Starlink Project
Starlink System Note 8.1

R.F. Warren-Smith

6th March 1991

Naming Conventions for Accessing Starlink Subroutine Libraries

---

Contents

1 Introduction

This document defines a set of naming conventions to be used for accessing Starlink subroutine libraries on VAX/VMS machines. The purpose of these conventions is to:

- Provide a uniform interface for software developers who use Starlink libraries.
- Simplify the process of linking with Starlink libraries.
- Facilitate the use of shareable images, so that enhancements to libraries can be made available without the need to re-link the calling software.
- Increase the modularity of libraries, and hence allow them to be maintained and developed more easily by separate individuals.

The new arrangements are based on the use of VMS shareable images and on the definition of a new uniform set of VMS logical names and DCL symbols. A library may comply with the naming conventions described in this document by providing its facilities via logical names and symbols having the form specified in later sections. However, no restriction is placed on other names or symbols which may be provided, so an existing library may be upgraded simply by defining any new names or symbols required, along with the associated files if necessary. Names and symbols which already exist to serve the same purpose may be left in place so that software that depends on them will continue to work.

2 Use of Shareable Images

As far as possible, Starlink subroutine libraries should make their executable code available to users in the form of VMS shareable images. This approach confers a number of advantages, in particular the ability for other software to assimilate new features when the library is upgraded without the need to re-link the calling software. The linking procedures themselves are also simplified by removing the need to know about, or to specify, all the sub-libraries (or sub-sub-libraries *etc.*) which may be required.

However, shareable images place a responsibility on the developer of a subroutine library to ensure that any modifications which may be made are compatible with all previously distributed versions of the library. Any changes which would require the calling software to be re-linked will take a long time to propagate through all affected software and should always be avoided. In particular, the need to ensure upward-compatibility of shareable image *transfer vectors* must be appreciate — the VAX Linker Utility Manual should be consulted for details.

In some cases, the use of a shareable image may not be possible or appropriate. For instance, the library's executable files may have been provided commercially, or it may be desirable to allow the user to replace certain library routines with private versions – something which cannot be done in quite the same way with a shareable image[1] (of course the standard, but under-used

[1]This facility can still be provided by placing the routine to be replaced on its own in a separate shareable image. A user can then provide an alternative version of this image and access it by assigning the appropriate logical name to it at run time.

technique of passing a subroutine or function as an argument to another routine, so that the user can specify which routine should be called at a lower level, is not affected in any way by shareable libraries). In such cases, the prescription given for the use of shareable libraries in this document may not apply, but the spirit of the naming convention described here should be followed as far as possible.

3 Stand-alone and ADAM Libraries

Starlink subroutine libraries should be viewed as separate items of software which are intended to work together, as part of a complete programming environment. To function together effectively they must adhere to various conventions, and the ease with which they can be integrated into a single application will depend on the extent of the conventions which they share.

In some situations, a highly "Starlink-specific" implementation of a library may be required. By making full use of Starlink conventions, such a system can be made particularly easy to use in a Starlink environment. In other situations, however, a more general-purpose implementation might be better, especially if it may need to be used on non-Starlink machines where other supporting software will not be available. To satisfy these requirements, it is anticipated that many Starlink libraries will be made available in two "compatibility levels".

The "*stand-alone*" version of a library will be a general-purpose implementation which is intended for use on its own, separate from any pre-defined programming environment. Such a library may call the stand-alone versions of other Starlink libraries, but this dependence should generally be kept to a minimum so that the maximum freedom of use and portability is achieved. The stand-alone version of a library is provided for use by applications which do not intend to integrate closely with any particular programming environment.

Conversely, the *ADAM* version of a library will be closely bound to the Starlink ADAM programming environment, which allows a closer degree of cooperation between separate libraries (in particular through the use of the ADAM parameter system). A more integrated appearance can then be provided for the user, along with additional facilities, so that the ADAM version of a library will usually contain extra routines which are not available in the stand-alone version. This version of a library should always be used by applications which make use of ADAM facilities.

4 Library Names and Prefixes

The naming conventions described in this document are based upon the use of a unique name for each subroutine library together with a simple unique abbreviation of it. This abbreviation is commonly referred to as the *library prefix*, although both it and the full library name are used as prefixes when constructing logical names and DCL symbols associated with the library.

In the remainder of this document, the library name:

```
LIBRARY
```

and its abbreviation:

```
LIB
```

are used to denote these, with the understanding that they should be replaced with the specific library name or prefix in question.

