SUN/124.10

Starlink Project
Starlink User Note 124.10

P. T. Wallace

13th June 1995

HLP — Interactive Help System
v3.3
User Guide

Abstract

The Starlink HLP system allows an application program to retrieve named items from a hierarchically-arranged library of text. The facility resembles VAX/VMS HELP. It consists of a library plus utility programs for creating, listing and reading help libraries.

Contents

1 INTRODUCTION

The Starlink HLP system is a set of subprograms and utilities which allows an application program to retrieve named items from a hierarchically-arranged library of text.

The facility is functionally very similar to the VAX/VMS Help system. The major differences are that the Starlink HLP system (i) is implemented in a portable way and is not tied to the VAX, and (ii) allows independent creation of multiple libraries which are bound together at run-time and appear to the user as a single "tree". The system is written in a free-standing manner and does not call any other Starlink packages.

The present document will be of most interest to application programmers, though users of application packages which incorporate the HLP system may find Section 2 of some value. Section 5 is relevant only to those providing software support for the package, or others who are interested in the internals of the system.

1.1 Comparison with VMS Help

Readers already familiar with the VMS Help system may find it most convenient to begin by looking at how the Starlink HLP system differs. In the list, below, the most useful or significant features are described first, with less important differences later.

- The package is portable, and available from Starlink in source form.
- The Starlink system allows different parts of the help text tree to be stored in separate files which are joined at run-time, transparently. It is possible to refer to a file from more than one point in the hierarchy.
- If no keyword is supplied, the VMS Help system delivers the help information for the "`HELP`" topic, though this behaviour can be suppressed. In the Starlink HLP system each library has a unique top-level topic which is returned under these circumstances.
- In the VAX version, the "`...`" ellipsis feature only works for top-level topics. In the Starlink HLP system ellipsis can be used from any level.
- The keyword matching capabilities are slightly more elaborate in the Starlink version: a keyword can consist of multiple words, separated by underscores, each of which can be individually abbreviated. For example, the keyword "`ACTIVE_GALACTIC_NUCLEI`" could be abbreviated "`A_G_N`" (or "`A`" *etc*).
- In both systems, "`%`" and "`*`" wildcard characters can be used when specifying the keyword to be searched for. In the VMS system these characters must not form part of a keyword in a library. In the Starlink system it is possible (though not recommended) to use such characters, their special function being suppressed by preceding them with a "`\`" "escape" character in the search string.
- In the Starlink system there is no capability for specifying different help libraries on the command line. Any such facilities must be provided by the application.

- The VAX system treats a keyword beginning with the character "/" as a special case, signifying a command qualifier. This feature does not exist in the Starlink HLP system; all topics and subtopics begin with a level number.
- Like the VAX Help system, the Starlink HLP system ignores records beginning with "!". However, the Starlink system allows exclamation marks to appear in HLP text, whereas the VAX system would interpret them as end-of-line comments.
- The package includes subprogram interfaces and utility programs only; no equivalent of the DCL HELP command has been implemented.
- The Starlink system offers fewer library maintenance facilities. In particular, a library has to be rebuilt in its entirety from source when a single topic is changed (but note that pieces of library can be separate files in the Starlink system).
- The VMS routine LBR$OUTPUT_HELP supports five options (through its *flags* argument) that are irrelevant for the Starlink HLP_HELP routine. The only option common to both is the one which enables and disables interactive prompting.
- In the Starlink HLP system a series of keywords is separated by spaces; in the VMS Help system, "/" can also be used.
- There are minor differences in the handling of unusual and unimportant conditions, *e.g.* duplicate keywords in a library, use of punctuation characters in keywords.
- In the VAX system, top-level topics always appear in alphabetical order. In the Starlink system they appear in the order specified in the help source, as do subtopics in both systems.
- There are small differences in the way blank lines are handled. In particular, multiple blank lines preceding and following the text for a given topic are ignored in the Starlink version.
- There are unimportant differences in the formatting of output text – where blank lines occur, what keywords are converted to uppercase, *etc*.

1.2 Portability

Although the Starlink HLP facility is functionally similar to the one provided with VAX/VMS, it is not tied to the VAX, makes no use of the VMS Librarian utility and Run-Time Library and does not depend (to any significant extent) on DEC Fortran extensions. Written almost entirely in ANSI-standard Fortran, the few unavoidable machine and operating system dependencies are isolated within a small number of routines which are supplied in different forms for different platforms. At present, VAX/VMS, Sun/SunOS, Sun/Solaris, DECstation/Utrix and DEC Alpha/OSF-1 variants are available (there is also a private PC/MS-Fortran version available from the author), plus functionally inferior but machine-independent variants.

The present document describes only the VAX and Unix versions. The VAX release contains also the PC versions. More information on portability issues can be found in Section 5.2.

1.3 Package Contents

The main components of the Starlink HLP system are as follows:

- A utility program CREHLP, which reads a file of help text and writes a help library;
- A command procedure, HLIB, which front-ends the CREHLP program.
- An object library containing, among other things:
  - a subprogram, HLP_HELP, which executes an interactive help session; and
  - a subprogram, HLP_ERRMES, which translates internal error codes into message strings.

  For VAX/VMS and some Unix platforms two shareable libraries are supplied in addition to the non-shared library. One (HLP_IMAGE_ADAM.EXE on VMS) is for use with applications which run under the ADAM software environment; the other (HLP_IMAGE.EXE on VMS) is for use with stand-alone applications. There are associated link option files (HLP_LINK.OPT and HLP_LINK_ADAM.OPT for VMS, hlp_link and hlp_link_adam for Unix).

