SUN/144.15

Starlink Project
Starlink User Note 144.15

A J Chipperfield

17 August 2001

ADAM
Unix Version 4.0

Abstract

This document describes the use of the Starlink Software Environment, ADAM, on Unix. It is primarily of use to programmers but the early sections contain information useful to any user.

It is assumed that the reader is familiar with the concepts of ADAM programming and that the Starlink software is installed in the standard way.

Contents

1 Introduction

The Starlink Software Collection provides an infrastructure of facilities likely to be required by any astronomical application package. The Starlink Software Environment (ADAM) is a particular way of combining the elements of the collection to provide an integrated system with a common look and feel across various packages. Originally developed for VMS at ROE, the system is now supported by Starlink and has been ported to Unix.

Normal data analysis programs for the Starlink Environment are known as A-tasks and it is possible to combine many A-tasks into a single monolithic executable file (an A-task monolith) for efficiency – the tasks may still be invoked separately. The Environment also supports programs for instrumentation control. These need to behave differently from A-tasks and are known as I-tasks,

Programs written for the Starlink Software Environment may be run directly from a Unix shell or from a variety of other user-interfaces including the original ADAM user-interface, ICL (see SG/5), and IRAF cl (see SUN/217).

The first five sections of this document are relevant for any user but the remaining sections will only be of interest to people writing software to run in the environment. For an introduction to these topics and details of how to write ADAM programs, see SG/4 for A-tasks and SUN/134 for I-tasks.

Fine detail about methods of controlling the behaviour of ADAM programs and ICL using environment variables is also given in the appendices.

2 Running from the shell

Most Starlink application programs are part of a 'package'. The documentation for the package will tell you how to start it up and run its applications.

Stand-alone ADAM programs are run from the shell in the same way as any Unix program by typing:

```
% program_name
```

where `% program_name` may be a full pathname.

The Interface File, which defines the program's parameters, is usually in the same directory as the executable but an alternative search path for Interface Files may be specified in environment variable `ADAM_IFL`. If the variable is undefined, or the search is unsuccessful, the directory in which the executable was found is assumed. A file with the same name as the executable and with extension `.ifc` or, failing that, `.ifl` is sought.

If the task is built into a monolith, a link with the name of the task must be made to the monolithic executable and an individual Interface File for the task provided. The link can then be executed as if it were a simple program. (This is all usually set up by package startup scripts – see SSN/46.)

When running tasks directly from a shell, the normal Unix rules for the use of meta-characters on the command line will apply – these sometimes conflict with the ADAM parameter definition syntax. Characters to be particularly wary of are '`"`', '`\`' and '`$`'.

3 ICL for Unix

ICL for Unix behaves very like the VMS version described in SG/5, however there are differences. The ICL HELP command will give more up-to-date and Unix oriented information than SG/5.

ICL is started from a Unix shell by a command of the form:

`% icl` [*ICL_options*] [*command_filenames...*]

Where:

ICL_options (optional) have not yet been fully developed and should not be used without advice. ICL options start with '`-`' and must appear before any command-file names.

command_filenames... (optional) are the names of any files containing ICL commands which are to be obeyed by ICL before the ICL prompt appears. They are loaded *after* any ICL login files (see Appendix E). A file extension of `.icl` is assumed if no extension is specified. All filenames must appear after any ICL options.

Examples:

```
% icl
```

This will be the normal invocation. Only defined ICL login files will be loaded before the ICL prompt is output.

```
% icl -io test_io comfile1 comfile2
```

which would set the value of the ICL option `-io` to `test_io` and the ICL command files `comfile1.icl` and `comfile2.icl` would be automatically loaded into ICL (after any defined ICL login files) before the ICL prompt was output.

4 Input-line Editing

When you reply to prompts from either an ADAM program or ICL, previous input can be recalled and edited – automatic filename completion is also possible in much the same way as it would be on the shell command line. At prompts for ADAM program parameters, you can also paste the suggested value into the input buffer and edit it. The maximum length of an input line is 256 characters – the terminal will beep if you attempt to input more.

