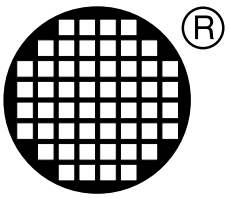

inmos®

occam 2 User Library Specifications

Steven R Huggins

SW-0114-4

APPROVED 5 February, 1991

Contents

1 Introduction

This document is to give the specifications of the user visible entry points in the occam 2 toolset libraries. The toolset in question is the **D7205**, for the IBM/NEC PC, the **D6205**, for the VAX, the **D5205**, for the Sun3, and the **D4205**, for the Sun4. If the specifications have already been written elsewhere then the appropriate document is referred to, otherwise the specification of the routine is given here.

All visible entry entry points are ones that are intended for the user and are documented in a user accessible place e.g. **occam 2 toolset User Manual**; however, documentation for the user and specifications are two different things, hence this document.

A total of 383 entry points are mentioned here.

2 Related Documents

Reference is made to the following documents and publications:

occam 2 Reference Manual by INMOS Limited, Prentice Hall, Hemel Hempstead, 1988. Prentice Hall International Series in Computer Science. ISBN 0-13-629312-3

Transputer Development System by INMOS Limited, Prentice Hall, Hemel Hempstead; 1st edition, 1988, ISBN 0-13-928995-X; 2nd edition, 1988, 1990, ISBN 0-13-929068-0.

SW-0044 occam-2 language implementation manual, Conor O'Neill

SW-0102 Iserver 1.42 Specification, Bob Green

3 Definitions

entry point a procedure or function identifier

visible entry point an entry point that is a legal occam identifier

4 Compiler Libraries

55 entry points

The compiler libraries are **occam2.lib**, **occama.lib**, **occam8.lib**, **occamutl.lib** and **virtual.lib**. Only the user visible entry points are included here. This then means that the contents of **occamutl.lib** and **virtual.lib** are not mentioned. The compiler libraries are the only libraries which do not need to be #USEd in the source code of a user's routine or program.

Entry Point	Where Documented
DRAW2D	SW-0044
CLIP2D	”
MOVE2D	”

BITCOUNT	”
CRCWORD	”
CRCBYTE	”
BITREVNBITS	”
BITREVWORD	”
FRACMUL	”
UNPACKSN	”
ROUNDSN	”
IEEE32REM	”
IEEE64REM	”
REAL64EQ	as for REAL32EQ, but operates on REAL64’s
REAL64GT	as for REAL32GT, but operates on REAL64’s
REAL32OP	occam 2 Reference Manual, Appendix M
IEEE32OP	”
REAL32REM	”
REAL32EQ	”
REAL32GT	”
REAL64OP	”
IEEE64OP	”
REAL64REM	”
IEEECOMPARE	”
DIEEECOMPARE	”
ABS	occam 2 Reference Manual, Appendix K
FPINT	”
ISNAN	”
ORDERED	”
NOTFINITE	”
SCALEB	”
FLOATING.UNPACK	”
ARGUMENT.REDUCE	”
MULBY2	”
DIVBY2	”
LOGB	”
MINUSX	”
COPYSIGN	”
NEXTAFTER	”
SQRT	”
DABS	”
DFPINT	”
DISNAN	”
DORDERED	”
DNOTFINITE	”
DSCALEB	”
DFLOATING.UNPACK	”
DARGUMENT.REDUCE	”
DMULBY2	”
DDIVBY2	”
DLOGB	”
DMINUSX	”
DCOPYSIGN	”
DNEXTAFTER	”
DSQRT	”