Whenever a library name or prefix is used to construct a compound name it should be separated from the subsequent part of the name with a single underscore character '_'. This follows the established Starlink practice for naming subroutines and functions.[2]

Library names. The library name LIBRARY is the name or acronym used to identify the library as a software item in the Starlink Software Index.[3] Thus examples of library names are:

CHR	*Character Handling Routines*
EMS	*Error and Message Service*
HDS	*Hierarchical Data System*
IDI	*Image Display Interface*
PRIMDAT	*Primitive Data Processing*
SGS	*Simple Graphics System*

Library names should be no more than 8 characters long,[4] should begin with an alphabetic character and should contain only alphanumeric characters (no underscores are allowed). Existing library names should not be re-used (see the Starlink Software Index for a list of names currently in use) and potentially confusing names should be avoided. In particular, names or terms which are used for other purposes (*e.g.* command names), or which have potentially ambiguous meanings, should not be used as library names.

Library prefixes. The library prefix LIB should consist of three alphabetic characters[4] chosen, where possible, to be an abbreviation of the library name. If the library name itself consists of just three characters, then both the name and its abbreviation should be the same. The prefixes for the libraries above are:

```
CHR, EMS, HDS, IDI, PRM, SGS
```

The library prefix is intended for use not only as part of the naming conventions described in this document, but also as a prefix for the subroutines, functions and symbolic constants within the library (according to the Starlink subroutine naming conventions described in SGP/16). Thus, routines in the SGS library have names such as:

[2]The names of symbolic constants differ slightly in using a double underscore '__'.

[3]Some existing libraries (*e.g.* PAR) are not distributed as separate software items, but all new libraries should be designed with separate distribution in mind.

[4]Some existing libraries do not adhere to these conventions, but they should be observed by all new software.

`SGS_LINE`, `SGS_SHTX`, *etc.*

In some libraries, more than one routine prefix may be used to sub-divide the routines into separate facilities. In such cases the library prefix should be chosen from amongst the routine prefixes in use. For instance, the HDS library currently uses all the following routine prefixes, some of them internally:

`CMP`, `DAT`, `DAU`, `EXC`, `HDS`, `PRO`, `REC`

but only "HDS" is used as the library prefix.

To avoid conflicts, routine prefixes which are already in use are documented in an appendix to SGP/16 which is periodically updated. A note of prefixes allocated since the last revision of SGP/16 is held at RAL and a new prefix may be reserved by contacting any member of the Starlink project team at RAL.

5 Library Directory Names

For each library, a logical name of the following standard form:

```
LIBRARY_DIR
```

should be defined to identify the directory in which user-accessible files reside. All other logical name and symbol definitions associated with the library should be made in terms of this single directory name, so that if the software is moved to another location, only the assignment of this name need be changed.

The library directory name should be defined in the system logical name table by means of a suitable command in the SSC:STARTUP.COM file, such as:

```
$ DEFINE/SYSTEM LIBRARY_DIR <disk>:<directory>
```

This command will then be executed at system startup. Other software may test for the existence or non-existence of the directory name in order to determine whether the library is installed and available for use. The choice of name for the actual directory in which the software resides will be made by the Starlink Software Librarian, while the name of the disk will be determined by the System Manager who installs it.

6 Library Startup Files

It is intended that applications which call Starlink libraries may be run without further preparation once the Starlink login file has been executed using the command:[5]

[5]Except that ADAM applications will also require the $ADAMSTART command.

```
$ @SSC:LOGIN
```

However, additional logical name definitions (*e.g.* for include files, *etc.*) will usually be needed if software development is planned. Each library should therefore provide a DCL command file whose purpose is to define any logical names and/or DCL symbols needed when developing software which calls the library.

This file should reside in the library directory and a DCL global symbol of the following standard form:

```
LIB_DEV
```

should be defined in order to execute it. This symbol should be defined by means of a suitable command in the Starlink login file SSC:LOGIN.COM, such as:

```
$ LIB_DEV :== @LIBRARY_DIR:LIB_DEV
```

where LIB_DEV.COM is the command file in question (this file name is recommended for future use but existing files which serve the same purpose need not be changed).

Other software may test for the existence or non-existence of the LIB_DEV symbol in order to determine whether a library is installed in a form which allows software development.

A software developer wishing to make use of the library's facilities should first execute the command:

```
$ LIB_DEV
```

to set up the appropriate development environment. A sequence of such commands might typically be placed in a LOGIN.COM file in order to define a programming environment which is used frequently. Pre-defined programming environments such as ADAM may also make use of these commands in their own startup procedures. To facilitate this type of use, a library's startup file should ensure that if the command issued is of the form:

```
$ LIB_DEV NOLOG
```

then the presence of "NOLOG" as the first parameter will suppress the output of any informational message. Otherwise, library startup files may optionally send messages to SYS$OUTPUT when they are executed in order to identify themselves.