Other useful items include:

- An example application program, TSTHLP, which allows a help library to be interrogated.
- An example help library, `demo.hlp` (source) and `demo.shl` (library format).
- Example implementations of the user-supplied routines which (a) output a line of text, (b) obtain an interactive response and (c) translate a library name into a filename. The names of these examples are HLP_OUTSUB, HLP_INSUB and HLP_NAMETR respectively.
- The HLP_CREH subprogram, which carries out the translation of a file of help text into the library format. (The CREHLP program is just a front-end for HLP_CREH.)

Also supplied, but of less interest, are:

- A number of internal subprograms, some of which may have uses outside the HLP system but are not officially part of the published interface to the system, *e.g.* some of the character-string handling and I/O routines.
- Command procedures for managing development of the software.
- A utility program, LSTHLP, that lists a help library file for diagnostic purposes, showing the various index pointers *etc.* Section 5.3 includes an example of such a listing.

2 USING HELP

The example program TSTHLP, together with the sample library `demo.shl`, shows a typical help session. Type the following command:

VAX/VMS	*UNIX*
`$ RUN HLP_DIR:TSTHLP`	`% /star/bin/tsthlp`

This will produce an announcement, followed by the question:

```
Name of help library?
```

Give the reply:

VAX/VMS	*UNIX*
`HLP_DIR:DEMO.SHL`	`/star/bin/examples/hlp/demo.shl`

This will produce a "`:`" prompt; enter one of the following:

- <CR> to display the top-level help text and subtopics,
- a topic name to display the help text for the nominated topic, or
- a period to terminate the program.

At other prompts, enter:

- a subtopic name to display the text for that subtopic,
- <CR> to move back one level in the hierarchy,
- question mark "`?`" to redisplay the text for the current topic, or
- ellipsis "`...`" to display all the text below the current point in the hierarchy.

The topic and subtopic names in the library are called "keywords". When searching for a given keyword, it is unnecessary to give the full name. The rules for abbreviating keywords are as follows:

(1) The simplest form of abbreviation is to give only the first few characters. Thus "FRED" is a valid abbreviation of "FREDERIC".

(2) A keyword can be made up of one or more "words" separated by an underscore. Each such word can be abbreviated individually. Thus "A_D" is a valid abbreviation of "ANNO_DOMINI".

(3) The characters "%" and "*" match, respectively, any single non-space character or any sequence of such characters.

(4) The character "\" in the entered string is itself ignored but causes the next character to be accepted literally (but not space). This means that the various special characters can appear in keywords; however, exploiting this feature is not recommended.

(5) If a word from the entered string contains but does not end with a "*" wildcard, it must not be truncated. Thus, although "SAM" and "S*E" are both valid matches for "SAMPLE", "S*PL" is not. This feature allows abbreviations to be used which unambiguously specify the end of the string as well as the beginning.

(6) Neither the keyword in the library nor the string entered may contain leading or embedded spaces.

When the entered string is a valid abbreviation of more than one topic but is not an exact match for any topic, all the matched topics are reported. Thus "* * *" would display all the level-three topics, and "..." would display every topic in the library. When the entered string is an exact match for a topic, no further matches are reported once the exact match has been found.

Keyword comparisons are not case-sensitive: "aBc" is a valid match for "ABC" for example.

3 CREATING A HELP LIBRARY

Help files exist in two forms: (i) the *source* form, which you create using a text editor, and (ii) the *library* form, which is read by the application program when it calls the HLP_HELP subprogram. The source form is read sequentially (a two-pass operation) by the HLP_CREH routine to write the randomly-addressable indexed library file which can be interrogated by means of the HLP_HELP routine.

No facilities exist for inserting a single topic into an existing help library.

3.1 The Source File

Each source file contains one hierarchy of help text. Embedded within the file may be pointers to other help libraries, though it is perfectly possible and normal for all the help text for a given application to be stored in a single source file. Whether or not multiple files are used, the result always appears to the user as a single "tree" of help information.

Each topic within a help source file consists of lines of plain text, preceded by a special line of text containing the *level number* and the *keyword*. The level number, a single decimal digit in the first character position, can be from 0 to 9, and shows the hierarchical level of the help text

which follows. The keyword is a unique name for the topic, which will be specified by the user to retrieve the text for that topic or one of its subtopics.

The source file comprises four sorts of record; COMMENT, KEYWORD, TEXT and END records. All are ordinary formatted alphanumeric records, as produced by a text editor. Up to 132 characters are accepted, though a maximum of 80 characters is recommended.

COMMENT records have "`!`" as their first character, and are ignored.

The format of a KEYWORD record is either:

n keyword

where *n* is the level number, or:

@library n keyword

where *library* is the name of another (different) help library and follows the "`@`" with no intervening spaces. The *@library*, *n* and *keyword* fields are separated by one or more spaces. Spaces before the *n* or "`@`" are not allowed. More details on level numbers and keywords are given later.

TEXT records are just plain text, and apply to the preceding keyword. Any that precede the first keyword are ignored. Blank lines before and after each group of TEXT records are ignored. Note that the rules for recognizing other sorts of record mean that text records cannot begin with a decimal digit, "`@`" or "`END`" *etc.* It is also recommended that the character "`/`" is avoided; "`/`" is reserved for future use by Starlink and also could cause incompatibilities with VMS Help.

The optional END record consists of the three characters "`END`", in any mixture of uppercase and lowercase.

The rules for *level numbers* are as follows:

(1) Higher-level topics have smaller level numbers. Thus a topic at level *n* embraces any material at levels $n + 1$, $n + 2$, *etc.*.

(2) The order in which the keyword records appear in the source file defines the hierarchy. A level number one larger than the previous one means a subtopic; a level number less than or equal to the previous one means that the current branch has finished and we have moved to the next branch of the given level.

(3) A level number can be less than the previous one, the same, or one more. Increases greater than one are not allowed.