5 debug.lib

4 entry points

```
PROC DEBUG.ASSERT (VAL BOOL assertion)

  -- Purpose:  To possibly stop the calling process, acting according to the
  --           value of an assertion:-
  --           assertion TRUE:    No effect.
  --           assertion FALSE :  Action depends on whether debugger is running:
  --           Debugger present:  sends an error message to the debugger along
  --                              with Iptr and Wdesc for caller and stops process
  --                not present:  stops process or processor depending on error
  --                              mode
  -- In:       assertion - the value of the assertion made in the call of this
  --           routine


PROC DEBUG.MESSAGE (VAL []BYTE message)

  -- Purpose:  To send a message to the debugger which is to be displayed
  --           with normal program output.  The precise action depends on
  --           whether the debugger is running or not:-
  --           Debugger present:  sends a message to the debugger along with
  --                              the Iptr and Wdesc for the caller
  --                not present:  no action
  -- In:       message - the message to send to the debugger


PROC DEBUG.STOP ()

  -- Purpose:  To stop the calling process, first sending information to
  --           the debugger if it is running:-
  --           Debugger present:  sends an error message to the debugger
  --                              along with Iptr and Wdesc for caller and
  --                              stops process
  --                not present:  stops process or processor depending on
  --                              error mode


PROC DEBUG.TIMER (CHAN OF INT stop)

  -- Purpose:  To enable the debugger to find a way into a
  --           deadlocked program by always ensuring there
  --           is a process on the timer or the run queues.
  -- Channels: stop - for input
  -- Notes:    This procedure will continually wait on the
  --           timer until it is told to terminate by the
  --           receipt of any value on the stop input channel.
  --           The odds are that this process will be found
  --           on the timer queue.
  --           This process may be run at any priority on
  --           16 and 32 bit processors.
```

6 convert.lib

22 entry points

As regards the **HEX** routines here, some extra information is in order. Only capital hex digits are allowed (i.e. A - F rather than a - f), and also no # is allowed on the hex number for the **STRINGTOHEX** and **STRINGTOHEXxx** routines, and a # is not output by the **HEXTOSTRING** and **HEXxxTOSTRING** routines.

Entry Point	Where Documented
STRINGTOINT	occam 2 Reference Manual, Appendix O
INTTOSTRING	”
STRINGTOHEX	”
HEXTOSTRING	”
STRINGTOBOOL	”
BOOLTOSTRING	”
STRINGTOINT16	”
INT16TOSTRING	”
STRINGTOINT32	”
INT32TOSTRING	”
STRINGTOINT64	”
INT64TOSTRING	”
STRINGTOHEX16	”
HEX16TOSTRING	”
STRINGTOHEX32	”
HEX32TOSTRING	”
STRINGTOHEX64	”
HEX64TOSTRING	”
STRINGTOREAL32	”
STRINGTOREAL64	”
REAL32TOSTRING	see below for latest spec.
REAL64TOSTRING	”

```
PROC REAL32TOSTRING (INT len, []BYTE string, VAL REAL32 X, VAL INT Ip, Dp)

  -- Purpose:  To take a binary representation of a real number and convert
  --           it into a string of characters that is the decimal
  --           representation of that number, formatted according to given
  --           rules ( see Notes below ).
  -- Out:      len - the number of characters ( BYTEs ) of string occupied
  --           by the formatted decimal representation of the real number
  -- Out:      string - an array containing the formatted decimal
  --           representation of the real number in the first len bytes,
  --           the remaining bytes being undefined
  -- In:       X - the real number, in IEEE format, to be converted
  -- In:       Ip - the first of two formatting values
  -- In:       Dp - the second of two formatting values
  -- Notes:    Rounding mode is round to nearest.
  --           Which format is used depends on the combination of values
  --           of Ip, Dp and X.  In all cases, any digits beyond the 9th
  --           significant digit for single precision or 17th significant
  --           digit for double precision will be given as 0 and cannot
  --           be considered accurate.
  --           If string overflows this routine acts as an invalid process.
```

```
  -- Case (i):    Ip = 0, Dp = 0  => free format
  --           Where possible a fixed point representation is used.  If
  --           it is not used then exponential form is used.  It is not
  --           used if more than 9 | 17 significant digits of accuracy
  --           ( single | double ) would be required before the decimal
  --           point, or if there are more than 3 zeroes after the decimal
  --           point before the first significant digit.  In any case, the
  --           maximum number of characters returned in string is 15 for a
  --           single precision X, and 24 for a double precision X.
  --           string is left justified.
  --           If X is infinity or a NaN, then the string will contain one
  --           of "Inf", "-Inf" or "NaN", but not the quotes.
  -- Case (ii):   Ip = 0, Dp > 0  => exponential format
  --           The form of exponential format is, firstly either a minus
  --           sign or a space ( this latter instead of an explicit plus
  --           sign ), a fraction in the form <digit>.<digits>, the
  --           exponential symbol ( E ), the sign of the exponent ( + or
  --           - ), then the exponent, which is two digits for a single
  --           precision X, three digits for a double precision X.  Dp
  --           gives the number of digits in the fraction ( 1 before the
  --           point, and the others after, but not counting the point ).
  --           The total number of characters in string is ( Dp + 6 )
  --           for a single precision X, and ( Dp + 7 ) for a double
  --           precision X.
  --           If Dp = 1 then the fraction is of the form <space>digit
  --           ( which, note, will not result in occam syntax for the
  --           real ).
  --           If X is infinity or a NaN, then the string will contain one
  --           of " Inf", "-Inf" or " NaN", but not the quotes, padded on
  --           the right to fill the field width.
  -- Case (iii):  Ip > 0, Dp > 0  => fixed point if possible
  --           Ip gives the number of places before the point, not
  --           counting the sign place; Dp the number of places after
  --           the point.  Padding spaces are added on the left as
  --           necessary.  If the number will not fit the format,
  --           then an exponential format is used with the same field
  --           width as the fixed point representation would have had.
  --           If Ip and Dp are very small then an exponential
  --           representation may not fit in the field width so the
  --           special value "Ov" with a sign is returned.
  --           There are always ( Ip + Dp + 2 ) characters in string,
  --           the 2 being the decimal point and the sign ( - or space ).
  --           If X is infinity or a NaN, then the string will contain one
  --           of " Inf", "-Inf" or " NaN", but not the quotes, padded on
  --           the right to fill the field width.
  -- All other combinations of Ip and Dp are meaningless and will cause
  --           an error.

PROC REAL64TOSTRING (INT len, []BYTE string, VAL REAL64 X, VAL INT Ip, Dp)

  -- As REAL32TOSTRING but for double precision reals
```