7 Shareable Image Names

The files which contain the executable code for a library's shareable images should reside in the library directory and should be assigned logical names of the following standard form in order to identify them:

LIB_IMAGE

and/or LIB_IMAGE_ADAM

The first of these names refers to the stand-alone version of the library and the second to the ADAM version; they are required at run-time to identify the relevant shareable images to other executable images linked against them. If either version of the library does not exist, then the corresponding logical name should be left undefined. Other software may test for the existence or non-existence of either of these logical names in order to determine the availability of a particular version of the library.

In cases where there is no difference between the stand-alone and ADAM shareable images, both logical names may refer to the same image file. In general, however, these names will refer to different files, and it should always be assumed that the images will differ. In particular, it should be assumed that their transfer vectors will be incompatible, so that they may not be interchanged once an application has been linked with a particular version.

The logical names identifying a library's shareable images should be defined in the system logical name table at system startup by means of suitable commands in the SSC:STARTUP.COM file, such as:

```
$ DEFINE/SYSTEM LIB_IMAGE      LIBRARY_DIR:LIB_IMAGE
$ DEFINE/SYSTEM LIB_IMAGE_ADAM LIBRARY_DIR:LIB_IMAGE_ADAM
```

where LIB_IMAGE.EXE and LIB_IMAGE_ADAM.EXE are the shareable image files in question (these file names are recommended for future use but existing files need not be changed).

8 Linker Options Files

Each library should provide linker options files (with a file type of .OPT) to be used by software developers when linking against the library. These files should reside in the library directory and should be assigned logical names of the following standard form in order to identify them:

`LIB_LINK`

and/or `LIB_LINK_ADAM`

The first of these allows linking with the stand-alone version of the library, and the second with the ADAM version. As usual, if either version of the library does not exist, then the corresponding logical name should be left undefined.

Normally the options file for linking with a shareable image will refer directly to that image, which should be identified by its logical name according to the conventions in §7, *e.g:*

```
LIB_IMAGE/SHARE
```

or

```
LIB_IMAGE_ADAM/SHARE
```

In cases where there is no difference between the stand-alone and ADAM executable images, both versions of the linker options file should nevertheless still be provided, and each options file should access the image through the appropriate logical name.

Users of linker options files should always assume that the stand-alone and ADAM linking processes will differ. Thus, a stand-alone application might be linked with a particular library as follows:

```
$ LIB_DEV               ! Done initially, normally in a LOGIN.COM file
$ FORTRAN MYPROG
$ LINK MYPROG,LIB_LINK/OPT
```

while an ADAM application would be linked thus:

```
$ LIB_DEV               ! Done initially, normally in a LOGIN.COM file
$ FORTRAN MYPROG
$ ALINK MYPROG,LIB_LINK_ADAM/OPT
```

Note that only those libraries explicitly referenced in the application need be considered when constructing a linker command line. Any sub-libraries called at lower levels will be included automatically through having previously been linked into those shareable images which **are** referenced.

In practice, explicit reference to startup commands or linker options files will not normally be required in order to access libraries which form part of the ADAM programming environment, since the appropriate ADAM startup and linking procedures will access all relevant libraries automatically. However, newly developed libraries, or those which are not part of ADAM, may still be accessed in ADAM applications by referring to them explicitly, as above. The number of libraries which are made available as part of the ADAM environment is likely to increase as a result of standardising the methods of library access, but the relevant ADAM documentation should be consulted for a complete list of these.

The logical names identifying the linker options files should be defined in the process logical name table by means of suitable commands in the library startup file (§6), such as:

```
$ DEFINE/NOLOG LIB_LINK      LIBRARY_DIR:LIB_LINK
$ DEFINE/NOLOG LIB_LINK_ADAM LIBRARY_DIR:LIB_LINK_ADAM
```

where LIB_LINK.OPT and LIB_LINK_ADAM.OPT are the linker options files in question (these file names are recommended for future use but existing files need not be changed).

9 Include File Names

The convention for naming and accessing include files from Fortran is already well-established within ADAM and, to a lesser degree, within non-ADAM Starlink software. Subroutine libraries may provide Fortran include files for public use and should define logical names to access them as follows:

`LIB_ERR` — *Error codes associated with the library*
`LIB_PAR` — *Other constants defined by PARAMETER statements*

Users of include files should always refer to them by their logical names, *e.g:*

```
INCLUDE 'LIB_ERR'
INCLUDE 'LIB_PAR'
```

No distinction is drawn between ADAM and stand-alone versions of include files because, in practice, they will always be the same (there being no difficulty if some of the constants defined in these files are simply not used by one or other versions of the library).