Line recall and editing for ADAM uses virtually the same keys as `tcsh` with the `emacs` key bindings.

4.1 Input-line Recall

This is achieved using the up and down arrow keys – the recalled line can then be edited in the normal way. Up to a maximum of 100 input lines will be remembered but in the case of programs run from the shell, only responses to prompts in the current invocation of the program can be recalled.

4.2 Suggested Value Recall

If there are no characters on the input line, the TAB key (or ESC,ESC) will cause the suggested value (if any) to be inserted as the input line. It can then be edited in the normal way. (There will be no suggested value at the ICL> prompt.)

See Filename Completion (Section 4.3) for the effect of these characters if there is already some input on the line.

4.3 Filename completion

If there are currently characters on the input line, the TAB key (or ESC,ESC) will cause filename completion to be attempted on the word preceding the cursor. If no match is found, "No match." will be printed and the terminal will beep; if more than one match is found, "Multiple matches." will be printed and the terminal will beep – the input line will be set to the longest common prefix. A list of all possible matches can be displayed by typing ESC,CNTL/D. (At the end of line, just CNTL/D is sufficient – elsewhere CNTL/D will delete the character at the cursor.)

To overcome the problem of Starlink NDF and HDS filenames usually being required without the .sdf extension, .sdf will be omitted from the end of any completed filename. This behaviour may be altered by setting environment variable ADAM_EXTN to a comma-separated list of extensions (in fact any strings) which are to be omitted from the end of completed filenames. If no truncation of filenames is required, ADAM_EXTN should be set to a null string.

When a single match is found, the filename is truncated if required and copied to the input line followed by a single space.

For example, suppose the default for filename truncation (.sdf) is in use and the current directory contains two files file.dat and file.sdf. The dialogue might go as follows (<> indicates typing by the user):

```
Give NDF name > <x><TAB>
No match.[beep]
Give NDF name > <f><TAB>
Multiple matches.[beep]
Give NDF name > file.<CNTL/D>
file.dat    file.sdf
Give NDF name > file.<s><TAB>
Give NDF name > file <Return>
Give auxiliary data file name > <file.d><TAB>
Give auxiliary data file name > file.dat <Return>
```

(The last four lines would appear as two lines on the terminal, the second and fourth overwriting the first and third respectively.)

4.4 Editing the Input Line

The left and right arrows and the delete key may be used as expected to edit the current input line. Input is always in 'insert' rather than 'overwrite' mode.

Key sequences for more complex line editing are given in Appendix F.

4.5 Other Special Keys

`CNTL/Z` This will suspend the program in the normal way. It may be re-started with the `fg` command.

`CNTL/C` This will abort a task running from the shell. If running ICL, the effect depends on what is happening at the time. Generally, ICL itself will keep running but the current activity will be aborted (see SG/5 for information on ICL exception handling).

5 The ADAM_USER Directory

When any ADAM application, or ICL, is run, a directory known as the **ADAM_USER** directory is used to hold various automatically-created files. These are: program parameter files, the global parameter file (GLOBAL.sdf) and task identifying files (used by the ADAM inter-task message system). The directory is also used by the MAG library as the default to contain information about the user's tape devices.

By default, subdirectory `adam` in the user's HOME directory is used. If you want to use some other directory, set its name in environment variable ADAM_USER.

Whichever directory is used, it will be created if necessary (and possible). The directory and/or its contents can be deleted when not in use, but this may remove memory of parameter values last used.

6 Compiling and Linking

Originally ADAM programs were always written in Fortran, the top-level module being written as a subroutine with a single `INTEGER` argument, the ADAM status. The link scripts now also accept C functions. If the top-level function is a C function, it must have a single argument of type `int *`. If it is not written as a Fortran-callable routine, the source file must be presented to the link script.

Note that the complete set of C interfaces for the Starlink libraries is not yet available so it may not be possible to write your program entirely in C without the aid of something like the CNF package (see SUN/209).

To compile and link ADAM programs, it is necessary to add `/star/bin` to your PATH environment variable. This is done if you have 'sourced' `/star/etc/login` to set up for Starlink software generally.

