DEBUG")
$ force=p4.NES.""".AND.F$LOCATE("FORCE",p3).LT.F$LENGTH(p3).OR.-
       F$TRNLNM("FAKE_OPTION_FORCE")
$ pid=F$GETJPI("","PID")
$ arch=F$GETSYI("ARCH_NAME")
$ IF arch.NES."Alpha".AND.arch.NES."IA64"
$ THEN
$   WRITE SYS$OUTPUT "Sorry, FAKE_RTL can't do ''arch'"
$   EXIT
$ ENDIF
$ GOTO 'p2'
$
$ DEF:
$   IF force.OR.F$TRNLNM("FAKE_DIR").EQS.""
$   THEN
$     WRITE SYS$OUTPUT "Defining FAKE_RTL environment"
$     ThisFile=F$ENVIRONMENT("PROCEDURE")
$     ThisDisk=F$PARSE(ThisFile,,,"DEVICE")
$     ThisDir=F$PARSE(ThisFile,,,"DIRECTORY")
$     DEFINE/NOLOG FAKE_DIR 'ThisDisk''ThisDir'
```

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

```
$!
$! IMAGE HEADER
$!
$!        Fixed Header Information
$!
$!                image format major id: 3, minor id: 0
$!                header block count: 3
$!                image type: shareable (EIHD$K_LIM)
$!                        global section major id: %X'04', minor id: %X'0003E9'
$!                        match control: ISD$K_MATLEQ
$!                I/O channel count: default
$!
$!--------------------------------------------------------------------------
$!
$!
$!   Now to check the symbol vector. ANALYZE/IMAGE lists each symbol as a
$!   "Universal Symbol Specification". For example:
$!
$!        6) Universal Symbol Specification (EGSD$C_SYMG)
$!                data type: DSC$K_DTYPE_Z (0)
$!                symbol flags:
$!                        (0)  EGSY$V_WEAK      0
$!                        (1)  EGSY$V_DEF       1
$!                        (2)  EGSY$V_UNI       1
$!                        (3)  EGSY$V_REL       1
$!                        (4)  EGSY$V_COMM      0
$!                        (5)  EGSY$V_VECEP     0
$!                        (6)  EGSY$V_NORM      1
$!                psect: 0
$!                value: 592 (%X'00000250')
$!                symbol vector entry (procedure)
$!                        %X'00000000 0005F9AC'
$!                        %X'00000000 0008B980'
$!                symbol: "DBASIC$DATE_T"
$!
$!
$!   So here's how we do it...
$
$ IF arch.EQS."IA64" THEN GOTO IA64Image
$ IF arch.NES."Alpha"
$ THEN
$   WRITE SYS$OUTPUT "Sanity check failure. Architecture=''arch' - shouldn't get
to here!"
$   EXIT
$ ENDIF
$!
$! Alpha specific
$!   Start with full analysis text
$!
$ ANALYZE/IMAGE/OUT=TMP_'pid'.ANL 'newimg'
$!
$!   Find image ident and match control
$ SEARCH/OUT=TMP_'pid'.GSM TMP_'pid'.ANL "global section major id:",-
                                    "match control:"
$ OPEN/READ in TMP_'pid'.GSM
$ what="Global section ID"
$ READ/END=BadImg in line
$ maj=F$ELEMENT(1,"'",line)
$ min=F$ELEMENT(3,"'",line)
$ what="Match control"
$ READ/END=BadImg in line
$ mat=F$ELEMENT(1,"_",line)
$!
$ dat=""
$ CLOSE in
$!
$! Remove headers
$ ff[0,8]=12
```

```
$ SEARCH/MATCH=NOR/EXACT/OUTPUT=TMP_'pid'.AN1 TMP_'pid'.ANL -
      "''ff'","''newimg'","Analyze Image  ","ANALYZ "
$!
$! Get symbol blocks
$ SEARCH/EXACT/NOHEAD/OUTPUT=TMP_'pid'.AN2 TMP_'pid'.AN1 "symbol: "/WINDOW=(5,0)
$!
$! Remove other junk
$ SEARCH/EXACT/NOHEAD/OUTPUT=TMP_'pid'.AN3 TMP_'pid'.AN2 "symbol","%X"
$
$!   We should now have a list of symbol definitions like this:
$!
$!               value: 0 (%X'00000000')
$!               symbol vector entry (constant)
$!                     %X'00000000 00000000'
$!                     %X'00000000 0035800C' (3506188)
$!               symbol: "C$_EPERM"
$!
$!
$!   Now parse the list. The objective is to produce a file with value,
$!   type, vector entry and symbol name on one line, then sort on value.
$!
$!   There are 3 types of symbol - "constant", "procedure" and "data cell"
$!   For constants, we need the second symbol vector entry value - as it's
$!   the constant value. For data cells the symbol vector gives us the offset
$!   to the data from the start of the shareable image. If any data cell
$!   vector entries are present, we generate a file containing the symbol
$!   names and offsets, sorted on offset. This can be used to generate a
$!   replica data area in the fake image.
$!
$ OPEN/READ in TMP_'pid'.AN3
$ OPEN/WRITE out TMP_'pid'.VEC
$ OPEN/WRITE dat TMP_'pid'.DATA
$!
$ genloop: READ/END=endgenloop in line
$   line=F$EDIT(line,"COLLAPSE")
$   IF F$LOCATE("value:",line).NE.0 THEN GOTO genloop
$   val=F$ELEMENT(1,"'",line)
$   READ/END=endgenloop in line
$   type=F$EDIT(line,"COLLAPSE")-"symbolvectorentry"
$   IF type.EQS.line THEN GOTO loop     ! Read symbol vector values
$   READ/END=endgenloop in line
$   v1=F$ELEMENT(1,"'",line)-" "
$   READ/END=endgenloop in line
$   v2=F$ELEMENT(1,"'",line)-" "
$   READ/END=endgenloop in line
$   line=F$EDIT(line,"COLLAPSE")
$   IF F$LOCATE("symbol:",line).NE.0 THEN GOTO loop
$   sym=F$ELEMENT(1,"""",line)
$   WRITE OUT "''val' ''v1' ''v2' ''sym' ''type'"
$   IF type.EQS."(datacell)"
$   THEN
$     WRITE dat "''v2' ''sym'"
$     dat="Data Present"
$   ENDIF
$ GOTO genloop
$ endgenloop:
$!
$!  Generate a match control string. Note required spaces at beginning
$!  of line to keep it at the top of the vector file even after sorting
$!  Add comment tag to indicate if data is present
$!
$ matall="ALWAYS"
$ matequ="EQUAL"
$ matleq="LEQUAL"
$ matnev="NEVER"
$ match="  GSMATCH="+'mat'+",%X''maj',%X''min' ! "+dat
$ write out match
$ SET NOON
```