The naming of include files which do not fit into either of the above categories is arbitrary. However, logical names should always be provided for accessing them, and the library prefix LIB_ should be used at the start of each logical name.

A Summary

This appendix summarises the definitions and files which should normally form part of a Starlink subroutine library, indicating which features form a mandatory part of the library naming conventions and which are optional.

A.1 The SSC:STARTUP.COM File

Definitions such as the following will normally be required in SSC:STARTUP.COM (the Starlink system startup file[6]):

```
$ DEFINE/SYSTEM LIBRARY_DIR    <disk>:<directory>         ! Library directory name
$ DEFINE/SYSTEM LIB_IMAGE      LIBRARY_DIR:LIB_IMAGE      ! Stand-alone shareable image
$ DEFINE/SYSTEM LIB_IMAGE_ADAM LIBRARY_DIR:LIB_IMAGE_ADAM ! ADAM shareable image
```

These logical name definitions are mandatory, except that if either image file does not exist, then the corresponding logical name should be left undefined. Any existing logical name definitions for the same files may remain in place.

The image files must reside in the library directory and the file names given above are a recommendation for future use.

A.2 The SSC:LOGIN.COM File

A definition such as the following will normally be required in SSC:LOGIN.COM (the Starlink login file[6]):

```
$ LIB_DEV :== @LIBRARY_DIR:LIB_DEV
```

This global symbol definition is mandatory unless the library is installed in a form which does not permit software development, in which case it should be left undefined. Existing symbol definitions for the same purpose may remain in place. If the value "NOLOG" is supplied as the first parameter of the startup procedure, then any informational messages must be suppressed. Such messages are otherwise optional.

The library startup file must reside in the library directory and the name given above is a recommendation for future use.

A.3 The Library Startup File

Definitions such as the following will normally be required in the library startup file which is executed by the $LIB_DEV command:

[6]Note, however, that Starlink software items which are considered "optional" have their startup commands placed in a *local* version of the Starlink STARTUP.COM and LOGIN.COM files which reside in the LSSC: directory.

```
$ DEFINE/NOLOG LIB_LINK      LIBRARY_DIR:LIB_LINK      ! Stand-alone link options
$ DEFINE/NOLOG LIB_LINK_ADAM LIBRARY_DIR:LIB_LINK_ADAM ! ADAM link options
$ DEFINE/NOLOG LIB_ERR       LIBRARY_DIR:LIB_ERR       ! Error codes
$ DEFINE/NOLOG LIB_PAR       LIBRARY_DIR:LIB_PAR       ! Other constants
$ DEFINE/NOLOG LIB_?????     LIBRARY_DIR:LIB_?????     ! Any other include files
```

These logical name definitions are mandatory, except that if any of the files do not exist, then the corresponding name should be left undefined. Existing logical name definitions for the same files may remain in place.

The files must reside in the library directory and the names given above are a recommendation for future use.

A.4 Overall Summary

The following is an overall summary of the files which a library may need to provide and the logical names or symbols to be used to refer to them:

File/Directory	**Purpose**	**Name/Symbol**	**Defined in...**
<disk>:<directory>	*Library directory*	LIBRARY_DIR	SSC:STARTUP.COM
LIB_IMAGE.EXE	*Stand-alone shareable image*	LIB_IMAGE	SSC:STARTUP.COM
LIB_IMAGE_ADAM.EXE	*ADAM shareable image*	LIB_IMAGE_ADAM	SSC:STARTUP.COM
LIB_DEV.COM	*Library startup file*	LIB_DEV	SSC:LOGIN.COM
LIB_LINK.OPT	*Stand-alone link options file*	LIB_LINK	LIBRARY_DIR:LIB_DEV.COM
LIB_LINK_ADAM.OPT	*ADAM link options file*	LIB_LINK_ADAM	LIBRARY_DIR:LIB_DEV.COM
LIB_ERR.FOR	*Error codes (include file)*	LIB_ERR	LIBRARY_DIR:LIB_DEV.COM
LIB_PAR.FOR	*Other constants (include file)*	LIB_PAR	LIBRARY_DIR:LIB_DEV.COM
LIB_?????.FOR	*Any other include files*	LIB_?????	LIBRARY_DIR:LIB_DEV.COM