

SUN/110.2

Starlink Project
Starlink User Note 110.2

R.F. Warren-Smith

5th October 1990

Copyright © 2000 Council for the Central Laboratory of the Research Councils

SST - A Simple Software Tools Package

Version 1.0

User's Manual

Abstract

The Simple Software Tools package (SST) is a package of applications designed to help with the production of software and documentation, with particular emphasis on ADAM programming using Fortran 77.

Contents

1	INTRODUCTION	1
2	GETTING STARTED	1
3	OVERVIEW OF SST APPLICATIONS	2
3.1	Producing L ^A T _E X Documentation (PROLAT)	2
3.2	Producing Help Libraries (PROHLP)	3
3.3	Producing STARLSE Package Definitions (PROPAK)	4
3.4	Converting “Old-Style” ADAM/SSE Prologues (PROCVT)	5
3.5	Producing Source-Code Statistics (FORSTATS)	5
A	DESCRIPTIONS OF INDIVIDUAL APPLICATIONS	7
	FORSTATS	8
	PROCVT	10
	PROHLP	12
	PROLAT	14
	PROPAK	17

1 INTRODUCTION

The Simple Software Tools package (SST) is a package of applications designed to help with the production of software and documentation, with particular emphasis on ADAM programming using Fortran 77.

As its name suggests, SST is intended for performing fairly simple manipulation of software, but it aims to tackle some of the commonly encountered problems which are not catered for in more sophisticated commercial software tools (such as FORCHECK and VAXset) which are available on Starlink. In future, SST may also duplicate a few of the simpler facilities which commercial products offer in order to avoid the cost (and generally much higher overheads, such as disk space) which prevent the commercial products being freely available on all Starlink machines.

This document describes the first version of the SST package, whose main initial purpose is to provide tools to extract information from subroutine “prologues” and to format this information in a variety of ways to produce different forms of user documentation. A simple source-code and comment statistics tool is also included.

2 GETTING STARTED

All the applications in the SST package run under ADAM, so before you start you should execute the command:

```
$ ADAMSTART
```

This is a good opportunity to add this command to your LOGIN.COM file if you have not already done so.

The commands within the SST package itself can be used either from the ADAM command language ICL, or from DCL. These commands are initially made available by typing ‘SST’ at the command prompt, *e.g.* from ICL:

```
$ ICL  
ICL> SST
```

or from DCL:

```
$ SST
```

In either case, the following message will be displayed, indicating that the package is ready for use:

```
Simple Software Tools (SST) commands are now available.
```

You can then invoke any of the SST applications by typing the appropriate command (see Appendix A for a detailed description of the applications available).

When working from ICL you can obtain help about SST by using the HELP command, thus:

```
ICL> HELP SST
```

You can also obtain help on each individual application by giving its name, *e.g.*:

```
ICL> HELP FORSTATS
```

will give help on the SST application FORSTATS. “In-line” help on an application’s parameters is also available, in the normal ADAM way, by typing ‘?’ or ‘??’ in response to any prompt requesting a value for a parameter.

3 OVERVIEW OF SST APPLICATIONS

3.1 Producing L^AT_EX Documentation (PROLAT)

The SST application PROLAT is provided to facilitate the production of documentation for packages of applications and subroutine libraries. It works by extracting the necessary information from “prologues” (*i.e.* header comments) within the source-code files which comprise the software, and formatting this information into a L^AT_EX document. The documentation in Appendix A, for example, has been produced by this means.

For PROLAT to work correctly, the prologue information must use the layout generated by the extended Language Sensitive Editor STARLSE (SUN/105), with which a number of the SST applications are designed to integrate. It is also possible to convert some older ADAM/SSE prologue layouts into the required form automatically if required (see §3.4 for details).

