[an error occurred while processing this directive]

HP OpenVMS Systems Documentation

HP OpenVMS Programming Concepts Manual

Chapter 26
Shareable Resources

This chapter describes the techniques available for sharing data and program code among programs. It contains the following sections:

Section 26.1 describes how to share code among programs.

Section 26.2 describes shareable images.

Section 26.3 defines and describes how to use local and global symbols to share images.

The operating system provides the following techniques for sharing data and program code among programs:

- DCL symbols and logical names
- Libraries
- Shareable images
- Global sections
- Common blocks installed in a shareable image
- OpenVMS Record Management Services (RMS) shared files

Symbols and logical names are also used for intraprocess and interprocess communication; therefore, they are discussed in Chapter 34.

Libraries and shareable images are used for sharing program code.

Global sections, common blocks stored in shareable images, and RMS shared files are used for sharing data. You can also use common blocks for interprocess communication. For more information, refer to Chapter 3.

26.1 Sharing Program Code

To share code among programs, you can use the following operating system resources:

- Text, macro, or object libraries that store sections of code. Text and macro libraries store source code; object libraries store object code. You can create and manage libraries using the Librarian utility (LIBRARIAN). Refer to the *HP OpenVMS Command Definition, Librarian, and Message Utilities Manual* for complete information about using the Librarian utility.
- Shareable images are images that can be linked with executable images. These images can also be stored in libraries.

26.1.1 Object Libraries

You can use object libraries to store frequently used routines, thereby avoiding repeated recompiling, which allows you to minimize the number of files you must maintain, and simplify the linking process. The source code for the object modules can be in any supported language, and the object modules can be linked with any other modules written in any supported language.

Use the .OLB file extension for any object library. All modules stored in an object library must have the file extension .OBJ.

26.1.1.1 System- and User-Defined Default Object Libraries

The operating system provides a default system object library, STARLET.OLB. You can also define one or more default object libraries to be automatically searched before the system object library. The logical names for the default object libraries are LNK$LIBRARY and LNK$LIBRARY_1 through LNK$LIBRARY_999. To use one of these default libraries, first define the logical name. The libraries are searched sequentially starting at LNK$LIBRARY. Do not skip any numbers. If you store object modules in the default libraries, you do not have to specify them at link time. However, you do have to maintain and manage them as you would any library.

The following example defines the library in the file PROCEDURES.OLB (the file type defaults to .OLB, meaning object library) in $DISK1:[DEV] as a default user library:

```
$ DEFINE LNK$LIBRARY $DISK1:[DEV]PROCEDURES
```

26.1.1.2 How the Linker Searches Libraries

When the linker is resolving global symbol references, it searches user default libraries at the process level first, then libraries at the group and system level. Within levels, the library defined as LNK$LIBRARY is searched first, then LNK$LIBRARY_1, LNK$LIBRARY_2, and so on.

26.1.1.3 Creating an Object Library

To create an object library, invoke the Librarian utility by entering the LIBRARY command with the /CREATE qualifier and the name you are assigning the library. The following example creates a library in a file named INCOME.OLB (.OLB is the default file type):

```
$ LIBRARY/CREATE INCOME
```

26.1.1.4 Managing an Object Library

To add or replace modules in a library, enter the LIBRARY command with the /REPLACE qualifier followed by the name of the library (first parameter) and the names of the files containing the (second parameter). After you put object modules in a library, you can delete the object file. The following example adds or replaces the modules from the object file named GETSTATS.OBJ to the object library named INCOME.OLB and then deletes the object file:

```
$ LIBRARY/REPLACE INCOME GETSTATS
$ DELETE GETSTATS.OBJ;*
```

You can examine the contents of an object library with the /LIST qualifier. Use the /ONLY qualifier to limit the display. The following command displays all the modules in INCOME.OLB that start with GET:

```
$ LIBRARY/LIST/ONLY=GET* INCOME
```

Use the /DELETE qualifier to delete a library module and the /EXTRACT qualifier to recreate an object file. If you delete many modules, you should also compress (/COMPRESS qualifier) and purge (PURGE command) the library. Note that the /ONLY, /DELETE, and /EXTRACT qualifiers require the names of modules---not file names---and that the names are specified as qualifier values, not parameter values.

26.1.2 Text and Macro Libraries

Any frequently used routine can be stored in libraries as source code. Then, when you need the routine, it can be called in from your source program.