(4) The first keyword record in the source file defines the highest level topic for that file. Each file must have exactly one such record: no further keyword records at that level (or higher) are permitted.

(5) Though it is conventional to start each source file with a zero-level topic, it is permissible to start at any level; apart from limiting the number of levels which can appear in that library, it makes no difference.

(6) Where one help library refers to another it is not necessary for the levels to match. It is possible, for example, for the same library to appear twice in a tree at two different levels.

(7) Irrespective of the range of levels in the individual libraries, the complete help tree must not go beyond level 9.

The rules for *keywords* are as follows:

(1) The maximum length of a keyword is 64 characters. In practice, keywords longer than about 20 characters are inconvenient to use and are discouraged.

(2) Keywords remain in the format in which they were entered, with use of uppercase and lowercase preserved. Character case is, however, ignored when matching.

(3) The wildcard, escape and abbreviation features available during text retrieval mean that there is considerable freedom in choosing keywords. However, it is strongly recommended that only alphabetic characters (uppercase and lowercase), decimal digits, dollar sign, underscore and hyphen be used. Prohibited or deprecated are asterisk, percent sign, ellipsis (three or more consecutive periods), at sign, slashes (forward or back), parentheses (left or right), and quote marks (single or double).

(4) Accessing topics whose names are valid abbreviations of any that precede them can be awkward: it would be unwise to have a topic PLOT following a topic PLOTXY for example. Topics arranged in alphabetical order automatically comply with this recommendation.

Here is an example help source file:

```
!
!  Example help library
!
0 PROGRAMMING_LANGUAGES

Programming via front-panel switches, or by plugboards, is no longer
in fashion.  Even macho programmers now resort to describing what the
computer is to do in terms of text which is assembled or compiled into
machine code, or which is interpreted and executed line by line.

1 Assemblers

One line of assembly language used to turn into one machine instruction,
but these days you're never quite sure.

1 Compilers

A compiler turns high-level code which is supposed to be machine-
independent but isn't into machine code which definitely isn't .

2 Fortran

An archaic language, a fossil remnant of 1950s IBM machines.  Used
to excellent effect by hordes of programmers round the world.  Produces
more efficient code than anything except assembler.  Its imminent
demise has been announced annually since about 1963.

2 PASCAL
```

```
Used for teaching structured programming.  Comes in various toxic
vendor-specific flavours.

2 C

The most successful computer virus to date.  Great to write in.
Produces really impressive gibberish code.  Goes wrong in all sorts
of fun ways.

1 Interpreters

There's nothing quite like changing a line of code and instantly
seeing the result.

2 BASIC

Revolting old-fashioned language which lots of people understand
and use, and which runs surprisingly fast on lots of computers.

2 Forth

Forth combines fast execution with compact code and rapid program
development turnaround.  Other benefits are really sensational
gibberish code which no-one can ever understand, and a propensity
to spectacular crashes.

END
```

The file `HLP_DIR:DEMO.HLP` on VAX/VMS or `/star/bin/examples/hlp/demo.shl` on Unix platforms contains a more elaborate example.

3.2 Creating the Library File

To translate one or more files of help source into a library which can be read by an application program, execute the `hlib` command script:

VAX/VMS	*UNIX*
`$ @HLP_DIR:HLIB` *source library*	`% hlib` *source_1 source_2 ...*

where *source..* is a file of help text to be input and (on VAX/VMS platforms only) *library* is the library file to be written.

On **VAX/VMS**, the default file extensions are `.HLP` for the source file and `.SHL` for the library file. A wildcard in the name of the source file will cause each such file to be processed individually. If the first argument is omitted it defaults to "`*`". If the second argument is omitted then *library* = *source* with extension changed to `.SHL`. A wildcarded name in the second argument defaults to the name field of the first filename. Typical uses are as follows.

To translate source `EXAMPLE.HLP` into library file `EXAMPLE.SHL`:

```
$ @HLP_DIR:HLIB EXAMPLE
```

To translate all `*.HLP` source files in the current directory into `*.SHL` library files:

```
$ @HLP_DIR:HLIB}
```

To translate all `*.HLP` source files in the current directory into `*.SHL` library files in directory `[.SUB]`:

```
$ @HLP_DIR:HLIB * [.SUB]*
```

On **Unix** platforms, less flexibility is provided. The library file always has file extension `.shl`. A series of source file names may be specified, perhaps using a wildcard, and each one will be translated into an appropriately-named `.shl` file. Typical uses are as follows.

To translate source `example.hlp` into library file `example.shl`:

```
% hlib example.hlp
```

To translate all `*.hlp` source files in the current directory into `*.shl` library files:

```
% hlib *.hlp
```

Programmers wishing to integrate help library creation into their application packages may use the HLP_CREH subprogram:

```
CALL HLP_CREH (NAMETR, LUIN, SOURCE, LUOUT, LIB, LUERR, EOS, JSTAT)
```

where the arguments are as follows:

Given:

`NAMETR`	`EXTERNAL`	subroutine to translate library names into filenames
`LUIN`	`INTEGER`	I/O unit number for reading help source
`SOURCE`	`CHARACTER*(*)`	filename for help source, or spaces
`LUOUT`	`INTEGER`	I/O unit number for writing help library
`LIB`	`CHARACTER*(*)`	filename for help library, or spaces
`LUERR`	`INTEGER`	I/O unit number for error messages
`EOS`	`CHARACTER*1`	character to use as end-of-string

Returned:

`JSTAT`	`INTEGER`	status: 0 =OK, −9=fail

The conventional EOS value in all present implementations is `CHAR(0)`.