To use PROLAT, you should supply a file containing prologue information in the correct format as input, and the resulting output will then be a L^AT_EX document describing the software in the input file (by default this is assumed to be an ADAM application program, otherwise known as an A-task). The document will be written to a file (prolat.tex by default) which can then be processed directly with the L^AT_EX command.

For example, if an application with suitable prologue information resides in the file PROG.FOR, then the following sequence of commands would extract this information and send the derived document to a Canon laser printer:

```
ICL> PROLAT PROG.FOR
ICL> SPAWN LATEX PROLAT
ICL> $DVICAN PROLAT
ICL> $PRCN PROLAT.DVI-CAN
```

When working from ICL, notice how ‘SPAWN’ must be used in front of the L^AT_EX command. This is to overcome problems with the way L^AT_EX writes to the terminal, which can cause ICL

to “hang” if a $\$LATEX$ command is issued directly. The last three commands are, of course, simply the standard processing sequence for sending a \LaTeX source file to a Starlink laser printer (see SUN/9, SUN/12 & SUN/34 for further details of \LaTeX). The entire operation could also be performed from DCL directly.

As with most SST applications, a “wild card” input file may be specified in order to combine a number of program descriptions into a single document. Thus, the following command could be used to document an entire software system if the necessary files resided in the current directory:

```
ICL> PROLAT *.FOR
```

PROLAT has a number of options to tailor its behaviour. For instance, you can specify that you are processing information about a subroutine library (rather than application programs) so as to produce a slightly different documentation format. You can also specify that you want only the descriptive part of the document without any \LaTeX declarations. This latter option is useful if you are adding the information to a larger document which already contains all the necessary \LaTeX commands (*e.g.* as an appendix, such as in the present document).

Note that PROLAT does not attempt to provide a sophisticated text-processing language and does not, therefore, exploit all the layout facilities which \LaTeX has to offer. Instead, it is simply intended for achieving a close approximation to the final form of a document as quickly and easily as possible. If you want to go further and use fancy \LaTeX fonts or an extravagant layout in your final document, then you should be prepared to edit the output file from PROLAT to achieve this. By adopting this approach, the prologue layout can be kept relatively simple, so that it can function as the source for other types of documentation (see below), as well as for possible future documentation formats yet to be invented.

3.2 Producing Help Libraries (PROHLP)

The SST application PROHLP functions in a rather similar way to PROLAT (above), except that its output format is not \LaTeX , but a text file designed for insertion into a help library. By using PROHLP you can therefore make program and subroutine documentation available on-line, as well as simply on paper.

The help file format produced by PROHLP is designed to integrate both with the ADAM help system and also with the STARLSE editor. This means that it is easy to make prologue information available to the person running a program, and also to make subroutine information available to someone using STARLSE (see §3.3 for details of the latter). As before, the initial prologue information must be in the form generated by STARLSE.

To give an example, suppose you have an ADAM application program (an A-task) in the file PROG.FOR, along with suitable prologue information, and you want to make this information available in a help library so that it can be accessed when the program is run. The first step would be to create the library (if you have not already done so), as follows:

```
ICL> $LIBRARY/CREATE/HELP MYLIB.HLB
```

The prologue information is then extracted using PROHLP which, by default, writes it to the file PROHLP.HLP:

```
ICL> PROHLP PROG.FOR
```

The extracted information can then be inserted into the library and the intermediate help file deleted:

```
ICL> $LIBRARY/HELP MYLIB.HLB PROHLP.HLP
ICL> $DELETE PROHLP.HLP;*
```

You can check that the help library contains the correct information by using the DCL command \$HELP with the /LIBRARY qualifier.

This help information can be made available to the application program via the ADAM help system. To do this, the interface (.IFL) file for the application should contain references to the appropriate entries in the help library. Thus, at the start of the interface file, before any parameter definitions, there should be a line such as:

```
helplib 'MYDISK::[ME.PROGS]MYLIB'
```

which identifies the help library to be used. Note that a full directory specification is needed (or an equivalent logical name). Each of the application's parameters for which help information is available should also have a line in its interface file entry such as:

```
helpkey *
```

where '*' specifies that a default set of keywords should be used to access the parameter information in the help library. The library structure which results from using PROHLP is compatible with this set of defaults.