```
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ SORT TMP_'pid'.VEC  'name'.VEC
$ SORT TMP_'pid'.DATA 'name'.DATA
$ Cleanup:
$ err=$status
$ DELETE TMP_'pid'*.*;*
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ IF .NOT.err THEN EXIT
$ IF endphase.NES."VECTOR" THEN GOTO GEN
$ EXIT
$ BadImg: WRITE SYS$OUTPUT "Can't parse image ''what' -> ''line'"
$ EXIT
$
$ IA64Image:
$
$  ANALYZE/IMAGE/NOPAGE/SECTION=SYMBOL_VECTOR/OUT=TMP_'pid'.ANL 'newimg'
$  SEARCH/OUT=TMP_'pid'.GSM TMP_'pid'.ANL "Algorithm:","Major ID:","Minor ID:"
$
$ OPEN/READ in TMP_'pid'.GSM
$ what="Algorithm"
$ READ/END=BadIA64Img in line
$ algo=F$ELEMENT(1,":",F$EDIT(line,"COLLAPSE"))-"/"
$ what="Major ID"
$ READ/END=BadIA64Img in line
$ maj=F$ELEMENT(1,":",F$EDIT(line,"COLLAPSE"))-"."
$ what="Minor ID"
$ READ/END=BadIA64Img in line
$ min=F$ELEMENT(1,":",F$EDIT(line,"COLLAPSE"))-"."
$ CLOSE/NOLOG in
$
$!
$!  Find and parse the symbol vector. Note that the IA64 output is much closer
$!  to what we actually need than the Alpha output, but for historical reasons
$!  it will be "translated" back to the same format the Alpha code generates.
$!
$ OPEN/READ in TMP_'pid'.ANL
$ OPEN/WRITE out TMP_'pid'.VEC
$ OPEN/WRITE dat TMP_'pid'.DATA
$
$ dat=""
$ FindIA64SymVec: READ/END=NoIA64Vec in line
$   IF F$LENGTH(line).EQ.0.OR.F$LOCATE("SYMBOL VECTOR",line).GT.0 THEN GOTO
FindIA64SymVec
$ READ/END=NoIA64Vec in line
$ READ/END=NoIA64Vec in line
$ IF F$LOCATE("FileAddr Offset",line).GT.0 THEN GOTO NoIA64Vec
$ READ/END=NoIA64Vec in line
$
$ IF p4.NES."" THEN SET VERIFY
$ GenIA64loop: READ/END=EndIA64Vec in line
$  IF F$LENGTH(line).EQ.0 THEN GOTO GenIA64loop
$  IF F$LOCATE("The analysis",line).EQ.0 THEN GOTO EndIA64Vec
$  FileAddr=F$EDIT(F$EXTRACT(0,8,line),"COLLAPSE")
$  Offset=F$EDIT(F$EXTRACT(9,8,line),"COLLAPSE")
$  indx=F$EDIT(F$EXTRACT(18,7,line),"COLLAPSE")
$  val=F$EDIT(F$EXTRACT(28,16,line),"COLLAPSE")
$  IF FileAddr.EQS."".OR.Offset.EQS."".OR.indx.EQS."" THEN GOTO GenIA64loop
$  type=F$EDIT(F$EXTRACT(45,11,line),"COLLAPSE")-"("-")"
$  IF type.EQS."" THEN type="SPARE"
$  DATA="(datacell)"
$  DATAABS="(constant)"
$  PROCEDURE="(procedure)"
$  SPARE="(spare)"
$  type='type'
```

```
$  gp=F$EDIT(F$EXTRACT(56,16,line),"COLLAPSE")
$  IF gp.EQS."" THEN gp="0000000000000000"
$  sym=F$EDIT(F$ELEMENT(1,"""",line),"COLLAPSE")
$  IF sym.EQS."""" THEN sym="*spare*"
$  WRITE OUT "''Offset' ''val' ''val' ''sym' ''type'"
$  IF type.EQS."(datacell)"
$  THEN
$    WRITE dat "''val' ''sym'"
$    dat="Data Present"
$  ENDIF
$ GOTO GenIA64loop
$ EndIA64Vec:
$!
$!  Generate a match control string. Note required spaces at beginning
$!  of line to keep it at the top of the vector file even after sorting
$!  Add comment tag to indicate if data is present
$!
$ ALWAYS="ALWAYS"
$ EQUAL="EQUAL"
$ LESSEQUAL="LEQUAL"
$ NEVER="NEVER"
$ match="  GSMATCH="+'algo'+",''maj',''min' ! "+dat
$ write out match
$ SET NOON
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ SORT TMP_'pid'.VEC  'name'.VEC
$ SORT TMP_'pid'.DATA 'name'.DATA
$ CleanupIA64:
$ err=$status
$ IF F$TRNLNM("IN").NES."" THEN CLOSE/NOLOG in
$ IF F$TRNLNM("OUT").NES."" THEN CLOSE/NOLOG out
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG dat
$ DELETE TMP_'pid'*.*;*
$ IF .NOT.err THEN EXIT
$ IF endphase.NES."VECTOR" THEN GOTO GEN
$ EXIT
$
$ NoIA64Vec: $status=4
$  WRITE SYS$OUTPUT "Error - could not find symbol vector"
$ GOTO CleanupIA64
$
$ BadIA64Img: WRITE SYS$OUTPUT "Can't parse image ''what' -> ''line'"
$ EXIT
$!
$ GEN:
$ VecEntrySize=16
$ IF arch.EQS."IA64" THEN VecEntrySize=8
$!
$!  Generate MACRO code and options file for the fake image
$!
$ v=F$PARSE("SYS$DISK:[]''name'.VEC")
$ IF F$SEARCH(v).EQS.""
$ THEN
$   WRITE SYS$OUTPUT "Cannot find file ''name'.VEC"
$   EXIT
$ ENDIF
$ IF .NOT.(exp.OR.force.OR.F$SEARCH("FAKE_''name'.MAR").EQS."") THEN GOTO COMPILE
$ WRITE SYS$OUTPUT "Generating MACRO code for ''name'"
$ ON WARNING THEN GOTO VecCleanup
$ ON CONTROL_Y THEN GOTO VecCleanup
$!
$!  Parse vector file
$!
$ OPEN/READ vec 'v'
$ READ vec match    ! First record is the match control string
$ n=F$PARSE(v,,,"NAME")
```