6.1 Include Files

All public include files, such as *pkg* _par and *pkg* _err files, will be found in /star/include.

The filenames of the Fortran include files are in lower case but they should be specified in the program in the standard Starlink way. That is, for example:

```
INCLUDE 'SAE_PAR'
INCLUDE 'PAR_ERR'
```

Links are then set up to the public include files in /star/include using the appropriate *pkg*_dev script. For example:

```
% par_dev
```

creates links PAR_PAR and PAR_ERR to /star/include/par_par and /star/include/par_err respectively. (SAE_PAR is defined by star_dev.)

For C programs, includes take the normal form:

```
#include "par_err.h"
```

(-I/star/include is included in the link scripts).

6.2 ADAM Link Scripts

Two shell scripts are provided to link ADAM programs. alink is used to link A-tasks and A-task monoliths, and ilink to link I-tasks. The executable images produced may be run either from a suitable user-interface such as ICL, or directly from the shell.

The following description uses alink in examples; ilink is used in the same way.

To link an ADAM program, ensure that /star/bin is added to your environment variable PATH, then type:

% alink [-xdbx] *file* [*arguments*]

Where:

-xdbx optional (but must be the first argument if used), overcomes some problems with debugging ADAM tasks, notably with xdbx and ups on SunOS, by supplying dummy source files for required ADAM system routines. It also has the effect of inserting a -g option into the argument list. For more details, see Appendix B.

prog_module specifies a file containing the task's main subroutine. The filename should be of the form *path*/*name*.**f**, *path*/*name*.**c**, *path*/*name*.**o** or *path*/*name*. The *path*/ component is optional but *name* must be the name of the program's main subroutine and will be the name of the executable file produced in the current working directory. If the file extension is .f or .c, the file will be compiled appropriately; if none of the permitted extensions is given, .o is appended.

Note that after a .f file has been compiled, on some platforms the .o file will be retained in the current working directory so that subsequent alinks with an unchanged .f file may specify the name with no extension (or .o). The object files for any compiled C files will always be retained.

arguments optional, is any additional arguments (library specifications, compiler options *etc.*) legal for the Fortran compiler, or any C files to be compiled separately and linked in. The list of Starlink libraries automatically included in the link is given in Appendix C; the method of including other Starlink libraries in ADAM programs will be specified in the relevant Starlink User Note.

In most cases it will be:

% `alink` *prog_module* ‘*pkg*`_link_adam`‘

where ***`pkg`*** is the specific software item name *e.g.* `ndf`.

Appendix B gives details of the command used within `alink/ilink` to compile and/or link the tasks. This may assist users who wish to alter the standard behaviour.

By default, programs are statically linked with the Starlink libraries. On platforms where shared libraries are installed, programs may be linked with them by setting the environment variable ALINK_FLAGS1 appropriately. On currently supported systems it should be set to `-L/star/share`. (See Appendix B for more details.)

6.3 Interface Files

Interface Files may be compiled by the `compifl` program which is available in `/star/bin`. Compiled Interface Files (`.ifcs`) must be produced on the platform on which they are to be used.

Generally, the only things which will need changing when porting Interface Files between operating systems are file and device names. Case is significant on Unix and obviously the format of filenames is different from VMS, for example.

6.4 Monoliths

The top-level routine for Unix A-task monoliths should be of the form:

```
      SUBROUTINE TEST( STATUS )
      INCLUDE 'SAE_PAR'
      INCLUDE 'PAR_PAR'

      INTEGER STATUS

      CHARACTER*(PAR__SZNAM) NAME

      IF (STATUS.NE.SAI__OK) RETURN

*     Get the action name
      CALL TASK_GET_NAME( NAME, STATUS )

*     Call the appropriate action routine
      IF (NAME.EQ.'TEST1') THEN
        CALL TEST1(STATUS)
      ELSE IF (NAME.EQ.'TEST2') THEN
        CALL TEST2(STATUS)
```

```
ELSE IF (NAME.EQ.'TEST3') THEN
  CALL TEST3(STATUS)
END IF
END
```

To run such a monolith from a Unix shell, link the required action name to the monolith, then execute the linkname (possibly via an alias). For example:

```
% ln -s $KAPPA_DIR/kappa add
% add
```

Separate Interface Files are required for each action run from the shell – a monolithic Interface File is required for monoliths run from ICL.