3.3 Producing STARLSE Package Definitions (PROPAK)

If you have developed a subroutine library using STARLSE, with prologue information in the correct form, then the SST application PROPAK can be used to generate an LSE *package definition* to facilitate future use of the library from within the STARLSE editor. This package definition will define LSE templates for each of the routines in the library. It can then be used to make these templates available within the editor, in the same way that the standard ADAM-subroutine templates are provided within STARLSE.

A package definition can be generated by providing the appropriate source code files (containing prologue information) as input to PROPAK, for instance:

```
ICL> PROPAK *.FOR PACK=XYZ
```

Note that a value for the PACK parameter (the name of the package to be produced) must also be given. It will be prompted for if not supplied on the command line.

By default, the output package definition is written to the file PROPAK.LSE, which can then be used to extend your STARLSE editing environment. To do this, the file should be read into an editing buffer and the LSE command 'DO' executed to compile the definition. If you then move to a buffer with a file type of '.FOR' or '.GEN', the routine templates you have just defined will become available for use. Note that LSE allows you to save an extended editing context for future use so that you do not have to repeat this compilation process each time (although the resulting files can be rather large). For details of how to do this the VAXset documentation for LSE should be consulted.

Accessing help libraries. An important feature of PROPAK is its ability to make a package definition refer to a matching help library produced from the same source files using PROHLP. To do so, the HELP parameter should be used with the PROPAK command to specify the (full) name of the help library to which the package should refer. Help library references will then be inserted into the package definition and the associated help information will become available from within STARLSE when the definition is compiled. (To access help on a particular routine, the cursor is placed on the routine name in the editing buffer and the help key GOLD PF2 is used.)

Note that the ATASK parameter should be set to FALSE when using PROHLP for this purpose in order to obtain a help library suitable for a subroutine library, rather than a package of applications programs.

3.4 Converting “Old-Style” ADAM/SSE Prologues (PROCVT)

A considerable amount of ADAM software already in existence uses an “old-style” prologue layout based on templates held in the ADAM_PRO directory. Since there has never been a fully satisfactory method of converting such prologue information into documentation, it is doubtful whether many of these older prologues are of a high enough quality to be used for this purpose now. Nevertheless, in a spirit of optimism, the SST application PROCVT has been provided to permit the conversion of these old-style prologues into the new (STARLSE) form required by other SST applications. PROCVT can also be used as a useful first step in upgrading an old-style prologue, even if the information it contains also needs attention before it becomes suitable for documentation purposes.

Normally, PROCVT will be used with both an input and an output file specified, for instance:

```
ICL> PROCVT OLD.FOR NEW.FOR
```

This command will convert the old-style prologue held in the file OLD.FOR into STARLSE format and will write it (together with all the associated source code) to the file NEW.FOR. Be careful not to use PROCVT on a file which already has a suitable prologue layout – you will not be pleased with the result!

Note that perfect results cannot always be expected with PROCVT unless the original old-style prologue template has been followed very carefully. Consequently, it may sometimes be necessary to use STARLSE afterwards to make fine adjustments before adequate documentation can be produced using other SST commands. To facilitate this, PROCVT will insert appropriate STARLSE placeholders into the output file wherever information is missing.

3.5 Producing Source-Code Statistics (FORSTATS)

The SST application FORSTATS is provided as a “quick look” method of inspecting large Fortran 77 software systems, with the aim of assessing the volume of code present and its likely quality. These two factors (code volume and quality) are usually the most important indicators of the amount of work needed to maintain, or change, an existing software package, especially if you are not initially familiar with it.