```
$ OPEN/WRITE opt FAKE_'n'.OPT
$!
$!  Options file contains:
$ WRITE opt match          ! Match control string
$ WRITE opt "FAKE_''n'"          ! reference to FAKE_''n' object module
$ WRITE opt "FAKE_RTL/SHARE"     ! FAKE_RTL support image
$!
$!  Check to see if data was found in the source image
$ hasdata=F$LOCATE("Data Present",match).LT.F$LENGTH(Match)
$!
$!  MACRO source contains a title
$ OPEN/WRITE mac FAKE_'n'.MAR
$ WRITE mac " .TITLE FAKE_''n'"
$ WRITE mac " .PSECT RWData,RD,WRT,NOEXE,PAGE" ! read/write psect
$ IF hasdata
$ THEN       ! If data present, we need to map it as an init routine
$   WRITE mac "     .EXTRN LIB$INITIALIZE"
$   WRITE mac "     .PSECT
LIB$INITIALIZE,NOPIC,CON,REL,GBL,NOSHR,NOEXE,NOWRT,LONG"
$   WRITE mac "     .LONG MapData"
$ ENDIF
$ WRITE mac " .PSECT ROData,RD,NOWRT,NOEXE,PAGE"      ! read only psect
$!
$!  string containing the name of the real shareable image
$ WRITE mac " .ALIGN LONG"
$ WRITE mac " RealRTL: .ASCID /REAL_''n'/"
$!
$!  Macros to define a normal routine call and a JSB call
$!
$ COPY SYS$INPUT mac
$ DECK
       .PSECT $CODE,RD,NOWRT,EXE,PAGE