7 crc.lib

2 entry points

```
INT FUNCTION CRCFROMLSB (VAL []BYTE InputString,
                         VAL INT PolynomialGenerator,
                         VAL INT OldCRC)

  -- Purpose:  To calculate a CRC value for a given string, starting from
  --           the least significant bit.
  -- Returned: The CRC value calculated for InputString with generator
  --           PolynomialGenerator and initial CRC value OldCRC.
  -- In:       InputString - the string for which a CRC value is desired
  -- In:       PolynomialGenerator - an integer which acts somewhat like
  --           an encryption key in that it is something which remains
  --           constant during the calculation but on which the CRC value
  --           intimately depends
  -- In:       OldCRC - this is used only as the initial CRC value during
  --           the iteration over the string, which does mean that the
  --           final CRC value ( the value that is returned ) depends on it
  -- Notes:    The string of bytes is polynomially divided starting from
  --           the least significant bit of the least significant byte in
  --           inreasing bit order.

INT FUNCTION CRCFROMMSB (VAL []BYTE InputString,
                         VAL INT PolynomialGenerator,
                         VAL INT OldCRC)

  -- Purpose:  To calculate a CRC value for a given string, starting from
  --           the most significant bit.
  -- Returned: The CRC value calculated for InputString with generator
  --           PolynomialGenerator and initial CRC value OldCRC.
  -- In:       InputString - the string for which a CRC value is desired
  -- In:       PolynomialGenerator - an integer which acts somewhat like
  --           an encryption key in that it is something which remains
  --           constant during the calculation but on which the CRC value
  --           intimately depends
  -- In:       OldCRC - this is used only as the initial CRC value during
  --           the iteration over the string, which does mean that the
  --           final CRC value ( the value that is returned ) depends on it
  -- Notes:    The string of bytes is polynomially divided starting from
  --           the most significant bit of the most significant byte in
  --           decreasing bit order.
```

8 hostio.lib

103 entry points

The routines in **hostio.lib** are meant as the user's interface to the server, and hence the host

machine. The routines are therefore dependent on the functionality of the iserver; see SW-0102. Constants referred to in the following specifications are to be found in the include file **hostio.inc**.

A naming convention has been adopted for these routines. The basic ones use the server protocol directly and map directly onto server functions. These have the prefix '**sp.**'. Routines which use the basic routines have the prefix '**so.**'. In many cases the mapping between the '**so.**' routines and the '**sp.**' routines is one to one.