Counting lines of code is obviously a simple method of measuring software volume, but the measurement of quality is more difficult and subjective. The approach used by FORSTATS

depends on the observation that if corners are cut during code development, then the first and most obvious sign (at least on Starlink) is usually the omission of in-code comments and prologue information. Past experience with Starlink software has shown that there is an important correlation between the ease of understanding and supporting a large software package and the overall level of commenting which its original author has provided.

FORSTATS attempts to quantify this by producing a number of simple statistics. These are based on line and character counts derived from the code and comments in each program unit of a Fortran 77 software system. It compares these statistics with a set of expected values, which have been derived experimentally from existing Starlink software, and flags values which lie outside the expected range.

When developing software, FORSTATS can be used to highlight those areas where more attention to in-code commentary would be useful to assist future support programmers. Conversely, when inheriting an existing software package, FORSTATS can be used as an aid to identifying possible problem areas before planning changes, *etc.* In a large software system, it can also be invaluable simply for listing the program units present and indicating their size.

To run FORSTATS on a software system contained in a set of '.FOR' files, you might use the command:

```
ICL> FORSTATS *.FOR
```

This command will analyse the specified files and write a statistics file (FORSTATS.LIS by default), which can then be printed. When it is first run, FORSTATS will append an explanatory key to the statistics file to help you interpret its contents. You can suppress this key in future by specifying 'NOKEY' on the command line.

A DESCRIPTIONS OF INDIVIDUAL APPLICATIONS

FORSTATS

Produces statistics on Fortran 77 source code and comments

Description:

This application analyses a sequence of Fortran 77 source code files, divides their contents into program units, and produces statistics about the number and distribution of code and comment lines in each unit. These statistics are compared with typical values expected for well-crafted Fortran 77 code and obvious deviations from the expected values are flagged.

It is intended as a quick-look tool for assessing the size and likely quality of a Fortran 77 software system and is based on the observation that adequate commenting is one of the first things to be sacrificed if corners are cut during code development. A relatively simple analysis of comment and code lines can therefore reveal if this has occurred and can highlight potential trouble spots for visual inspection.

Usage:

```
FORSTATS IN [OUT]
```

Parameters:**IN() = LITERAL (Read)**

A list of (optionally wild-carded) file specifications which identify the Fortran 77 source code files to be used for input. Up to 10 values may be given, but only a single specification such as `'*.FOR'` is normally required. There is no limit to the number of program units which may be held in each input file.

KEY = _LOGICAL (Read)

If KEY is set to TRUE, then an explanatory key will be appended to the output statistics file to describe how to interpret the values it contains. If KEY is set to FALSE, no such key will be appended. This parameter behaves as a latch and remains set to the value previously used until a new value is specified on the command line. [TRUE]

OUT = FILE (Write)

The file to which the statistics derived from the input source code will be written. This file is intended for printing. [FORSTATS.LIS]

Examples:

```
FORSTATS *.FOR
```

Analyses the source code held in the files `*.FOR` and writes a summary of the derived statistics to the default output file `FORSTATS.LIS`. This file may then be printed.

```
FORSTATS SOFTWARE.FOR SOFTWARE.LIS NOKEY
```

Analyses the source code held in the file `SOFTWARE.FOR` and writes the derived statistics to the file `SOFTWARE.LIS`. No explanatory key is written to the output file on this or subsequent invocations of `FORSTATS` until a TRUE value is given for the KEY parameter on the command line.

```
FORSTATS IN=["A*.FOR", "B*.FOR"] OUT=ANALYSIS.LIS
```

In this example, a sequence of input file specifications is given and each will be analysed in turn. The combined statistics will be written to the file ANALYSIS.LIS.

Notes:

This is a general-purpose tool and may be used on any Fortran 77 software system, since it does not depend on any particular layout or format. However, the analysis performed does include a test for the existence of prologue information. For this purpose, prologues are taken to be delimited by '+' or '-' characters appearing in the second column of a comment line. Note that unlike some other SST applications, both these characters are considered as equivalent by FORSTATS; i.e. the first occurrence begins a prologue and the second occurrence ends it, regardless of which character is used.