7 Help Files

The Starlink portable help system (HLP) is used to provide help if ? or ?? is typed in response to a parameter prompt. Instructions for creating help libraries and navigating through them may be found in SUN/124.

Interface File entries specifying help libraries may be given as standard pathnames *e.g.*:

```
helplib /star/help/kappa/kappa.shl
```

or

```
help %/star/help/kappa/kappa ADD Parameters IN1
```

Note that the file extension `.shl` is optional.

The system will also accept environment variables and ~ in the help library name, *e.g.*:

`$KAPPA_DIR/kappahelp.shl` or `$KAPPA_HELP` or `~/help/myprog`

For historical reasons, so that the same Interface File entry will work with both Unix and VMS, the VMS form:

`KAPPA_DIR:kappahelp.shl` or `KAPPA_HELP:`

will be accepted.

If the VMS form is used, the environment variable name will be forced to upper case, and the filename to lower case for interpretation. Unix-style specifications will be interpreted as given.

8 Miscellaneous Points

- For Unix ADAM applications linked since ADAM V3.1, the ADAM_USER directory and global parameter file, `GLOBAL.sdf`, are created automatically.
- Compiled Interface Files (`.ifcs`) will only work for the platform type on which they were created.
- Monolith top-level routines must be written in the style described in Section 6.4 for use on Unix. Monoliths are linked using `alink`.
- ADAM tasks will usually default to interpreting environment variables and '~' in filenames used with HDS. This is not necessarily true of names used with other ADAM facilities such as FIO and may be overridden by user or programmer action for HDS filenames.

9 References

Note: Only the first author is listed here.

SG/4	M.D.Lawden	ADAM – The Starlink Software Environment.
SG/5	J.A.Bailey	ICL – The Interactive Command Language for ADAM.
SUN/101	Jo Murray	Introduction to ADAM Programming.
SUN/124	P.T.Wallace	HELP – Interactive Help System.
SUN/134	B.D.Kelly	ADAM – Guide to Writing Instrumentation Tasks.
SUN/171	P.M.Allan	MAG – Access to Magnetic Tapes.
SUN/209	P.M.Allan	CNF and F77 – Mixed Language Programming.
SUN/217	A.J.Chipperfield	Running Starlink Programs from IRAF CL.
SSN/4	P.C.T.Rees	EMS – Error Message Service.
SSN/64	A.J.Chipperfield	ADAM – Organization of Application Packages.

Key:

SG	Starlink Guide.
SSN	Starlink System Note.
SUN	Starlink User Note.

10 Document Changes

10.1 SUN/144.13

The sections on the use (Section 6.2) and details (Appendix B) of the ADAM link scripts, `alink` and `ilink`, have been modified to reflect some changes designed to allow easy switching between static and dynamic linking with the Starlink libraries.

10.2 SUN/144.14

The document was re-formatted to include the standard copyright statement - there are no changes in substance.

10.3 SUN/144.15

The ADAM and MESSYS libraries are removed from the list of available libraries.

Environment variable ADAM_NOPROMPT is added to the list of ADAM Environment Variables.