4 RETRIEVING HELP TEXT

Once a help library has been created, it can be interrogated by an application program through the HLP_HELP subprogram. The programmer supplies the following four things:

- The call to HLP_HELP.
- A subprogram which handles one line of retrieved help text, typically displaying it in some way.
- A subprogram which provides one line of interactive input.
- A subprogram which translates a help library name into a filename.

The HLP package includes a demonstration application TSTHLP.FOR, together with subprograms HLP_OUTSUB.FOR, HLP_INSUB.FOR and HLP_NAMETR.FOR.

4.1 The HLP_HELP Subprogram

The HLP_HELP routine takes a line of text containing commands or keywords and enters an interactive help session, using supplied application-specific routines to obtain further command and keyword input and to handle retrieved text. Once the help session is complete, control passes back to the calling program.

The usual way to generate the initial line of input will be to interpret the application's own help command and to present the arguments appropriately formatted. For example, the application may respond to a command "`HELP COMP C`" by calling HLP_HELP with "`COMP C`" as the initial response string. The HLP_HELP routine will then search for the subtopic "`C`" of topic "`COMP`", hand back to the application any text it finds, request further lines of input, and allow the user to explore the help tree before finally terminating and returning to the application.

The response strings accepted at each stage are given in Section 2.

The HLP_HELP routine is an integer function subprogram, which returns the number +1 to indicate successful completion and a range of negative integers to indicate various error conditions (see Section 4.6). The call is as follows:

```
ISTAT = HLP_HELP (OUTSUB, LOUT, INLINE, LU, LIB, JFLAGS, INSUB, NAMETR)
```

where the arguments are:

Given:

`OUTSUB`	`EXTERNAL`	user-supplied output subroutine (note 2, below)
`LOUT`	`INTEGER`	maximum record length accepted by OUTSUB
`INLINE`	`CHARACTER*(*)`	string specifying required help text (note 3)
`LU`	`INTEGER`	I/O unit number for reading help library file
`LIB`	`CHARACTER*(*)`	name of help library file (note 4)
`JFLAGS`	`INTEGER`	flags (note 5)
`INSUB`	`EXTERNAL`	user-supplied interactive input routine (note 6)
`NAMETR`	`EXTERNAL`	user-supplied name translation routine (note 7)

Returned:

`HLP_HELP`	`INTEGER`	status: $+1$ = OK
		-11 = illegal current level
		-12 = OUTSUB reported error
		-13 = INSUB reported error
		else = other errors (note 8)

Notes:

(1) This routine is similar, but not identical, in its argument list and action to the VAX/VMS routine LBR$OUTPUT_HELP.

(2) The user-supplied OUTSUB routine is responsible for knowing where to write the information, how to handle pagination, and so on. Details are given in Section 4.2, below.

(3) The INLINE string contains the initial series of help keywords, separated by spaces. The interactive help session will begin at the place in the help tree so specified. INLINE may contain leading as well as embedded and trailing spaces. The command itself, for example "`HELP`", is not included in this string, nor any command qualifiers which the application supports. A maximum of 9 keywords is accepted; any more are ignored. Keywords longer than 64 characters are truncated.

(4) The help library specified by LIB is in a special format, produced by the HLP_CREH routine (as described in Section 3.2). The system is not compatible with VMS .HLB files, though the source form of the library is very similar (see Section 3.1). The name is subject to environment-dependent translation at open time through the NAMETR routine (see section 4.4).

(5) At present, JFLAGS controls only one option, though more may be added in the future. JFLAGS=1 means that interactive help prompting is in effect, while JFLAGS=0 value

means that the help text for the topic specified in INLINE is looked up but no interactive dialogue ensues.

(6) The user-supplied INSUB routine is responsible for carrying out a prompted read from the interactive command device. Details are given in Section 4.3, below. (Note that if JFLAGS=0 INSUB is never called.)

(7) The user-supplied NAMETR routine is responsible for translating a help library name into a filename. Details are given in Section 4.4, below.

(8) The status values returned by this routine may be translated into text by means of the routine HLP_ERRMES. See Section 4.6.

4.2 The Output Subprogram

The user-supplied output routine, which is named OUTSUB in the above example call to HLP_HELP but can have any name, is an integer function subprogram which accepts one argument, the string to be output, and returns a status of $+1$ if OK. The output routine is responsible for knowing where to write the information, how to handle pagination, and so on.

The HLP package contains a simple example of an output routine. It is called HLP_OUTSUB and is the one used in the TSTHLP demonstration program.

Note that there is no obligation for applications simply to display the lines of text retrieved by the help system. For example, information could be encoded into the help text which allows the application to plot graphs, log usage information, change the display colour *etc*.

4.3 The Input Subprogram

The user-supplied interactive input routine, which has the name INSUB in the above example call to HLP_HELP but which can have any name, is an integer function subprogram with arguments STRING, PROMPT, L. The argument STRING receives the line input, PROMPT is the string to output prior to reading the line, and L the number of characters input. If the call is successful, a function value of $+1$ is returned.

The HLP package contains a simple example of an input routine. It is called HLP_INSUB and is the one used in the TSTHLP demonstration program.

4.4 The Name Translation Subprogram

Help libraries have two names:

- The name specified either through the `LIB` argument of the HLP_HELP call or following "`@`" in the help source file.
- The filename required by Fortran OPEN statements.

It is the job of the user-supplied NAMETR routine to translate the first form into the second.

The call is as follows:

```
CALL nametr (KMD, INSTR, OUTSTR, JSTAT)
```

where the arguments are:

Given:

KMD	CHARACTER*(*)	function number (see below)
INSTR	CHARACTER*(*)	given string

Returned:

OUTSTR	CHARACTER*(*)	returned string
JSTAT	INTEGER	status: 0 = OK
		−16 = a string had to be truncated
		−17 = other error

The function number KMD is always zero when the NAMETR routine is called by the HLP system, and means "INSTR is a help library name; translate it into a filename OUTSTR". Non-zero KMD values are available to implementors for their own purposes.