Source code modules are stored in text libraries. The file extension for a text library is .TLB.

When using VAX MACRO assembly language, any source code module can be stored in a macro library. The file extension for a macro library is .MLB. Any source code module stored in a macro library must have the file extension .MAR.

You also use LIBRARIAN to create and manage text and macro libraries. Refer to Section 26.1.1.3 and Section 26.1.1.4 for a summary of LIBRARIAN commands.

26.2 Shareable Images

A **shareable image** is an image that can be linked with executable images. If you have a program unit that is invoked by more than one program, linking it as a shareable image provides the following benefits:

- Saves disk space---The executable images to which the shareable image is linked do not physically include the shareable image. Only one copy of the shareable image exists.
- Simplifies maintenance---If you use transfer vectors and the GSMATCH (on VAX systems) or symbol vectors (on Alpha and I64 systems) option, you can modify, recompile, and relink a shareable image without having to relink any executable image that is linked with it.

Shareable images can also save memory, provided that they are installed as shared images. See the *HP OpenVMS Linker Utility Manual* for more information about creating shareable images and shareable image libraries.

26.3 Symbols

Symbols are names that represent locations (addresses) in virtual memory. More precisely, a symbol's value is the address of the first, or low-order, byte of a defined area of virtual memory, while the characteristics of the defined area provide the number of bytes referred to. For example, if you define TOTAL_HOUSES as an integer, the symbol TOTAL_HOUSES is assigned the address of the low-order byte of a 4-byte area in virtual memory. Some system components (for example, the debugger) permit you to refer to areas of virtual memory by their actual addresses, but symbolic references are always recommended.

26.3.1 Defining Symbols

A symbolic name can consist of letters, digits, underscores (_), and dollar signs ($). Uppercase and lowercase letters are equivalent. By convention, dollar signs are restricted to symbols used in system components. (If you do not use the dollar sign in your symbolic names, you will never accidentally duplicate a system-defined symbol.)

26.3.2 Local and Global Symbols

Symbols are either local or global in scope. A **local symbol** can only be referenced within the program unit in which it is defined. Local symbol names must be unique among all other local symbols within the program unit but not within other program units in the program. References to local symbols are resolved at compile time.

A **global symbol** can be referenced outside the program unit in which it is defined. Global symbol names must be unique among all other global symbols within the program. References to global symbols are not resolved until link time.

References to global symbols in the executable portion of a program unit are usually invocations of subprograms. If you reference a global symbol in any other capacity (as an argument or data value---see the following paragraph), you must define the symbol as external or intrinsic in the definition portion of the program unit.

System facilities, such as the Message utility and the VAX MACRO assembler, use global symbols to define data values.

The following program segment shows how to define and reference a global symbol, RMS$_EOF (a condition code that may be returned by LIB$GET_INPUT):

```
CHARACTER*255   NEW_TEXT
INTEGER         STATUS
INTEGER*2       NT_SIZ
INTEGER         LIB$GET_INPUT
EXTERNAL        RMS$_EOF
STATUS = LIB$GET_INPUT (NEW_TEXT,
2                       'New text: ',
2                       NT_SIZ)
IF ((.NOT. STATUS) .AND.
2   (STATUS .NE. %LOC (RMS$_EOF))) THEN
  CALL LIB$SIGNAL (RETURN_STATUS BY VALUE)
END IF
```

26.3.3 Resolving Global Symbols

References to global symbols are resolved by including the module that defines the symbol in the link operation. When the linker encounters a global symbol, it uses the following search method to find the defining module:

1. Explicitly named modules and libraries---Generally used to resolve user-defined global symbols, such as subprogram names and condition codes. These modules and libraries are searched in the order in which they are specified.
2. System default libraries---Generally used to resolve system-defined global symbols, such as procedure names and condition codes.
3. User default libraries---Generally used to avoid explicitly naming libraries, thereby simplifying linking.

If the linker cannot find the symbol, the symbol is said to be unresolved and a warning results. You can run an image containing unresolved symbols. The image runs successfully as long as it does not access any unresolved symbol. For example, if your code calls a subroutine but the subroutine call is not executed, the image runs successfully.

If an image accesses an unresolved global symbol, results are unpredictable. Usually the image fails with an access violation (attempting to access a physical memory location outside those assigned to the program's virtual memory addresses).