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

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

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

```
          MOVQ R20,-(SP)
          MOVQ R19,-(SP)
          MOVQ R18,-(SP)
          MOVQ R17,-(SP)
          MOVQ R16,-(SP)
         PUSHAL 'Pref'A_%EXTRACT(0,28,Routine) ; address of routine
         PUSHAB 'Pref'S_%EXTRACT(0,28,Routine) ; name of routine
         PUSHAB RealRTL   ; name of real shareable image
         CALLS #15,G^FAKE_LOGJSB ; dispatch to routine
          RET
        .ENDM
$ EOD
$ abort="FALSE" ! assume all will work
$ NextAddr=0  ! set expected next vector address
$ vec="SYMBOL_VECTOR=("    ! header for symbol vector entries
$!
$!  Walk through the file
$ vecloop: READ/END=EndVec vec rec
$  Addr=%X'F$ELEMENT(0," ",rec)'  ! get address
$  v1=%X'F$ELEMENT(1," ",rec)'           ! get vector values
$  v2=%X'F$ELEMENT(2," ",rec)'          !
$  SName=F$ELEMENT(3," ",rec)           ! get symbol name
$  Typ=F$ELEMENT(4," ",rec)             ! get symbol type
$  IF F$INTEGER(Addr).EQ.F$INTEGER(NextAddr) THEN GOTO AddrOK
$!
$!  Not at expected location
$!
$    IF Addr.LT.NextAddr
$    THEN
$      ! This should never happen
$      WRITE SYS$OUTPUT -
       "FATAL ERROR - vector overlap. Expecting ''NextAddr', got ''Addr'"
$      WRITE SYS$OUTPUT "Jacketing not feasible!"
$      abort="TRUE"
$      GOTO EndVec
$    ENDIF
$!
$    WRITE SYS$OUTPUT -
        "Warning - vector hole before ''sname'. Expecting ''NextAddr', got
''Addr'"
$!
$!  Fill the vector with "spare" entries up to next entry
$!
$    FillLoop: IF NextAddr.GE.Addr THEN GOTO AddrOK
$      va=F$FAO("!XL",NextAddr)
$      NextAddr=NextAddr+VecEntrySize
$      WRITE opt "''vec'SPARE) ! ''va' ***Fill to ''rec'"
$    GOTO FillLoop
$ AddrOK:
$  NextAddr=NextAddr+VecEntrySize ! reset next expected address
$  GOSUB GenSym            ! generate the symbol name
$  IF typ.EQS."(procedure)"
$  THEN
$!    Options file entry is a PROCEDURE type symbol vector entry
$!
$    WRITE opt "''vec'''Lbl'=PROCEDURE)  ! ''Addr'"
$!
$!    OpenVMS naming convention is that JSB routines end with "_Rn" where
$!    "n" is the highest register modified. Check to see if this routine
$!    name ends in _Rn. If it does, generate a JSB entry instead of a
$!    CALL entry and issue a message.
$!
$    tag=F$EXTRACT(F$LENGTH(sName)-3,3,sName)
$    rn=tag-"_R"
$    IF f$EXTRACT(0,2,tag).EQS."_R".AND.(rn.GT.0).AND.(rn.LT.10)
$    THEN
$      WRITE SYS$OUTPUT "Assumed JSB routine ''sName'"
$      WRITE mac "JSBRoutine ''sName', ''Lbl', ''Pref' ; ''Addr'"
```

```
$    ELSE
$      WRITE mac "CallRoutine ''sName', ''Lbl', ''Pref' ; ''Addr'"
$    ENDIF
$!
$  ELSE IF typ.EQS."(constant)"
$  THEN
$!     Options file entry is a DATA symbol vector entry
$!
$    WRITE opt "''vec'''Lbl'=DATA)  ! ''Addr' CONST"
$!
$!     MACRO code is jusr a global symbol definition
$    WRITE mac "'''Lbl'==''v2' ; ''Addr'"
$!
$  ELSE IF typ.EQS."(datacell)"
$  THEN
$!     Options file entry is a DATA symbol vector entry
$!
$    WRITE opt "''vec'''Lbl'=DATA)  ! ''Addr' VAR"
$!
$!     Code will be generated later... see below
$!
$  ELSE IF typ.EQS."(spare)"
$  THEN
$      WRITE opt "''vec'SPARE) ! ''addr' SPARE"
$  ELSE
$!
$!     Shouldn't happen - flag the record
$    WRITE SYS$OUTPUT "Error unexpected vector entry type /''rec'/"
$  ENDIF
$  ENDIF
$  ENDIF
$  ENDIF
$ GOTO vecloop
$ EndVec:
$!
$!  Finished with vector file (input) and options file (output)
$ CLOSE vec
$ CLOSE opt
$ IF abort THEN EXIT
$ IF .NOT.hasdata THEN GOTO Finish
$!
$!    Generate code for data - we read the data listing, which is sorted
$!    on offset
$!
$   d=F$PARSE("SYS$DISK:[]''name'.DATA")
$   psize=8192      ! Page size in bytes (Alpha only!)
$   OPEN/READ dat 'd'
$
$!
$!    PSECT for exported data, aligned on Alpha page boundary
$   WRITE mac ".PSECT MapData,RD,WRT,NOEXE,13"
$   WRITE mac ".ALIGN 13"
$!
$!    Loop through data symbols
$!
$   READ/END=EndDat dat rec
$   b=0
$BlockLoop:  !
$     b=b+1
$     WRITE SYS$OUTPUT "Start of data block ''b' ''rec'"
$     WRITE mac "map''b'_start:" ! label at start of data block
$     v=F$ELEMENT(0," ",rec)     ! Get offset
$     val=%X'v'                  ! translate to decimal
$     i=val-(val/psize*psize)    ! check alignment from beginning of page
$     base=val-i          ! calculate base offset
$     hwm=val             ! set high water mark
$     IF i.GT.0 THEN WRITE mac ".BLKB ''i'"    ! pad if not aligned
$     sname=F$ELEMENT(1," ",rec)! get symbol name
```

```
$     basesym'b'=sname           ! generate name for string
$     DatLoop:                   ! loop on data entries
$       READ/END=EndDat dat rec  ! get next entry
$       v1=F$ELEMENT(0," ",rec)  ! get offset
$       val1=%X'v1'              ! convert to decimal
$       GOSUB GenSym             ! generate symbol name
$       alloc=""                 ! assume no allocation
$       delta=val1-val           ! calculate size
$!           If entry is non zero set allocation
$       IF delta.GT.0.AND.delta.LT.psize THEN alloc=".BLKB ''delta'"
$       WRITE mac "''Lbl':: ''alloc'    ;''v'" ! declare symbol
$       v=v1                     ! set for next record offset (HEX)
$       val=val1                 ! set next record (decimal)
$       sname=F$ELEMENT(1," ",rec)      ! next record name
$       IF delta.GT.psize  ! Delta larger than page, start next block
$       THEN
$         WRITE SYS$OUTPUT "Hole in data exceeded threshold at ''rec'"
$        GOSUB EndSec              ! Fill end of data section
$         WRITE mac ".ALIGN 13"
$         GOTO BlockLoop   ! next block
$       ELSE
$         hwm=val   ! set highwater mark
$       ENDIF
$   GOTO DatLoop
$
$ GenSym:
$!   This is needed to deal with upper and lowercase symbol names
$!   since MACRO cannot generate lowercase names, and images like DECC$SHR
$!   contain symbols which differ only in case, we need to generate "fake"
$!   names for lowercase symbols that won't clash with uppercase.
$!   Here we do that by prefixing the symbol with "L_" and truncating at
$!   31 characters. Uppercase only names go straight through.
$!
$   IF F$EDIT(sName,"UPCASE").NES.sName
$   THEN
$     Pref="L"
$     Lbl=F$EXTRACT(0,31,"''Pref'_''sName'")
$   ELSE
$     Pref="U"
$     Lbl=sName
$   ENDIF
$ RETURN
$
$ EndSec:    ! End of data section
$   size=hwm-base   ! calculate total data size
$   pages=(size+psize)/psize     ! work out size in pages
$   pages'b'=pages  ! save per block
$   extra=pages*psize-size ! calculate padding at end
$   alloc=""
$   IF extra.GT.0 THEN alloc=".BLKB ''extra'"
$   WRITE mac "map''b'_pad: ''alloc'"
$   WRITE mac "map''b'_end:"     ! label at end of section
$ RETURN