The HLP package contains a suitable NAMETR routine, called HLP_NAMETR. The translation it provides is to enclose the supplied name within a prefix and suffix. This simple transformation adequately supports applications which must run on multiple platforms. The prefix and suffix are originally spaces, which HLP_NAMETR treats as meaning "no translation required". The full set of options supported by HLP_NAMETR is as follows:

KMD	*INSTR*	*OUTSTR*	*action*
0	library name	filename	translate library name to filename
1	prefix	–	specify prefix
2	suffix	–	specify suffix
3	–	prefix	enquire prefix
4	–	suffix	enquire suffix

This is what the application must do:

```
        :
*  Routines used by HLP_HELP
      EXTERNAL ...,hlp_NAMETR
```

```
        :
        :
*  Define prefix and suffix for help name translations
      CALL HLP_NAMETR(1,PREFIX,DUMMY,J)
      IF (J.NE.0) ... error
      CALL HLP_NAMETR(2,SUFFIX,DUMMY,J)
      IF (J.NE.0) ... error
        :
        :
*  Perform help session
      CALL HLP_HELP(...,HLP_NAMETR)
        :
```

An example prefix and suffix might be "`/star/help/tpoint/`" and "`.shl`" respectively. A library name "`model`" would then become the filename "`/star/help/tpoint/model.shl`".

If this simple prefix+name+suffix model is not adequate, the programmer is free to supply his own translation routine, perhaps using environment variables or an internal table.

4.5 Terminal Handling

The example application TSTHLP works in the simplest way possible. Most real-life applications will be more sophisticated:

- The output routine HLP_OUTSUB used in TSTHLP outputs lines of text to the terminal without regard to screen management. The output routines of real applications may count lines and issue a "press return for more" message after a certain number, then wait for a response before going on; on appropriate terminals there may be screen erasures; and so on.
- Some sort of "abort" feature is useful, to provide a quick exit from the help session. On the VAX, detection of <CTRL>/Z in the application's input routine (by means of the `END=` feature of the `READ` statement), followed by simulation of multiple <CR> responses and ignoring of lines supplied to the output routine, is one way to arrange this.
- In the VMS DCL command HELP, a "`?`" response requires no <CR>, and it is possible an application may wish to provide this feature. (The demonstration program TSTHLP does not.)

4.6 Error Codes

The HLP_CREH routine and all internal subprograms return status information through an integer argument using the following codes:

$0 =$ OK

$-1 =$ Help system in wrong state

$-2 =$ Help library error on OPEN

$-3 =$ Help library error on WRITE

$-4 =$ Help library error on READ

$-5 =$ Help file error on CLOSE

$-6 =$ Attempt to WRITE outside HELP library

$-7 =$ Attempt to READ outside HELP library

$-8 =$ Help record overflows supplied string

$-9 =$ Help library creation failure

$-10 =$ Unused

$-11 =$ HLP_HELP internal error

$-12 =$ Line output failure

$-13 =$ Line input failure

$-14 =$ Invalid index entry

$-15 =$ Attempted switch to current library

$-16 =$ String too small for file name

$-17 =$ Name translation failed

HLP_HELP returns a function value of $+1$ rather than 0 to indicate success, for compatibility with the VMS HELP system. HLP_HELP may also return certain of the error values from the above list.

Most of these error conditions can only occur as a result of bugs in the HLP software. The most common one to arise during normal use is "`HELP library error on OPEN`" – for example where an incorrect library name has been supplied.

A status value `J` returned by the HLP routines may be translated into a message string `MES` by means of the HLP_ERRMES routine:

```
CALL hlp_ERRMES(J,MES)
```

The length of the `MES` string should be at least 50 characters.

4.7 Linking

ADAM tasks will be linked with the HLP library automatically. Other applications which call HLP_HELP, HLP_ERRMES, HLP_CREH, or any other of the other public routines in the HLP package may be linked as follows.

On VAX/VMS:

```
$ LINK program, HLP_LINK/OPT
```

On the Unix platforms:

```
% f77 program.o -L/star/lib `hlp\_link` -o program.out
```

(The above assumes that all Starlink directories have been added to the environment variables `PATH` and `LD_LIBRARY_PATH` as described in SUN/118.)

5 SOFTWARE SUPPORT GUIDE

This section is intended for support programmers and those interested in the internal workings of the HLP package. It is unlikely to interest application programmers or users.

5.1 Release Contents

The complete HLP system, for all platforms, is stored in a single VAX/VMS directory. The following files are included:

*.FOR	Fortran source (machine-independent)
*.BAT	various PC MS/DOS command procedures (PC only)
*.COM	various VAX/VMS DCL command procedures
*.OPT	various VAX/VMS link option files
*.MAR	transfer vector for VMS shareable images
*.IND	Fortran source (machine-independent)
*.VAX	Fortran source (VAX/VMS etc)
*.PCM	Fortran source (PC – Microsoft Fortran under MS/DOS)
*.SUN4	Fortran source (Sun SPARC etc)
*.EXE	Utility programs – VAX executables
*.OLB	VAX object module library
READ.ME	General information
HLP.NEWS	NEWS item for current release
.TEX,.TOC	LaTeX source for document
MAKEFILE.	make file for Unix platforms
MK.	mk script, which sets environment variables and runs make
HLIB.	Unix shell script for running CREHLP
HLP_LINK*.	Part of Unix link procedure.
VAX_TO_UNIX.USH	Unix shell script which accepts the release.