A Example Session

The following session shows the process of compiling, linking and running an example program, derived from SUN/101, on the Sun.

```
% source /star/etc/login
%
% ls
repdim2.f       repdim2.ifl     example.sdf
%
% cat repdim2.f
*   Program to report the dimensions of an NDF.
*   The CHR package is used to produce a nice output message.
*   See SUN/101, section 11.

      SUBROUTINE REPDIM2 (STATUS)
      IMPLICIT NONE
      INCLUDE 'SAE_PAR'
      INTEGER DIM(10), I, NCHAR, NDF1, NDIM, STATUS
      CHARACTER*100 STRING

*   Check inherited global status.
      IF (STATUS.NE.SAI__OK) RETURN

*   Begin an NDF context.
      CALL NDF_BEGIN

*   Get the name of the input NDF file and associate an NDF
*   identifier with it.
      CALL NDF_ASSOC ('INPUT', 'READ', NDF1, STATUS)

*   Enquire the dimension sizes of the NDF.
      CALL NDF_DIM (NDF1, 10, DIM, NDIM, STATUS)

*   Set the token 'NDIM' with the value NDIM.
      CALL MSG_SETI ('NDIM', NDIM)

*   Report the message.
      CALL MSG_OUT (' ', 'No. of dimensions is ^NDIM', STATUS)

*   Report the dimensions.
      NCHAR = 0
      CALL CHR_PUTC ('Array dimensions are ', STRING, NCHAR)
      DO  I = 1, NDIM
*      Add a 'x' between the dimensions if there are more than one.
         IF (I.GT.1) CALL CHR_PUTC (' x ', STRING, NCHAR)
*      Add the next dimension to the string.
         CALL CHR_PUTI (DIM(I), STRING, NCHAR)
      ENDDO
      CALL MSG_OUT ('  ', STRING(1:NCHAR), STATUS)

*   End the NDF context.
```

```
      CALL NDF_END (STATUS)
      END
%
% cat repdim2.ifl
interface REPDIM2
   parameter      INPUT
      position    1
      prompt      'Input NDF structure'
      default     example
      association '->global.ndf'
   endparameter
endinterface
%
% star_dev
% alink repdim2.f `ndf_link_adam`
f77 -g -c repdim2.f
repdim2.f:
        repdim2:
dtask_applic.f:
        dtask_applic:
%
% repdim2
INPUT - Input NDF structure /@example/ >
No. of dimensions is 1
Array dimensions are 856
%
% ls ~/adam
repdim2.sdf     GLOBAL.sdf
%
% compifl repdim2
!! COMPIFL: Successful completion
%
% ls
example.sdf   repdim2.f     repdim2.ifl
repdim2       repdim2.ifc   repdim2.o
%
```

Note that whether or not the file repdim2.o is retained depends upon the compiler used.

B Link Script Details

The link scripts firstly have to create a subroutine, DTASK_APPLIC, which is called by the ADAM fixed part and in turn calls the user's top-level routine. The difference between `alink` and `ilink` is just that the template DTASK_APPLIC for `alink` contains a call to close down the parameter system after each invocation of the task unless the environment variable ADAM_TASK_TYPE is set to 'I'.

If the user's main routine is written in C, a temporary routine, DTASK_WRAP is created as a Fortran-callable wrapper for the user's routine. DTASK_APPLIC will call DTASK_WRAP, which in turn calls the user's routine.

During installation (as part of the DTASK library in the PCS package), the actual compile/link command within `alink/ilink` is edited depending upon the platform and setting of various environment variables. The template command is:

```
F77 $FFLAGS -o $EXENAME \
$XDBX \
/star/lib/dtask_main.o \
dtask_applic.f \
$ALINK_FLAGS1 \
$ARGS \
-L/star/lib \
-lhdspar_adam \
-lpar_adam \
‘dtask_link_adam‘ \
$ALINK_FLAGS2 \
Additional system libraries
```

Notes:

- F77 is replaced by whatever the FC environment variable is when `alink/ilink` is installed.
- Any C source files specified will be compiled separately and linked in the above Fortran command. The C compiler command used will be of the form:

  ```
  % CC -c $CFLAGS $CARGS -I/star/include
  ```

  where:

  `CC` is replaced by the CC environment variable when `alink/ilink` script is installed.

  `$CFLAGS` is the value of the CFLAGS environment variable when `alink/ilink` is invoked.

  `$CARGS` is any C files or `-I` or `-D` arguments on the `alink/ilink` command line, plus, if the main routine is C, `dtask_wrap.c`

- `$EXENAME` is the basename of the *prog_module* argument of `alink/ilink` with any `.f`, `.c` or `.o` suffix removed.
- Environment variable FFLAGS may be used to specify any options which must be included at the start of the command line.