$
$ EndDat:    ! End of all data
$   CLOSE dat ! close input file
$   GOSUB GenSym
$   WRITE mac "''Lbl'::    ;''v'" ! define last label
$   GOSUB EndSec    ! fill end of section
$   WRITE mac "mapeop: .LONG"    ! mark end of all data
$!
$!   Now generate init routine
$!
$ WRITE mac " .PSECT $CODE        "
$ WRITE mac " .ENTRY MapData,^M<>"
$   c=1      ! for each data block
$   gsdloop:
$       WRITE mac ";MAP GLOBAL SECTION ''n'''c'_DATA" ! comment
```

```
$       sym=basesym'c'     ! get symbol name
$       WRITE mac " .PSECT RODataa"      ! define name of global section
$      WRITE mac " .ALIGN LONG"
$       WRITE mac " gsd''c':.ASCID /''n'''c'_DATA/"
$      WRITE mac " .ALIGN LONG"   ! define name of symbol in
$!                                ! real image at start of block
$       WRITE mac " bsym''c':.ASCID /''sym'/"
$
$       pagelets=pages'c'*16      ! calculate pagelet size
$       bytes=pages'c'*psize      ! calculate size in bytes
$      WRITE mac " .PSECT $CODE"
$      WRITE mac " PUSHAL map''c'_end"         ! address of end
$      WRITE mac " PUSHAL map''c'_start"! address of start
$      WRITE mac " PUSHL #''pagelets'"         ! size
$      WRITE mac " PUSHAB bsym''c'"            ! start symbol
$      WRITE mac " PUSHAB gsd''c'"             ! section name
$      WRITE mac " PUSHAB RealRTL"             ! real image name
$      WRITE mac " CALLS #6,G^FAKE_MAP_DATA"   ! map data
$       c=c+1       ! repeat for next section
$   IF c.LE.b THEN GOTO gsdloop
$   WRITE mac "RET" ! end of init routine
$ Finish:
$ WRITE mac " .END"  ! end of macro module
$ CLOSE mac
$ IF endphase.NES."VECTOR" THEN GOTO COMPILE
$ EXIT
$!
$ VecCleanup:       ! close any files that might be open
$ IF F$TRNLNM("DAT").NES."" THEN CLOSE/NOLOG DAT
$ IF F$TRNLNM("MAC").NES."" THEN CLOSE/NOLOG MAC
$ IF F$TRNLNM("VEC").NES."" THEN CLOSE/NOLOG VEC
$ IF F$TRNLNM("OPT").NES."" THEN CLOSE/NOLOG OPT
$ EXIT
$!
$ COMPILE:
$   IF exp.OR.force.OR.F$SEARCH("FAKE_''name'.OBJ").EQS.""
$   THEN
$     WRITE SYS$OUTPUT "Compiling FAKE_''name'"
$     opt=""
$     IF dbg THEN opt="/NOOPT/DEBUG"
$     MACRO'opt' FAKE_'name'
$   ENDIF
$ IF endphase.EQS."COMPILE" THEN EXIT
$ LINK:
$   IF exp.OR.force.OR.F$SEARCH("FAKE_''name'.EXE").EQS.""
$   THEN
$     WRITE SYS$OUTPUT "Linking FAKE_''name'"
$     opt=""
$     IF dbg THEN opt="/DEBUG"
$     LINK/SHARE'opt' FAKE_'name'/OPT
$   ENDIF
$ IF endphase.EQS."LINK" THEN EXIT
$ USE:
$   DEFINE/NOLOG REAL_'name' 'img'
$   fake_img=F$ELEMENT(0,";",F$SEARCH("FAKE_''name'.EXE"))
$   fake_'name'=="DEFINE/USER/NOLOG/NAME=CONFINE ''name' ''fake_img'"
$   real_'name'=="DEASSIGN ''name'"
$ EXIT
$
$ CreateOpt:
$!  Options file for FAKE_RTL support routines
$!
$ CREATE/LOG FAKE_RTL:.OPT
$ DECK
GSMATCH=LEQUAL,42,1 ! arbitrary GSMATCH
FAKE_RTL            ! link FAKE_RTL.OBJ
REAL_LIBRTL/SHARE   ! link against real LIBRTL and LIBOTS (important!)
REAL_LIBOTS/SHARE
```

```
SYMBOL_VECTOR=(-
FAKE_DOCALL=PROCEDURE,-          ! dispatch to routine by CALL
FAKE_LOGCALL=PROCEDURE,-   ! log argument list and dispatch by CALL
FAKE_DOJSB=PROCEDURE,-           ! dispatch to routine by JSB
FAKE_LOGJSB=PROCEDURE,-          ! log argument list and dispatch by JSB
FAKE_PUT=PROCEDURE,-             ! write a string to log file
FAKE_FIS=PROCEDURE,-             ! jacket for LIB$FIND_IMAGE_SYMBOL
FAKE_CALL=PROCEDURE,-            ! jacket for CALLG (OTS$EMUL_CALL)
FAKE_MOVE=PROCEDURE,-            ! jacket for MOVC3
FAKE_OUT=PROCEDURE,-             ! jacket for LIB$PUT_OUTPUT
FAKE_LOG=PROCEDURE,-             ! write to log file if logging enabled
FAKE_MAP_DATA=PROCEDURE-   ! map data area
)

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

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

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

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


       .PSECT RWData
.align long
```

```
tracing:.LONG 0 ; set to 1 if tracing enabled
skip:   .LONG 0     ; block recursion
depth:  .LONG 0 ; nesting depth
exhsta: ;    exit handler
       .LONG 1      ; buffer for final status
desblk: .LONG 0 ; Exit handler descriptor block
hndadr: .LONG 0 ; handler address
argc:   .LONG 1 ; argument count
staadr: .ADDRESS exhsta ; pointer to final status buffer
        .LONG 0

.ALIGN QUAD         ; FAB and RAB for output file
LogFAB:      $FAB FNM=ARGDUMP , DNM=<SYS$DISK:[].LOG> , RAT=CR
LogRAB: $RAB FAB=LogFAB

       ; desriptors for formatting strings
outlen: .WORD
.ALIGN QUAD
outdsc: .LONG ^X010E0000
       .ADDRESS outbuf
dscdsc: .LONG ^X010E0000
       .ADDRESS dscbuf

MaxPad=32    ; padding string for start of log strings
Padding: .ASCID /                                                /

maxstr=48
outmax=128                ; buffer length for strings
outbuf: .BLKB outmax
dscbuf: .BLKB outmax
$LIBDEF

;
;  Macro to format and output a string using FAO
;  Arguments are the FAO control string and corresponding FAO arguments

       .MACRO formout formstr, a1=#0,a2=#0,a3=#0,a4=#0,a5=#0,a6=#0,?OK
         MOVW Depth,Padding       ; set length of pad string
         CMPW #MaxPad,Depth       ; check max deptch
         BGTR OK
         MOVW #MaxPad,Padding     ; limit to macimum
       'OK':
         MOVW #outmax,outdsc      ; set descriptor to maximum size
         $FAO_S CTRSTR=formstr, OUTLEN=outlen, OUTBUF=outdsc, -
              P1=#Padding,P2=a1,P3=a2,P4=a3,P5=a4,P6=a5,p7=a6
         MOVW outlen,outdsc       ; set descriptor to actual size
         PUSHAB outdsc
         CALLS #1,FAKE_PUT ; write to log file
       .ENDM

       .PSECT $CODE

       .ENTRY FAKE_EXIT,^M<>      ;  Exit handler
       BLBC tracing,NoClose       ;    skip if not tracing
         formout eol,exhsta       ;    Format final status string
         $CLOSE FAB=LogFAB ;    Close log file
NoClose:RET

       .ENTRY FAKE_INIT,^M<>      ; setup log file
       PUSHAB dumpargs            ; check logical name to see if tracing enabled
       CALLS #1,G^LIB$GET_LOGICAL ; existence only
       BLBC R0,NoTrace
         MOVL #1,tracing   ; set tracing flag
         $CREATE FAB=LogFAB       ; Create and connect log file
          BLBC R0,fail
         $CONNECT RAB=LogRAB
         BLBC R0,fail
         MOVAB FAKE_EXIT,hndadr ;    init exit handler control block
```

```
        PUSHAB desblk
        $DCLEXH_S DESBLK=desblk  ; declare exit handler
        formout start            ; write start message
NoTrace:RET

       .MACRO GetRoutineAddress,?Ready
       ;  macro to check if a routine address is already known, and find
       ;  it if not. Note we must preserve all registers!

         CMPL #0,@12(AP)   ; is address zero?
         BNEQ Ready        ; no, already known
           PUSHR #^M<R0,R1>       ; save R0 & R1