Timing :

The execution time is approximately proportional to the total number of source code lines supplied for analysis. The time will be increased somewhat if the code resides in a large number of separate files, due to the need to open and close each file.

PROCVT

Converts "old-style" ADAM/SSE prologues into STARLSE format

Description:

This application converts "old-style" ADAM/SSE routine prologues into the format generated by the extended VAX Language Sensitive Editor STARLSE (SUN/105). The converted prologue information may then be used as input to other SST applications which generate documentation, help libraries, etc.

Usage:

```
PROCVT IN [OUT]
```

Parameters:**ATASK = _LOGICAL (Read)**

If ATASK is set to TRUE, then the style of output prologue will be suitable for application programs (i.e. ADAM A-tasks). If it is set to FALSE, then prologues suitable for ordinary subroutines or functions, such as a subroutine library, are produced. [TRUE]

IN() = LITERAL (Read)

A list of (optionally wild-carded) file specifications which identify the Fortran 77 source code files with "old-style" ADAM prologues which are to be used for input. Up to 10 values may be given, but only a single specification such as '*FOR' is normally required. There is no limit on the number of program units which may be held in each input file.

LANG = LITERAL (Read)

This value specifies the programming language (i.e. the particular dialect of Fortran 77) in which the source code is written. It does not affect the processing performed, but will be inserted into the output prologues under the "Language" section heading. If a null (!) or blank value is given, then an appropriate STARLSE placeholder will be used instead. The required entry can then be made later by hand. ['VAX Fortran']

OUT = FILE (Write)

The output file to which the source code will be copied with its prologue format converted. If multiple input files are specified, then the converted output will be concatenated into a single file. [PROCVT.FOR]

Examples:

```
PROCVT MYPROG_OLD.FOR MYPROG_NEW.FOR
```

Reads the application program source code held in the file MYPROG_OLD.FOR and converts its prologue information into STARLSE format. The converted source code is written to the file MYPROG_NEW.FOR.

```
PROCVT *.FOR OUT=SUBS.FOR ATASK=FALSE LANG="Starlink Fortran 77"
```

Reads the source code for a subroutine library, held in the files *.FOR, and converts its prologue information to STARLSE format. The converted source code is written to a single output file SUBS.FOR. The language specified in the converted prologues will be "Starlink Fortran 77".

```
PROCVT IN=["*.FOR", "*.GEN"] NOATASK
```

In this example, a sequence of input file specifications is given. Each will be processed in turn, converting its prologue information into STARLSE format. Output is written to the single default output file PROCVT.FOR.

Notes:

Due to the variety of layouts found in existing "old-style" ADAM/SSE prologues, this application is not guaranteed always to produce a perfect result. Particular problem areas are the inconsistent use of indentation, and the occasional use of section headings and placeholders (of the form '<...>') which do not quite match those given in the original ADAM/SSE prologue templates. Nevertheless, in most cases, PROCVT will produce a result which can be edited fairly easily using STARLSE to conform with the requirements of other SST applications. Appropriate STARLSE placeholders will be inserted into output prologues to facilitate the entry of any essential information which cannot be found in the corresponding input prologue.

Timing :

The execution time is approximately proportional to the total number of source code lines supplied for conversion. The time will be increased somewhat if the code resides in a large number of separate files, due to the need to open and close each file.

PROHLP

Converts routine prologue information into a help library input file

Description:

This application reads a series of Fortran 77 source code files containing prologue information formatted using STARLSE (SUN/105) and produces an output file containing user documentation for each routine in a format suitable for insertion into a help library. The help library format may be chosen to suit either a package of application programs (i.e. ADAM A-tasks) or a set of ordinary subroutines or functions, such as a subroutine library.