5.2 Portability Issues

The bulk of the system is coded in ANSI-standard Fortran. The only VAX extensions used are the SGP/16-sanctioned INCLUDE, DO WHILE, DO ... END DO and names longer than 6 characters. Full ANSI-compliance would be achieved by including the INCLUDEd text, recoding all the DOs and stripping the prefix "`HLP_`" from external names.

Specific machine dependencies are as follows. Filenames are given in their uppercase VAX/VMS forms, but are always lowercase on the Unix platforms.

- COMIC – Two versions of this INCLUDE file are supplied. `COMIC.FOR` is ANSI standard except that underscore, percent and backslash appear in character strings. `COMIC.SUN4` has a double backslash to override the "escape" mechanism used by the Fortran compilers found on most Unix platforms.
- FOPR – Two versions are supplied. `FOPR.VAX` uses the OPEN keyword `READONLY` to avoid requiring write access to the help library; it is used on the VAX and the DECstation.

The PC version uses the OPEN option `MODE='READ'` for the same purpose. The machine-independent version `FOPR.IND` uses a standard OPEN; this is the version used on the Sun.

- INSUB – Three versions are supplied. The VAX version `INSUB.VAX` uses the format descriptor `$` to suppress the newline after the prompt, and begins each line with a blank to act as a printer control code. The PC version `INSUB.PCM` is similar, but uses the format descriptor `\` to suppress the newline. `INSUB.SUN4`, which works on both the Sun and DECstation, like the VAX version uses `$`, but does not output a blank. With the machine-independent version `INSUB.IND`, a newline (unavoidably) follows the prompt.
- CREHLP – Three versions are supplied. `CREHLP.VAX` is essentially VAX-only, and works in conjunction with the `HLIB.COM` command procedure. `CREHLP.IND` works on the PC and may work on other machines; however, it uses explicit I/O unit numbers, which may need to be changed. `CREHLP.SUN4` works on the Sun and DECstations and may work on other Unix platforms; however, it uses explicit I/O unit numbers and also the filenames `fort.1` and `fort.2` (as required by the `hlib` script).
- LSTHLP – An explicit I/O unit number is used, which may need to be changed on some machines.
- OUTSUB – Two versions are supplied. `OUTSUB.IND`, which is used on VAX and PC, outputs a blank printer control code. `OUTSUB.SUN4` doesn't, and is used on the Sun and DECstation. In all versions, an explicit I/O unit number is used, which may need to be changed on some machines.
- TSTHLP – An explicit I/O unit number is used, which may need to be changed on some machines.
- UPCASE – The algorithm depends on ASCII coding, in that a–z is assumed to exist and be in the same collating order as A–Z.
- NAMETR – The routine supplied, HLP_NAMETR, either leaves the name unchanged or adds a prefix and suffix. Some other behaviour may be more approriate or convenient.

The VAX, Sun SPARCstation and DECstation versions of the HLP system are supported by the Starlink Project. Versions for other Unix platforms will be produced if and when they are needed and will also be Starlink-supported. The PC version is supported by the author.

5.3 Library Format

A help library file (as opposed to a help source file) consists of three regions:

(1) The *header* region, a single record, gives the size of the file, in characters, as a decimal number in `I12.12` format.

(2) The *index* region, terminated by an empty record.

(3) The *data* region, lines of help text interspersed with keyword records. The text region, and the whole file, are terminated by an empty record.

For a more detailed picture, it is best to look at an actual file, and one is listed below. It is the example help source file given in Section 3.1, translated into a help library with the HLIB procedure, and listed using the LSTHLP utility. The first column of the listing is the character address. All records are terminated with the chosen "EOS" character, conventionally the null `CHAR(0)`.

The file begins with the header record, in this case giving the total length of the file as 1937 characters. The records at character 13, 67, ... 348 are the index region of the file, terminated by the empty record at character 386. The records at character 387, 411, ... 1912 are the data region of the file, terminated by the empty record at character 1936.

The various fields of each index record are separated by spaces. Each record begins with three pointers in `I9.9` format. The first pointer indicates the position of the item in the data region of the file. The second pointer indicates the position of the next index entry down the branch of the help tree. The third pointer indicates the position of the next entry of the same level. In this simple library, which does not contain references to other libraries, the three pointers are followed by the level number and the keyword. Where the library refers to another, the name of the referred-to library, prefixed with an "`@`", is included prior to the level number; an example of this is given later.

```
  0   000000001937
 13   000000387 000000067 000000386 0 PROGRAMMING_LANGUAGES
 67   000000686 000000110 000000110 1 Assemblers
110   000000812 000000197 000000152 1 Compilers
152   000001473 000000310 000000386 1 Interpreters
197   000000955 000000237 000000237 2 Fortran
237   000001223 000000276 000000276 2 PASCAL
276   000001324 000000152 000000152 2 C
310   000001572 000000348 000000348 2 BASIC
348   000001709 000000386 000000386 2 Forth
386
387   0 PROGRAMMING_LANGUAGES
411   Programming via front-panel switches, or by plugboards, is no longer
480   in fashion.  Even macho programmers now resort to describing what the
550   computer is to do in terms of text which is assembled or compiled into
621   machine code, or which is interpreted and executed line by line.
686   1 Assemblers
699   One line of assembly language used to turn into one machine instruction,
772   but these days you're never quite sure.
812   1 Compilers
824   A compiler turns high-level code which is supposed to be machine-
890   independent but isn't into machine code which definitely isn't .
955   2 Fortran
965   An archaic language, a fossil remnant of 1950s IBM machines.  Used
1032  to excellent effect by hordes of programmers round the world.  Produces
1104  more efficient code than anything except assembler.  Its imminent
1170  demise has been announced annually since about 1963.
1223  2 PASCAL
1232  Used for teaching structured programming.  Comes in various toxic
1298  vendor-specific flavours.
1324  2 C
1328  The most successful computer virus to date.  Great to write in.
1392  Produces really impressive gibberish code.  Goes wrong in all sorts
```

```
1460   of fun ways.
1473   1 Interpreters
1488   There's nothing quite like changing a line of code and instantly
1553   seeing the result.
1572   2 BASIC
1580   Revolting old-fashioned language which lots of people understand
1645   and use, and which runs surprisingly fast on lots of computers.
1709   2 Forth
1717   Forth combines fast execution with compact code and rapid program
1783   development turnaround.  Other benefits are really sensational
1846   gibberish code which no-one can ever understand, and a propensity
1912   to spectacular crashes.
1936
```