26.3.3.1 Explicitly Named Modules and Libraries

You can resolve a global symbol reference by naming the defining object module in the link command. For example, if the program unit INCOME references the subprogram GET_STATS, you can resolve the global symbol reference when you link INCOME by including the file containing the object module for GET_STATS, as follows:

```
$ LINK INCOME, GETSTATS
```

If the modules that define the symbols are in an object library, name the library in the link operation. In the following example, the GET_STATS module resides in the object module library INCOME.OLB:

```
$ LINK INCOME,INCOME/LIBRARY
```

26.3.3.2 System Default Libraries

Link operations automatically check the system object and shareable image libraries for any references to global symbols not resolved by your explicitly named object modules and libraries. The system object and shareable image libraries include the entry points for the RTL routines and system services, condition codes, and other system-defined values. Invocations of these modules do not require any explicit action by you at link time.

26.3.3.3 User Default Libraries

If you write general-purpose procedures or define general-purpose symbols, you can place them in a user default library. (You can also make your development library a user default library.) In this way, you can link to the modules containing these procedures and symbols without explicitly naming the library in the DCL LINK command. To name a single-user library, equate the file name of the library to the logical name LNK$LIBRARY. For subsequent default libraries, use the logical names LNK$LIBRARY_1 through LNK$LIBRARY_999, as described in Section 26.1.1.

26.3.3.4 Making a Library Available for Systemwide Use

To make a library available to everyone using the system, define it at the system level. To restrict use of a library or to override a system library, define the library at the process or group level. The following command line defines the default user library at the system level:

```
$ DEFINE/SYSTEM LNK$LIBRARY $DISK1:[DEV]PROCEDURES
```

26.3.3.5 Macro Libraries

Some system symbols are not defined in the system object and shareable image libraries. In such cases, the *HP OpenVMS System Services Reference Manual* notes that the symbols are defined in the system macro library and tells you the name of the macro containing the symbols. To access these symbols, you must first assemble a macro routine with the following source code. The keyword GLOBAL must be in uppercase. The .TITLE directive is optional but recommended.

```
.TITLE macro-name
macro-name      GLOBAL
  .
  .
  .
.END
```

The following example is a macro program that includes two system macros:

LBRDEF.MAR

```
.TITLE $LBRDEF
$LBRDEF GLOBAL
$LHIDEF GLOBAL
.END
```

Assemble the routine containing the macros with the MACRO command. You can place the resultant object modules in a default library or in a library that you specify in the LINK command, or you can specify the object modules in the LINK command. The following example places the $LBRDEF and $LHIDEF modules in a library before performing a link operation:

```
$ MACRO LBRDEF
$ LIBRARY/REPLACE INCOME LBRDEF
$ DELETE LBRDEF.OBJ;*
$ LINK INCOME,INCOME/LIBRARY
```

The following LINK command uses the object file directly:

```
$ LINK INCOME,LBRDEF,INCOME/LIBRARY
```

26.3.4 Sharing Data

Typically, you use an installed common block either to facilitate interprocess communication or to allow two or more processes to access the same data simultaneously. However, you must have the CMKRNL privilege to install the common block. If you do not have the CMKRNL privilege, global sections allow you to perform the same operations.

26.3.4.1 Installed Common Blocks

To share data among processes by using a common block, you must install the common block as a shared shareable image and link each program that references the common block against that shareable image.

To install a common block as a shared image:

1. Define a common block---Write a program that declares the variables in the common block and defines the common block. This program should not contain executable code. The following HP Fortran program defines a common block:

   INC_COMMON.FOR

   ```
   INTEGER TOTAL_HOUSES
   REAL PERSONS_HOUSE (2048),
   2    ADULTS_HOUSE (2048),
   2    INCOME_HOUSE (2048)
   COMMON /INCOME_DATA/ TOTAL_HOUSES,
   2                    PERSONS_HOUSE,
   2                    ADULTS_HOUSE,
   2                    INCOME_HOUSE

   END
   ```

2. Create the shareable image---Compile the program containing the common block. Use the LINK/SHAREABLE command to create a shareable image containing the common block.