Usage:

```
PROHLP IN [OUT]
```

Parameters:**ATASK = _LOGICAL (Read)**

If ATASK is set to TRUE, then a help library format suitable for a package of application programs (i.e. ADAM A-tasks) is produced. If it is set to FALSE, then the help library format produced is suitable for a subroutine library. [TRUE]

IN() = LITERAL (Read)

A list of (optionally wild-carded) file specifications which identify the Fortran 77 source code files to be used for input. Up to 10 values may be given, but only a single specification such as '*.FOR' is normally required.

If the SINGLE parameter is set to TRUE (the default), then only a single prologue will be expected in each input file. If it is set to FALSE, then there is no limit to the number of prologues which may be held in each input file.

OUT = FILE (Write)

The output file to which the help information will be written. The information in this file will need to be inserted into a help library for use. [PROHLP.HLP]

SINGLE = _LOGICAL (Read)

If SINGLE is set to TRUE, then only a single prologue will be expected at the start of each input file; if the file contains more than one prologue, then the remaining ones will be ignored. If SINGLE is set to FALSE, then each input file will be searched for all the prologues it contains and each will be processed in turn. When appropriate, the former option (the default) will result in faster execution since only the initial prologue information must then be read, rather than the entire contents of each input file. [TRUE]

Examples:

```
PROHLP PROG.FOR PROG.HLP
```

Extracts prologue information from the application program source code held in the file PROG.FOR and produces a help library entry for it. The program's name is used as the level-1 help keyword. The output is written to the file PROG.HLP.

```
PROHLP *.FOR OUT=SUBS.HLP ATASK=FALSE
```

Extracts prologue information for a subroutine library, whose source code resides in the files *.FOR, one routine per file. Help library entries describing the routines are written to the file SUBS.HLP, with each routine's name being used as a level-1 help keyword.

```
PROHLP IN=SOURCE.FOR NOATASK NOSINGLE
```

Extracts prologue information from a sequence of subroutines or functions which are all held in the file SOURCE.FOR and produces help library entries for them in the default output file PROHLP.HLP. Each routine name is used to form a separate level-1 help keyword.

```
PROHLP IN=["A*.FOR", "B*.FOR"] NOATASK
```

In this example, a sequence of input file specifications is given. Each will be processed in turn to generate a help file entry from the first prologue encountered in each file. Output is written to the file PROHLP.HLP. Once again, each routine name generates a separate level-1 help keyword.

Notes:

Care must be taken to ensure that begin-prologue and end-prologue lines (starting '*+' and '*-' respectively) appear before and after each prologue and that '+' and '-' symbols are not used in the second column elsewhere in the file, otherwise prologues may not be correctly identified.

Timing :

The execution time is approximately proportional to the amount of information to be read from the input files. In addition, the time will be increased somewhat if the input code resides in a large number of separate files, due to the need to open and close each file. If there is only one prologue in each input file, then execution time will be minimised if SINGLE is set to TRUE, since only the initial prologue information need then be read, rather than the entire contents of each file.

PROLAT

Converts routine prologue information into Latex documentation

Description:

This application reads a series of Fortran 77 source code files containing prologue information formatted using STARLSE (SUN/105) and produces an output file containing Latex user documentation for each routine. The documentation format may be chosen to suit either a package of application programs (i.e. ADAM A-tasks) or a set of ordinary subroutines or functions, such as a subroutine library.

Usage:

PROLAT IN [OUT]

Parameters:**ATASK = _LOGICAL (Read)**

If ATASK is set to TRUE, then a style of documentation suitable for a package of application programs (i.e. ADAM A-tasks) is produced. If it is set to FALSE, then the documentation produced is suitable for a subroutine library. [TRUE]

DOCUMENT = _LOGICAL (Read)

If DOCUMENT is set to TRUE, then the output file will be a complete Latex document and will contain all necessary layout definitions, etc. Such a file may be passed directly to Latex for processing. If DOCUMENT is set to FALSE, then the output file will contain only the information extracted from the input files. In this case, additional Latex commands must be added to produce a complete document (see the Latex Definitions section for details). This latter option would typically be used when the output is to be included in a larger document which already contains the necessary definitions. [TRUE]

IN() = LITERAL (Read)

A list of (optionally wild-carded) file specifications which identify the Fortran 77 source code files to be used for input. Up to 10 values may be given, but only a single specification such as '*FOR' is normally required.