Where a help library includes references to other libraries, index entries like the following one appear:

```
  91   000002791 000000149 000000149 @cmnds 1 Commands
```

The corresponding keyword entry in the data region of the file is normal in appearance except that there are no text records.

5.4 The INCLUDE files

The files `helpic` and `comic` (on the Unix platforms – `HELPIC.FOR` and `COMIC.FOR` on the VAX, `HELPIC` and `COMIC` on the PC) contain definitions and COMMON declarations. They are included through "`INCLUDE 'helpic'`" and "`INCLUDE 'comic'`" statements, the interpretation of which is machine-dependent. The `comic` INCLUDE is used only by the "`HLP_COMSTR`" routine and exists in different forms for different platforms.

5.5 Subprograms

This section is a classified summary of the subprograms which make up the HLP package. For more complete information, including argument data types and character string lengths, it will be necessary to refer to the source code itself.

5.5.1 Supported interfaces

The following three routines, plus the templates described in the next section, constitute the supported interface to the HLP package. Use of any other HLP_ routines, or of the COMMON blocks used by the HLP system, is at the programmer's own risk.

```
SUBROUTINE HLP_ERRMES(J,MES)
```

HLP_ERRMES translates a HLP system error code `J` into a message string `MES`. See Section 4.6.

```
SUBROUTINE HLP_CREH(NAMETR,LUIN,SOURCE,LUOUT,LIB,LUERR,EOS,JSTAT)
```

HLP_CREH translates a file of help source SOURCE, on I/O unit LUIN, into a help library LIB, subject to name translation by the routine NAMETR, on I/O unit LUOUT. Any error messages go to I/O unit LUERR. EOS is the end-of-string character, conventionally CHAR(0). JSTAT is the status. See Section 3.2.

```
INTEGER FUNCTION HLP_HELP(OUTSUB,LOUT,INLINE,LU,LIB,JFLAGS,INSUB)
```

HLP_HELP is the main interface between an application and the HLP system, and carries out a complete interactive help session. Full details are given in Section 4.1.

5.5.2 Templates

```
SUBROUTINE HLP_NAMETR(KMD,INSTR,OUTSTR,JSTAT)
```

This implementation of the user-supplied name translation routine is suitable for use in a multiple-platform environment. It forms each filename by sandwiching the library name within a prefix and a suffix. HLP_NAMETR, or a user-supplied equivalent, is called by the HLP system prior to opening any help library, always with KMD=0. The library name to be translated is supplied in INSTR, and the filename is returned in OUTSTR ready for use in a Fortran OPEN statement. See Section 4.4 for further details of the HLP_NAMETR routine supplied, which uses other KMD values to set and enquire the prefix and suffix strings.

```
INTEGER FUNCTION HLP_INSUB(STRING,PROMPT,L)
```

This is a simple example of the user-supplied input routine which is to be called by the HLP_HELP routine each time an interactive response is required. STRING is the response received. PROMPT is the prompt string, output before STRING is solicited. L is the length of STRING, excluding any trailing spaces. The function returns the value +1 to indicate success. Note that the order of the arguments does not comply with Starlink standards (see SGP/16), a consequence of maintaining a family-resemblance with VMS Help. See Section 4.3. As well as being a template, HLP_INSUB is used in the LSTHLP and TSTHLP programs.

```
INTEGER FUNCTION HLP_OUTSUB(STRING)
```

This is a simple example of the user-supplied output routine which is to be called by the HLP_HELP routine each time a line of help text is to be output. STRING is the string to be output. The function returns the value +1 to indicate success. See Section 4.2. As well as being a template, HLP_OUTSUB is called in the LSTHLP and TSTHLP programs.

5.5.3 String handling

The following routines overlap in function with ones found in other Starlink packages, but are included in the HLP package for independence.

```
LOGICAL FUNCTION HLP_COMSTR(FULSTR,STR)
```

HLP_COMSTR compares two keywords for agreement, under the HLP package's rules for abbreviation (see Section 2). FULSTR is the full keyword read from the HLP library; STR is the abbreviated string, including wildcards *etc*, supplied by the user. A .TRUE. response is returned if the two strings match.

```
SUBROUTINE HLP_DEC(STRING,IPTR,NUM)
```

HLP_DEC decodes a decimal integer NUM from a string STRING, starting at position IPTR. The pointer IPTR is incremented ready for the next call.

```
INTEGER FUNCTION HLP_LENGTH(STRING)
```

The HLP_LENGTH function returns the length of a string STRING excluding any trailing spaces. An all-blank string is length 1. The routine is optimized for speed in the case where the string is mostly trailing spaces.

```
SUBROUTINE HLP_SPLIT(STRING,ISTART,IFROM,ITO)
```

HLP_SPLIT splits up a sentence into words. STRING consists of words separated by spaces. Starting at character position ISTART, this routine locates the beginning and end of the next word IFROM and ITO.

```
SUBROUTINE HLP_UPCASE(STRING)
```

HLP_UPCASE converts a string STRING to uppercase, in place.