             PUSHAL @12(AP)       ;  address buffer
             PUSHAL @8(AP) ;  symbol name
             PUSHAL @4(AP) ;  image name
             CALLS #3,G^LIB$FIND_IMAGE_SYMBOL
           POPR  #^M<R0,R1>       ; restore registers
        'ready':
         PUSHL  @12(AP)    ; return address on stack
       .ENDM

       .MACRO LogReturn,?done
       ;  macro to log the return from a routine
       ;  we must preserve all registers
         BLBC tracing,done ; skip if not tracing
          BLBS skip, done  ; recursion block
           MOVL #1,skip    ; protect from recursion
             PUSHR #^M<R0,R1>     ; save R0 & R1
               PUSHL R0    ; log return value
               PUSHAB @8(AP)      ; routine name
               CALLS #2,FAKE_RETURN      ; write return record
             POPR  #^M<R0,R1>     ; restore registers
            CLRL skip             ; unblock recursion
         'done':
       .ENDM


        .CALL_ENTRY,MAX_ARGS=4,HOME_ARGS=TRUE,LABEL=FAKE_DOCALL
;
; Args
;      Image Name
;       Routine Name
;       Address of Routine Address
;      AP
;
       GetRoutineAddress
       CALLG @16(AP),@(SP)+       ; note weird mode!
       RET

        .CALL_ENTRY,MAX_ARGS=4,HOME_ARGS=TRUE,LABEL=FAKE_LOGCALL
;
; Args
;      Image Name
;       Routine Name
;       Address of Routine Address
;      AP
;
       BLBC tracing,DoneEntry     ; skip logging if not tracing
       BLBS skip, DoneEntry       ; skip logging if recursing
         MOVL #1,skip             ; block recursion
           PUSHR #^M<R0,R1>       ; preserve registers
             PUSHL #1
             PUSHAL @16(AP)       ; arg list
             PUSHAB @8(AP) ; routine name
             CALLS #3,FAKE_DUMPARGS
           POPR  #^M<R0,R1>       ; restore registers
         CLRL  skip        ; unblock recursion
        DoneEntry:
```

```
        ADDW2 #2,depth              ; increment nesting detch
        GetRoutineAddress   ; get address
        CALLG @16(AP),@(SP)+        ; dispatch
        SUBW2 #2,depth              ; restore nesting depth
        BLBC tracing,DoneExit       ; skip logging if not tracing
        BLBS skip, DoneExit ; skip logging if recursing
          MOVL #1,skip              ; block recursion
            PUSHR #^M<R0,R1>        ; preserve registers
              PUSHL #0
              PUSHAL @16(AP)        ; arg list
              PUSHAB @8(AP) ; routine name
              CALLS #3,FAKE_DUMPARGS
            POPR  #^M<R0,R1>        ; restore registers
          CLRL  skip        ; unblock recursion
         DoneExit:
        LogReturn
        RET


          .CALL_ENTRY,MAX_ARGS=16,HOME_ARGS=TRUE,LABEL=FAKE_DOJSB
;
; Args
;       Image Name
;        Routine Name
;        Address of Routine Address
;       R16..R21 as quadwords
;
        GetRoutineAddress
        MOVQ 16(AP),R16     ; restore register arguments
        MOVQ 24(AP),R17
        MOVQ 32(AP),R18
        MOVQ 40(AP),R19
        MOVQ 48(AP),R20
        MOVQ 56(AP),R21
         JSB @(SP)+  ; dispatch to routine
        RET

          .CALL_ENTRY,MAX_ARGS=16,HOME_ARGS=TRUE,LABEL=FAKE_LOGJSB
;
; Args
;       Image Name
;        Routine Name
;        Address of Routine Address
;       R16..R21 as quadwords
;

        BLBC tracing,DoneJSBEntry  ; skip if not tracing
        BLBS skip, DoneJSBEntry          ; skip if recursing
          MOVL #1,skip                   ; block recursion
            PUSHR #^M<R0,R1>            ; save registers
              PUSHAQ 16(AP)             ; pass address of first register arg
              PUSHAB @8(AP)         ; routine name
              CALLS #2,FAKE_JSBENTRY      ; log the entry
            POPR  #^M<R0,R1>              ; restore registers
          CLRL  skip                ; unblock recursion
         DoneJSBEntry:
        GetRoutineAddress
        ADDW2 #2,depth              ; increment nesting depth
        MOVQ 16(AP),R16             ; restore register args
        MOVQ 24(AP),R17
        MOVQ 32(AP),R18
        MOVQ 40(AP),R19
        MOVQ 48(AP),R20
        MOVQ 56(AP),R21
         JSB @(SP)+                 ; dispatch to routine
        SUBW2 #2,depth              ; restore nesting depth
        LogReturn
```

```
        RET

        .ENTRY FAKE_RETURN,^M<r2,r3>
; writes routine name, return value and time stamp
; Arguments
;   4(AP) = String descriptor, routine name
;   8(AP) = return value
;
          PUSHR #^M<R0,R1>  ; save registers
            formout return,4(AP),8(AP),#0
          POPR #^M<R0,R1>   ; restore registers
          formout regs1,r0,r1,r2,r3,r4,r5 ; dump registers
          formout regs2,r6,r7,r8,r9,r10,r11
          formout blank
        RET


        .ENTRY FAKE_JSBENTRY,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>

        PUSHR #^M<R0,R1>
          formout blank
          formout JSBhead,4(AP)   ; write header
        POPR #^M<R0,R1>
        formout regs1,r0,r1,r2,r3,r4,r5         ; dump registers
        formout regs2,r6,r7,r8,r9,r10,r11
        MOVL 8(AP),R10      ; get address of first register arg
        CLRL R9                     ; reset arg counter
        .DISABLE FLAGGING   ; supress  QUADMEMREF informationals
        MOVQ (R10)+,R2              ; dump arguments
        JSB DumpArg
        MOVQ (R10)+,R2
        JSB DumpArg
        MOVQ (R10)+,R2
        JSB DumpArg
        MOVQ (R10)+,R2
        JSB DumpArg
        MOVQ (R10)+,R2
        JSB DumpArg
        MOVQ (R10)+,R2
        .ENABLE FLAGGING    ; restore informationals
        JSB DumpArg
        RET

        .ENTRY FAKE_DUMPARGS,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10>
;
; FAKE_DUMPARGS  dumps argument list
; Arguments
;   4(AP) = String descriptor, routine name
;   8(AP) = Argument pointer
;  12(AP) = Flag 1=write header
;
        BLBC 12(AP),NoHead
          PUSHR #^M<R0,R1>
            formout blank
            formout header,4(AP)    ; write header (routine name & timestamp)
          POPR #^M<R0,R1>
          formout regs1,r0,r1,r2,r3,r4,r5 ; dump registers
          formout regs2,r6,r7,r8,r9,r10,r11
        NoHead:
        MOVL   8(AP),R8             ; get argument pointer
        PROBER #0,#4,(R8)   ; readable?
        BNEQ   DoArgs       ; Arg list is readable
         formout argcnt,4(AP),#0 ; zero args passed or unreadable argument list
        RET
DoArgs:
        MOVL   (R8)+,R7             ; get argument count
        CLRL   R9           ; init counter
        BLBC 12(AP),retargs
```

```
        formout argcnt,4(AP),R7
        BICL   #^XFFFFFF80,R7    ; sanity check to 127 args
        BRB ArgLoop
      retargs:
         formout argret,4(AP),R7
        BICL   #^XFFFFFF80,R7    ; sanity check to 127 args

       ArgLoop:
        SOBGEQ R7,MoreArg ; count down arguments
          RET
         MoreArg:
        MOVL (R8)+,R2           ; get next arg
        JSB DumpArg
      BRB ArgLoop



DumpArg:     .JSB_ENTRY input=<r2,r9>,output=<r9>
;
; arg value in r2
; arg number is in R9 (pre incremented and returned)
;
; Arguments are written in HEX.
; If address is valid, reference value will be written as a hex longword.
; If address looks like a valid descriptor the data will be converted to a
;  string written
; If address looks like a printable ASCII string it will be written to
;  a maximum of MAXSTR characters

      INCL R9             ; update counter
       PROBER #0,#4,(R2)  ; readable?
      BEQL ByValue        ; no, assume by value

            ; readable, assume by reference
      formout refarg,r9,r2,(r2) ; write initial reference
;
;  We now want to see if the argument could be a descriptor. Use LIB$CVT_DX_DX
;  to convert to string. If the call succeeds we had a valid descriptor and
;  now have a string to display.
;  In the vast majority of cases this will be a string descriptor, but
;  LIB$CVT_DX_DX gives us all other scalar types for free.
;
      MOVW #maxstr,dscdsc ; set descriptor to maximum size
      PUSHAW outlen       ; variable to receive output length
      PUSHAB dscdsc       ; output descriptor
      PUSHL  R2           ; input descriptor?
      CALLS #3,G^LIB$CVT_DX_DX ; convert
      BLBS R0,IsDescr     ; conversion succeeded, valid descriptor
      CMPL R0,#LIB$_OUTSTRTRU    ; conversion OK, but truncated
      BEQL IsDescr

      ; NotDescriptor, might be a string
       CLRL R3            ; init counter
      MOVL R2,R4    ; get copy of start address
       CMPB (R4),#32      ; GEQ space character?
      BLSSU finarg  ; no, skip arg
      CMPB (R4)+,#126     ; printable? (and step to next character)
      BGTRU finarg  ; no, skip arg
chrloop:INCL R3           ; we now have at least one printable character
      PROBER #0,#4,(R4) ; check next character is readable
      BEQL str      ; if not, write what we've found as a string
      CMPB (R4),#0  ; NUL?
      BEQL AZstr    ; assume ASCIZ and write it
      CMPL R3,#maxstr     ; more than threshold?
      BGEQ longstr  ; yes, write maximum string
      CMPB (R4),#32 ; still printable?
      BLSSU str     ; no, write what we've found
      CMPB (R4)+,#126     ; still printable?
```

```
       BGEQU str    ; no, write what we've found
       BRB chrloop  ; keep looking


IsDescr:; valid descriptor
       MOVW outlen,dscdsc  ; set output descriptor to actual length
       formout dscarg,4(R2),#dscdsc ; output string
       RSB

str: ; string argument size in R3, address in R2
       formout strarg,R3,R2
       RSB

longstr: ; long string size in R3, address in R2
       formout lstarg,R3,R2
       RSB

AZstr:  ; nul terminated string, address in R2
       formout azsarg,R2
        RSB

ByValue: ; assume argument by value. count in R9, value in R2
       formout valarg,r9,r2
finarg:      RSB

;
; Routines to write the log file. Name is by logical name FAKE_ARGDUMP
; default is in current default directory, type ".LOG". File is created
; in FAKE_INIT and closed on image exit. Final status is written
; before closing.

       .ENTRY FAKE_PUT,^M<R2,R3,R4,R5,R6>
;
;    Write one string to file, argument passed by descriptor
;
       MOVL 4(AP),R1 ; get desriptor address
       MOVW (R1), LogRAB+RA
B$W_RSZ      ; set size
       MOVL 4(R1),LogRAB+RAB$L_RBF      ; set buffer address
       $PUT RAB=LogRAB     ; write record
       RET

       .ENTRY FAKE_LOG,^M<R2,R3,R4,R5,R6>
;
;    Write one string to file, if logging enabled argument passed by descriptor
;
       BLBC tracing,NoLog
         MOVL 4(AP),R1     ; get desriptor address
         MOVW (R1), LogRAB+RAB$W_RSZ     ; set size
         MOVL 4(R1),LogRAB+RAB$L_RBF     ; set buffer address
         $PUT RAB=LogRAB   ; write record
NoLog: RET

fail:  $EXIT_S R0   ; error exit
       RET

       .PSECT RWdata
fadr_s:      .LONG  ; data address range for fake image
fadr_e:      .LONG
radr_s:      .LONG  ; data address range for real image
radr_e:      .LONG

       .PSECT $CODE
       .ENTRY FAKE_MAP_DATA,^M<>
;
;      Routine to map data. A range of new addresses is created as a global
;      section. Data is copied from the old range to the global section, then
;      the old range is mapped to the section.
```

```
; Args
;      Image Name
;       Section name
;      base symbol name
;       Number of pages
;      start address
;      end address
;
       MOVL 20(AP),fadr_s  ; set fake address range
       MOVL 24(AP),fadr_e
       DECL fadr_e         ; adjust to last byte on previous page
       $DELTVA_S inadr=fadr_s  ; delete virtual addresses
       BLBC R0,MapFail
;
;             Create global section
       $CRMPSC_S inadr=fadr_s, gsdnam=@8(AP),pagcnt=16(AP),-
                 flags=#SEC$M_GBL!SEC$M_PAGFIL!SEC$M_WRT
       BLBC R0,MapFail
       PUSHAL radr_s       ; find start of data
       PUSHAB @12(AP)              ; symbol name
       PUSHAB @4(AP)       ; image name
       CALLS #3,G^LIB$FIND_IMAGE_SYMBOL
       BICL  #^X01FFF,radr_s      ; page align
       MOVL  16(AP),radr_e ; get data size in pagelets
       MULL2 #512,radr_e   ; convert to bytes
       MOVC3 radr_e,@radr_s,@fadr_s ; copy data
       ADDL2 radr_s,radr_e ; calculate end address
       DECL  radr_e        ; adjust to last byte on previous page
       $DELTVA_S inadr=radr_s     ; delete address range
       BLBC R0,MapFail
;             map to section
       $MGBLSC_S inadr=radr_s, gsdnam=@8(AP), -
                 flags=#SEC$M_GBL!SEC$M_PAGFIL!SEC$M_WRT
       BLBC R0,MapFail
       BLBC tracing,NoLogMap      ; If logging enabled, log mapping
         formout MapData,8(AP),radr_s,radr_e,fadr_s,fadr_e
       MOVL radr_s,R0              ; return real base address
NoLogMap:RET
MapFail: $EXIT_S R0
       RET

;
; Jacket routines.
; These are placed in this module to allow "fake" RTLs to access them, even
; if it would imply a self reference. For example, a fake LIBRTL cannot call
; LIB$FIND_IMAGE_SYMBOL directly. This image is linked against REAL_LIBRTL and
; REAL_LIBOTS so it always uses the "real" routine.
;
;  LIBRTL routines
         .CALL_ENTRY,MAX_ARGS=8,HOME_ARGS=TRUE,LABEL=FAKE_FIS
       CALLG (AP),G^LIB$FIND_IMAGE_SYMBOL
       RET
         .CALL_ENTRY,MAX_ARGS=8,HOME_ARGS=TRUE,LABEL=FAKE_OUT
       CALLG (AP),G^LIB$PUT_OUTPUT
       RET
;
;  LIBOTS emulated instructions.
         .CALL_ENTRY,MAX_ARGS=2,HOME_ARGS=TRUE,LABEL=FAKE_CALL
       CALLG @8(AP),@4(AP)
       RET
         .CALL_ENTRY,MAX_ARGS=3,HOME_ARGS=TRUE,LABEL=FAKE_MOVE
       MOVC3 4(AP),@8(AP),@12(AP)
       RET

       .END
$ EOD
$ RETURN
```