- `$XDBX` is set to `-g` if the `-xdbx` argument is given.
- `$ALINK_FLAGS1` may be set to determine the type of linking required. For instance:

```
% setenv ALINK_FLAGS1 -L/star/share
```

  would cause the linker to find the Starlink shared libraries (on platforms where they are installed), thus producing a dynamically linked executable.

- `$ARGS` is the *prog_module* argument (with `.o` appended if the original extension was not `.f`, `.c` or `.o`) or `$EXENAME.o dtask_wrap.o` if the main routine was in C, followed by the remaining arguments unchanged except that any `.c` file extensions are replaced by `.o`.
- To speed up the link, *pkg*_link_adam scripts are only used selectively. `dtask_link_adam` refers to DTASK, TASK and ERR libraries directly then invokes `subpar_link_adam` which references the necessary libraries directly, apart from HDS, HLP and PSX whose link_adam scripts are invoked.
- `ALINK_FLAGS2` may be useful in controlling the way system libraries are accessed.
- `Additional system libraries` A platform-dependent list of required system libraries which are not searched automatically is added to `alink/ilink` at installation time.
- Other adjustments are made during installation if PCS is not being installed in `/star`. In particular, a -L option is added to include the newly installed libraries before those in `/star/lib`. Similarly with -I options in the C compilation.

The `-xdbx` argument is provided to overcome some awkward problems which can arise when debugging ADAM applications. Usually it is sufficient to include `-g` in the arguments of `alink/ilink` but sometimes, notably when using `xdbx` and `ups` on SunOS, the debuggers do not behave sensibly if required source files are missing so the `-xdbx` argument should be used instead. The effects of the argument are:

- A dummy source file for `dtask_main`, the main routine of every ADAM task, is created in the working directory. The file contains an explanatory message to the user and the name of the user's top-level subroutine (which may be helpful in selecting a breakpoint).
- The source file of DTASK_APPLIC is not deleted from the working directory.
- `-g` is inserted in the compile/link command.

C Available Libraries

ADAM System Libraries

HDSPAR (DAT_ASSOC *etc.*). This library is named DATPAR in the VMS release
SUBPAR
PARSECON
STRING
LEX
DTASK
TASK
MISC (Miscellaneous routines required for Unix.)
AMS (and its subsidiary libraries MSP and SOCK)
ATIMER

Starlink Libraries Searched Automatically

The following separate Starlink libraries will be searched automatically by the ADAM link. The libraries used for ADAM may differ from the stand-alone versions (see relevant documents for details).

PAR
ERR/MSG/EMS
HDS
CHR
PSX
HLP
CNF

Libraries Not Searched Automatically

The following libraries must be included by optional arguments on Unix.

AGI
ARY
AST
GKS (includes GKSPAR)
GNS
GRP
IDI
IMG
NDF
SGS (includes SGSPAR)
PGPLOT (includes PGPPAR)

D ADAM Environment Variables

For more complex operation of ADAM tasks, the user may make use of the following environment variables:

HOME Is expected to specify the user's home directory.

ADAM_USER ADAM_USER may be set to define a directory other than `$HOME/adam` to hold the program parameter files *etc.* (see Section 5).

ADAM_IFL Optionally specifies a search path of directories in which the system is to look for Interface Files. See the Running from the Shell section for details.

ADAM_HELP Specifies a search path of directories in which the parameter system is to look for parameter help files if a full pathname is not specified in the Interface File – it is not usually required.

ADAM_ABBRV If this environment variable is set, keywords used on the command line may be abbreviated to the minimum unambiguous length. Note that there could always be an undetectable ambiguity between logical or special keywords and unquoted strings or names. To alleviate this problem slightly, special keywords (PROMPT, RESET *etc.*) must always be given with a minimum of two characters. This environment variable is set by default.

ADAM_NOPROMPT This will prevent the task from prompting either for parameter values or to resolve an ambiguous keyword. Status PAR__NOUSR will be returned for a parameter prompt, and SUBPAR__CMDSYER if an ambiguous keyword is given on the command line.

ADAM_EXTN Specifies a comma-separated list of extensions to be removed from the end of filenames after filename completion. By default `.sdf` will be removed. See the Filename Completion section for details.