Note that in some routines with a non-**VAL** parameter of type **[]BYTE**, although the value on entry to the routine of the array is irrelevant, the **SIZE** of the array is important. In these routines such an array parameter is nevertheless described simply as '**Out**'.

```
PROC sp.open (CHAN OF SP fs, ts, VAL []BYTE name, VAL BYTE type, mode,
              INT32 streamid, BYTE result)

  -- Purpose:  To open a file.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       name - the name of the file to be opened.  The name of the file
  --           must fit exactly into name, i.e. there are (SIZE name) characters
  --           in the name of the file.  A directory specification may form
  --           part of name.
  -- In:       type - either
  --             spt.binary, for a file containing raw bytes only
  --           or
  --             spt.text, for a file stored as text records separated by newlines
  -- In:       mode - one of
  --             spm.input            open existing file for reading
  --             spm.output           open new file, or truncate existing one, for
  --                                  writing
  --             spm.append           open new file, or append to existing one,
  --                                  for writing
  --             spm.existing.update  open existing file for update ( reading and
  --                                  writing ), starting at the beginning of the
  --                                  file
  --             spm.new.update       open new file, or truncate existing one,
  --                                  for update
  --             spm.append.update    open new file, or append to existing one,
  --                                  for update
  --           When a file is opened for update then the resulting stream may
  --           be used for input and output.
  -- Out:      streamid - if result is spr.ok, the identification number of the
  --           stream associated with the file that was opened; undefined
  --           otherwise
  -- Out:      result - spr.ok if the file was successfully opened; otherwise
  --           it takes on a value indicating what went wrong:
  --             spr.bad.packet.size   name too large:
  --                                     (SIZE name) > sp.max.openname.size
  --             spr.bad.name          null file name
  --             spr.bad.type          invalid type
  --             spr.bad.mode          invalid mode
  --           >=spr.operation.failed  the open failed - see hostio.inc or
  --                                   iserver documentation for further details
  -- Notes:    No path environment variable is used to locate the file; what
  --           is in name is all that is used ( cf. so.popen.read )
```

```
PROC so.open (CHAN OF SP fs, ts, VAL []BYTE name, VAL BYTE type, mode,
              INT32 streamid, BYTE result)

  -- As sp.open


PROC sp.close (CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result)

  -- Purpose:  To close a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream to be
  --           closed
  -- Out:      result - equal to spr.ok if the stream closed properly;
  --           otherwise >= spr.operation.failed - refer to hostio.inc or
  --           iserver documentation for further details
  -- Notes:    Before closing the stream unwritten data is flushed or any
  --           unread buffered input is discarded.


PROC so.close (CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result)

  -- As sp.close


PROC sp.read (CHAN OF SP fs, ts, VAL INT32 streamid,
              INT bytes.read, []BYTE data, BYTE result)

  -- Purpose:  To read a limited number of bytes from a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream to be
  --           read from
  -- Out:      bytes.read - if result is spr.ok then this gives the number
  --           of bytes read from the file; these bytes will be
  --           [data FROM 0 FOR bytes.read]; if bytes.read <> (SIZE data)
  --           then either an error occurred or the end of the file was
  --           reached.  If result is not spr.ok then bytes.read will be zero.
  --           0 <= bytes.read <= sp.max.readbuffer.size
  -- Out:      data - if result is spr.ok then this contains the bytes read
  --           from the file; otherwise it is undefined.  The number of bytes
  --           requested to be read is (SIZE data); the limit to this is
  --           sp.max.readbuffer.size.
  -- Out:      result - equal to spr.bad.packet.size if too many bytes asked for
  --           ie (SIZE data) > sp.max.readbuffer.size, otherwise it is equal
  --           to spr.ok


PROC so.read (CHAN OF SP fs, ts, VAL INT32 streamid,
              INT bytes.read, []BYTE data)

  -- Purpose:  To read an unlimited number of bytes from a stream.
  -- Channels: fs - from server
  --           ts - to server
```

```
  -- In:       streamid - the identification number of the open stream to be
  --           read from