For more information

OpenVMS Linker Utility Manual, Order Number: AA-PV6CD—TK
http://h71000.www7.hp.com/doc/73final/4548/4548PRO.HTML

HP OpenVMS Calling Standard, Order Number: AA-QSBBE-TE
http://h71000.www7.hp.com/doc/82final/5973/5973PRO.HTML

HP OpenVMS Programming Concepts Manual
http://h71000.www7.hp.com/doc/82FINAL/5841/5841PRO.HTML

HP OpenVMS RTL Library (LIB$) Manual, Order Number: AA-QSBHE-TE
http://h71000.www7.hp.com/doc/82final/5932/5932PRO.HTML

HP OpenVMS MACRO Compiler Porting and User's Guide, Order Number: AA-PV64E-TE
http://h71000.www7.hp.com/doc/82final/5601/5601PRO.HTML

John Gillings
Software Systems Consultant, OpenVMS Ambassador
Hewlett-Packard Pty Limited
OpenVMS Group, Customer Support Centre
Sydney, Australia

`OpenVMS homepage:`	http://www.hp.com/products/openvms
`OpenVMS Times:`	http://www.hp.com/products1/evolution/customertimes
`OpenVMS Patches:`	http://itrc.hp.com/
`OpenVMS Forum:`	http://forums.itrc.hp.com/service/forums/familyhome.do?familyId=288