   ```
   $ FORTRAN INC_COMMON
   $ LINK/SHAREABLE INC_COMMON
   ```

   For Alpha only, you need to specify a Linker options file (shown here as SYS$INPUT to allow typed input) to specify the PSECT attributes of the COMMON block PSECT and include it in the global symbol table:

   ```
   $ LINK/SHAREABLE INC_COMMON ,SYS$INPUT/OPTION
   _ SYMBOL_VECTOR=(WORK_AREA=PSECT)
   _ PSECT_ATTR=WORK_AREA,SHR
   ```

   With HP Fortran 90 on OpenVMS Alpha systems, the default PSECT attribute for a common block is NOSHR. To use a shared installed common block, you *must* specify one of the following:
   - The SHR attribute in a cDEC$ PSECT directive in the source file
   - The SHR attribute in the PSECT_ATTR option in the Linker options file. The shareable image must be installed.

   If the !DEC$ PSECT (same as cDEC$ PSECT) directive specified the SHR attribute, the LINK command is as follows:

```
$ LINK/SHAREABLE INC_COMMON  ,SYS$INPUT/OPTION
_ SYMBOL_VECTOR=(WORK_AREA=PSECT)
```

Copy the shareable image. Once created, you should copy the shareable image into SYS$SHARE before it is installed. The file protection of the .EXE file must allow write access for the processes running programs that will access the shareable image (shown for Group access in the following COPY command):

```
$ COPY/LOG DISK$:[INCOME.DEV]INC_COMMON.EXE SYS$SHARE:*.*
_ /PROTECTION=G:RWE
```

If you do not copy the installed shareable image to SYS$SHARE, before running executable images that reference the installed shareable common image, you must define a logical name that specifies the location of that image.
On Alpha and I64 systems, when compiling the program that contains the common block declarations, consistently use the *same* / ALIGNMENT and /GRANULARITY qualifiers used to compile the common block data declaration program that has been installed as a shareable image. For more information, see Section 26.3.4.3.

3. Install the shareable image---Use the DCL command SET PROCESS/PRIVILEGE to give yourself CMKRNL privilege (required for use of the Install utility). Use the DCL command INSTALL to invoke the interactive Install utility. When the INSTALL prompt appears, enter CREATE, followed by the complete file specification of the shareable image that contains the common block (the file type defaults to .EXE) and the qualifiers /WRITEABLE and /SHARED. The Install utility installs your shareable image and reissues the INSTALL prompt. Enter EXIT to exit. Remember to remove CMKRNL privilege. (For complete documentation of the Install utility, see the *HP OpenVMS System Management Utilities Reference Manual*.)
   The following example shows how to install a shareable image:

```
$ SET PROCESS/PRIVILEGE=CMKRNL
$ INSTALL
INSTALL> CREATE DISK$USER:[INCOME.DEV]INC_COMMON -
_INSTALL> /WRITEABLE/SHARED
INSTALL> EXIT
$ SET PROCESS/PRIVILEGE=NOCMKRNL
```

Note

A disk containing an installed image cannot be dismounted. To remove an installed image, invoke the Install utility and enter DELETE followed by the complete file specification of the image. The DELETE subcommand does not delete the file from the disk; it removes the file from the list of known installed images.

Perform the following steps to write or read the data in an installed common block from within any program:

1. Include the same variable and common block definitions in the program.
2. Compile the program.
   For Alpha and I64, when compiling the program that contains the common block declarations, consistently use the *same* / ALIGNMENT and /GRANULARITY qualifiers used to compile the common block data declaration program that has been installed as a shareable image. For more information, see Section 26.3.4.3.
3. Link the program against the shareable image that contains the common block. (Linking against a shareable image requires an options file.)

```
$ LINK INCOME, DATA/OPTION
$ LINK REPORT, DATA/OPTION
```

DATA.OPT

```
INC_COMMON/SHAREABLE
```

For Alpha only, linking is as follows:

```
INC_COMMON/SHAREABLE
PSECT_ATTR=WORK_AREA, SHR
```

If a !DEC$ PSECT (cDEC$ PSECT) directive specified the SHR PSECT attribute, the linker options file INCOME.OPT would contain the following line:

```
INC_COMMON/SHAREABLE
```

The source line containing the !DEC$ PSECT directive would be as follows:

```
!DEC$ PSECT /INC_COMMON/ SHR
```

4. Execute the program.
   If the installed image is not located in SYS$SHARE, you must define a logical name that specifies the location of that image. The logical name (in this example INC_COMMON) is the name of the installed base.

In the previous series of examples, the two programs INCOME and REPORT access the same area of memory through the installed common block INCOME_DATA (defined in INC_COMMON.FOR).

Typically, programs that access shared data use common event flag clusters to synchronize read and write access to the data. Refer to Chapter 7 for more information about using event flags for program synchronization.