If the SINGLE parameter is set to TRUE (the default), then only a single prologue will be expected in each input file. If it is set to FALSE, then there is no limit to the number of prologues which may be held in each input file.

OUT = FILE (Write)

The output file to which the Latex documentation will be written. [prolat.tex]

PAGE = _LOGICAL (Read)

If PAGE is set to TRUE, then a new output page will be started to hold the information extracted from each input prologue. If PAGE is set to FALSE, then this will not occur. The default behaviour is to start each prologue description on a new page for applications packages (ADAM A-tasks), but for routine descriptions in subroutine libraries to follow end-to-end. []

SINGLE = _LOGICAL (Read)

If SINGLE is set to TRUE, then only a single prologue will be expected at the start of

each input file; if the file contains more than one prologue, then the remaining ones will be ignored. If SINGLE is set to FALSE, then each input file will be searched for all the prologues it contains and each will be processed in turn. When appropriate, the former option (the default) will result in faster execution since only the initial prologue information must then be read, rather than the entire contents of each input file. [TRUE]

Examples:

```
PROLAT MYPROG.FOR MYPROG.TEX
```

Extracts prologue information from the application program source code held in the file MYPROG.FOR and produces a Latex user document for it. The Latex output is written to the file MYPROG.TEX.

```
PROLAT *.FOR OUT=SUBS.TEX ATASK=FALSE
```

Extracts prologue information for a subroutine library, whose source code resides in the files *.FOR, one routine per file. A Latex document describing the routines is written to the file SUBS.TEX.

```
PROLAT IN=SOURCE.FOR NOATASK NODOCUMENT NOSINGLE
```

Extracts prologue information from a sequence of subroutines or functions which are all held in the file SOURCE.FOR and produces Latex output in the default output file prolat.tex. This output file contains only the routine descriptions (no Latex definitions) and is therefore suitable for inclusion in a larger document.

```
PROLAT IN=["*.FOR", "*.GEN"] NOATASK PAGE
```

In this example, a sequence of input file specifications is given. Each will be processed in turn to generate Latex documentation from the first prologue encountered in each file. Output is written to the file prolat.tex with the description of each routine starting on a new page.

Notes:

Care must be taken to ensure that begin-prologue and end-prologue lines (starting '*+' and '*-' respectively) appear before and after each prologue and that '+' and '-' symbols are not used in the second column elsewhere in the file, otherwise prologues may not be correctly identified.

Timing :

The execution time is approximately proportional to the amount of information to be read from the input files. In addition, the time will be increased somewhat if the input code resides in a large number of separate files, due to the need to open and close each file. If there is only one prologue in each input file, then execution time will be minimised if SINGLE is set to TRUE, since only the initial prologue information need then be read, rather than the entire contents of each file.

Latex Definitions :

If the DOCUMENT=FALSE option is chosen, then the output file will contain none of the Latex command definitions needed to produce the final document. To define these commands, the contents of the file \$STARLINK_DIR/share/sst.tex must be included in the Latex input file in the preamble (in the standard SUN template). The layout definitions in this file are designed to operate correctly within the Latex environment normally used in a Starlink User Note (see the file \$STARLINK_DIR/share/sun.tex).

It is recommended that you include the contents of the file \$STARLINK_DIR/share/sst.tex in your final document explicitly rather than by using the Latex \include directive, otherwise it may not be possible to process the document in future if changes have to be made to the Latex definitions in this file.