5.5.4 Variable-length random-access file package

help libraries are standard Fortran direct-access files, which have a fixed record length, on which the system implements a variable-length record scheme. The following routines manage these files. They communicate in part through labelled COMMON.

```
SUBROUTINE HLP_HCLOSE(JSTAT)
```

HLP_HCLOSE closes the current help library. JSTAT is the status return.

```
SUBROUTINE HLP_HDREAD(IADR,STRING,NC,JSTAT)
```

HLP_HDREAD performs a direct-access read from the current help library. IADR is the character address from which the read is to begin. STRING is the record that is found there, and NC its length. JSTAT is the status.

```
SUBROUTINE HLP_HDWRIT(STRING,IADR,JSTAT)
```

HLP_HDWRIT performs a direct-access write to the current help library. IADR is the character address from which the write is to begin. STRING is the record to be written. JSTAT is the status.

```
SUBROUTINE HLP_HINIT(LU,FNAME,EOS)
```

HLP_HINIT initializes the HLP system labelled COMMON blocks. LU is the I/O unit number to be used for accessing the help library file. FNAME is the initial help library name. EOS is the special character that will signify end-of-string, conventionally CHAR(0).

```
SUBROUTINE HLP_HOPENR(NAMETR,JSTAT)
```

HLP_HOPENR opens or re-opens the current help library for reading, with the assistance of the library name to filename translation routine NAMETR. JSTAT is the status.

```
SUBROUTINE HLP_HOPENW(NAMETR,NCHARS,JSTAT)
```

HLP_HOPENW opens the current help library for writing, with the assistance of the library name to filename translation routine NAMETR. NCHARS is the length of the file, which must be known before the call. JSTAT is the status.

5.5.5 Indexed-sequential package

The HLP system uses its variable-length direct-access file system, described above, to implement the indexed-sequential structure described in Section 5.3. The file has an index region and a data region. The index region contains three sets of pointers, to the data, to the next index entry down the branch, and the next index entry at the same level in the current branch.

```
SUBROUTINE HLP_HREADD(STRING,NC,JSTAT)
```

HLP_HREADD performs a sequential-access read from the data region of the help library. The returned string STRING has length NC. JSTAT is the status.

```
SUBROUTINE HLP_HREADX(NAMETR,NAVIG,STRING,NC,JSTAT)
```

HLP_HREADX reads the help library index, leaving the sequential-access addresses pointing to the indexed record. If it is necessary to translate a library name into a filename, the routine NAMETR will be called. NAVIG controls which track through the library is required – down the branch or along a level. STRING is the record input and NC its length. JSTAT is the status.

```
SUBROUTINE HLP_HSEEKX(FNAME,IADRX,LOGLEV)
```

HLP_HSEEKX positions the index of the help library FNAME for a sequential access (using the HLP_HREADX routine) starting at character address IADRX. The caller also supplies LOGLEV, the *logical level*, which specifies the hierarchical level, on this occasion, of the help subtree contained in the library.

```
SUBROUTINE HLP_HTELLX(FNAME,IADRX,LOGLEV)
```

HLP_HTELLX inquires the help library's current name, index address and logical level.

5.5.6 Internal

The following are highly specialized internal routines, with no significant potential use outside the HLP system (and which are in any case inaccessible to users of the shareable libraries on VMS). For further details, see the source code.

```
SUBROUTINE HLP_HCHKL(RECORD,LEVEL,NAME)

SUBROUTINE HLP_HLEAP(NAMETR,STRING,FNAME,IADR,LOGL,JSTAT)

SUBROUTINE HLP_LINOUT(OUTSUB,LOUT,INDENT,BUFFER,JSTAT)

SUBROUTINE HLP_REPSUB(NAMETR,OUTSUB,LOUT,LEVCUR,NAME,OUTBUF,HLPBUF,ISTAT)

SUBROUTINE HLP_FOPR (NAMETR,LU,FILE,LREC,JSTAT)
```

5.6 Rebuilding the System

On the VAX, the DCL command procedure CREATE.COM in the release directory rebuilds the object library, compiles and links the executable programs, generates the demonstration help library from source, and produces the Unix release. The latter consists of an archive file, a `makefile`, and a script `mk` which sets up environment variables and then runs make. For details about the targets supported, execute `mk help`.

5.7 VAX/VMS Logical Names

On VAX/VMS, in order to run applications using HLP the following logical names must be assigned:

logical name	*points to*	*typical assignment*
HLP_DIR	HLP software	`STARDISK:[STARLINK.LIB.HELP]`
HLP_IMAGE	shareable library (non-ADAM)	`HLP_DIR:HLP_IMAGE.EXE`
HLP_IMAGE_ADAM	shareable library (ADAM)	`HLP_DIR:HLP_IMAGE_ADAM.EXE`

In order to link a non-ADAM application with the shared library, the following additional assignment is required:

HLP_LINK	link options file	`HLP_DIR:HLP_LINK.OPT`

6 ACKNOWLEDGMENTS

Early versions of the HLP system drew on the code developed by Roger Noble (Jodrell Bank) which runs on the Alliant machine as part of the OLAF system. Additional guidance came from the MRAO Cambridge HELP system, which runs on Norsk Data computers; listings of this software were kindly supplied by David Titterington. Development of the HLP_COMSTR string-comparison routine was done in collaboration with Paul Rees (Starlink Project). Brian McIlwrath and David Terrett (Starlink Project) wrote the HLIB command procedure and the `hlib` shell script respectively. The port to Unix was carried out with the help of David Terrett, Paul Rees and Chris Clayton. Further work, concerning use in the ADAM environment, the provision of shared libraries, and the use of the Starlink template makefile was carried out by Alan Chipperfield and Peter Allan (Starlink Project).

The VAX/VMS Help system was an invaluable guide in the development of the specification for the Starlink HLP package.