ADAM_TASK_TYPE If set to 'I', this will prevent A-tasks and A-task monoliths resetting active parameter values (and NULL states) after each invocation. Most other aspects of parameter system closedown (such as updating associated GLOBAL variables and unsetting dynamic defaults and MIN/MAX values) will still occur. This is of particular use for Graphical User Interfaces and is unlikely to be set directly by users.

ADAM_EXIT If this environment variable is set when an ADAM task terminates, the calling process will exit with system status set to 1 if the ADAM status was set, or 0 if the ADAM status was SAI__OK.

EMS_PATH Unix ADAM will now use the EMS subroutines for obtaining the message associated with Starlink status values. EMS_PATH may be used to override the default search path for the message files (see SSN/4 for details).

PATH In addition to its use by the system to find the required executable file, the environment variable PATH is used by the parameter system to find the pathname of the file being executed if it was invoked by simply typing its name (not its pathname). This is needed to

discover the directory in which to look for the Interface File if the ADAM_IFL search is unsuccessful. This process means that the use of links may cause confusion – the name and directory of the link will be used.

HDS_SHELL The interpretation of names given as values for parameters accessed via PAR or DAT routines will be handled by HDS. The environment variable HDS_SHELL (see SUN/92) will be effective. If it is not set when the application starts, interpretation with SHELL=0 (`sh`) is selected – thus environment variables and '~' are usually expanded. Note that parameter system syntax will usually prevent the use of more general shell expressions as names.

ICL Environment Variables See Appendix E for details of the environment variables used by ICL.

E ICL Environment Variables

ICL's operation can be controlled by several (optional) environment variables. The variables are:

ICL_LOGIN_SYS, ICL_LOGIN_LOCAL and ICL_LOGIN these may be set to specify ICL command files to be obeyed, in the above order, by ICL before the ICL prompt appears. A default file extension of `.icl` is assumed. For example:

```
% setenv ICL_LOGIN ~/myprocs
```

will cause file `myprocs.icl` in the user's home directory to be loaded as ICL starts up.

ICL_LOGIN_SYS and ICL_LOGIN_LOCAL should be reserved for system use. SSN/64 describes how they are used at Starlink sites.

EDITOR If set, this will override the ICL default editor (vi). For example:

```
% setenv EDITOR tpu
```

ICL_HELPFILE If set, this will override the default search path for the ICL helpfile. The default search path is:

```
../help/icl/iclhelp.shl
```

relative to each of the directories on the user's PATH.

This process for finding the default helpfile only operates if the default helpfile is not defined – *i.e.* initially and after any SET NOHELPFILE command.

SHELL Defines the shell which ICL will use to run Unix commands. If SHELL is undefined, `csh` will be used.

ICL_TASK_NAME This environment variable is set by ICL to the name by which it wants the task to register with the ADAM message system. Other user-interfaces controlling ADAM tasks via the ADAM message system will use the same mechanism. If the environment variable is not set, the task assumes it is being run directly from a shell and does not register with the message system.

In addition to the above, ICL also makes use of some of the environment variables listed in Section D, notably ADAM_USER and ADAM_EXTN.

F Edit Keys

In addition to the left/right arrows and delete key, the following keys may be used in editing the input line:

Key	Function
CNTL/A	Move to Start of Line
CNTL/B	Backward character (same as left arrow)
CNTL/C	Abort (see Section 4.5)
CNTL/D	Delete character, or list choices
CNTL/E	Move to End of Line
CNTL/F	Forward character (same as right arrow)
CNTL/H	Backward delete character (same as delete)
CNTL/K	Delete to End of Line
CNTL/N	Down history (same as down arrow)
CNTL/P	Up history (same as up arrow)
CNTL/R	Redisplay
CNTL/U	Delete Line
CNTL/Z	Suspend
ESC,CNTL/D	List choices
ESC,CNTL/H	Backward delete word
ESC,B	Backward word
ESC,D	Forward Delete word
ESC,F	Forward word

Note that some of these have changed from the undocumented features in ICL prior to Version 3.1-6.