PROPAK

Converts routine prologue information into a STARLSE package definition

Description:

This application reads a series of Fortran 77 source code files containing prologue information formatted using STARLSE (SUN/105) and produces an output file containing an LSE package definition suitable for use with STARLSE. Help library references may be included if required, allowing the package definition to access help information extracted using the PROHLP application.

Usage:

PROPAK IN [OUT] PACK [HELP]

Parameters:**HELP = LITERAL (Read)**

Name of the help file to which the package definition should refer for on-line help information (suitable help libraries may be produced using the PROHLP application). The full file name, including a directory name, should be given. Logical names may also be used. The help file need not actually exist at the time PROPAK is run.

If a null (!) or blank value is given for this parameter, then no help library references will be included in the package definition. [!]

IN() = LITERAL (Read)

A list of (optionally wild-carded) file specifications which identify the Fortran 77 source code files to be used for input. Up to 10 values may be given, but only a single specification such as '*FOR' is normally required.

If the SINGLE parameter is set to TRUE (the default), then only a single prologue will be expected in each input file. If it is set to FALSE, then there is no limit to the number of prologues which may be held in each input file.

OUT = FILE (Write)

The output file to which the STARLSE package definition will be written. The VAXset documentation on LSE should be consulted for details of how to use this file to extend the STARLSE editing environment. [PROPAK.LSE]

PACK = LITERAL (Read)

This value is the name of the LSE package to be generated. To avoid possible clashes with existing STARLSE packages, the prefix characters used on the routines themselves are recommended for use as the package name. For example, if routines in the package have names such as XYZ_ROUTN, then "XYZ" should be used as the package name.

SINGLE = _LOGICAL (Read)

If SINGLE is set to TRUE, then only a single prologue will be expected at the start of each input file; if the file contains more than one prologue, then the remaining ones will be ignored. If SINGLE is set to FALSE, then each input file will be searched for

all the prologues it contains and each will be processed in turn. When appropriate, the former option (the default) will result in faster execution since only the initial prologue information must then be read, rather than the entire contents of each input file. [TRUE]

Examples:

```
PROPAC MPK_*.FOR MYPACK.LSE MPK
```

Extracts prologue information from the routines held in the files MPK_*.FOR, one per file, and produces a STARLSE package definition for them. The package is called "MPK" and its definition is written to the file MYPACK.LSE.

```
PROPAC *.FOR PACK=TEST HELP=TEST_DIR:TESTHELP
```

Extracts prologue information for a subroutine library, whose source code resides in the files *.FOR, one routine per file. The resulting definition, of a STARLSE package called "TEST", is written to the default output file PROPAC.LSE. The package contains references to the help library TEST_DIR:TESTHELP, from which on-line help may be obtained when using the package within STARLSE.

```
PROPAC IN=SOURCE.FOR PACK=MYLIB NOSINGLE
```

Extracts prologue information from a sequence of subroutines or functions, all of which are held in the file SOURCE.FOR, and defines a STARLSE package called "MYLIB" in the default output file PROPAC.TEX.

```
PROPAC IN=["A*.FOR", "B*.FOR"] HELP=PACK_HELP
```

In this example, a sequence of input file specifications is given. Each will be processed in turn to generate a combined package definition which refers to a help library with the logical name PACK_HELP. The package name will be prompted for.

Notes:

Care must be taken to ensure that begin-prologue and end-prologue lines (starting '*+' and '*-' respectively) appear before and after each prologue and that '+' and '-' symbols are not used in the second column elsewhere in the file, otherwise prologues may not be correctly identified.

Timing :

The execution time is approximately proportional to the amount of information to be read from the input files. In addition, the time will be increased somewhat if the input code resides in a large number of separate files, due to the need to open and close each file. If there is only one prologue in each input file, then execution time will be minimised if SINGLE is set to TRUE, since only the initial prologue information need then be read, rather than the entire contents of each file.