  -- Out:      bytes.read - this gives the number of bytes read from the file;
  --           these bytes will be [data FROM 0 FOR bytes.read]; if
  --           bytes.read <> (SIZE data) then either an error occurred or
  --           the end of the file was reached.
  -- Out:      data - this contains the bytes read from the file; the number
  --           of bytes requested to be read is (SIZE data).

PROC sp.write (CHAN OF SP fs, ts, VAL INT32 streamid, VAL []BYTE data,
               INT length.written, BYTE result)

  -- Purpose:  To write a limited number of bytes to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream to
  --           write to
  -- In:       data - a string of bytes ALL of which are to be written to
  --           the stream
  -- Out:      length.written - this gives the number of bytes written.
  --           If length.written <> (SIZE data) then an error occurred.
  --           0 <= length.written <= sp.max.writebuffer.size
  -- Out:      result - if not equal to spr.bad.packet.size meaning that too
  --           many bytes were requested, ie (SIZE data) > sp.max.writebuffer.size,
  --           then it is equal to spr.ok

PROC so.write (CHAN OF SP fs, ts, VAL INT32 streamid, VAL []BYTE data,
               INT length)

  -- Purpose:  To write an unlimited number of bytes to a stream.
  -- channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream to
  --           write to
  -- In:       data - a string of bytes ALL of which are to be written to
  --           the file
  -- Out:      length - this gives the number of bytes written;
  --           if length <> (SIZE data) then an error occurred

PROC sp.gets (CHAN OF SP fs, ts, VAL INT32 streamid,
              INT bytes.read, []BYTE data, BYTE result)

  -- Purpose:  To read a line from the specified input stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream to be
  --           read from
  -- Out:      bytes.read - if result is spr.ok this is the number of bytes
  --           read; if result is spr.buffer.overflow then this will be
  --           (SIZE data) and will be less than the actual number of bytes
  --           read from the file; otherwise zero.
  -- Out:      data - an array of bytes holding line read if result is spr.ok;
  --           or holding a truncation of the line if result is
```

```
  --           spr.buffer.overflow.  It is undefined otherwise.  If result
  --           is spr.ok or spr.buffer.overflow then the bytes will be
  --           [data FROM 0 FOR bytes.read].
  -- Out:      result - equal to spr.ok if line was successfully read; otherwise
  --           takes on a value indicating what went wrong:
  --             spr.bad.packet.size   data is too large,
  --                                     (SIZE data) > sp.max.readbuffer.size
  --             spr.buffer.overflow   data not large enough to hold the line,
  --                                   in which case the line is truncated to fit
  --           >=spr.operation.failed  the read failed, either because the end of
  --                                   the file has been reached or some error
  --                                   occurred - see hostio.inc or iserver
  --                                   documentation for further details
  -- Notes:    Characters are read until a newline sequence is found, the end of
  --           the file is reached, or the number of characters is not less than
  --           sp.max.readbuffer.size.
  --           If a newline sequence is found, it is not included in data.

PROC so.gets (CHAN OF SP fs, ts, VAL INT32 streamid,
              INT bytes.read, []BYTE data, BYTE result)

  -- As sp.gets

PROC sp.puts (CHAN OF SP fs, ts, VAL INT32 streamid, VAL []BYTE data,
              BYTE result)

  -- Purpose:  To write a line to the specified output stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to be
  --           written to
  -- In:       data - the line to be written ( without a terminating newline
  --           sequence )
  -- Out:      result - equal to spr.ok if the line was successfully written;
  --           otherwise takes on a value indicating what was wrong:
  --             spr.bad.packet.size   too many bytes supplied,
  --                                     (SIZE data) > sp.max.writebuffer.size
  --           >=spr.operation.failed  the write failed - see hostio.inc or
  --                                   iserver documentation for further details
  -- Notes:    A newline sequence is added to the end of the bytes to be written.

PROC so.puts (CHAN OF SP fs, ts, VAL INT32 streamid, VAL []BYTE data,
              BYTE result)

  -- As sp.puts

PROC sp.flush (CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result)

  -- Purpose:  To flush the specified output stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream to be
```

```
  --           flushed
  -- Out:      result - equal to spr.ok stream successfully flushed; otherwise
  --           >= spr.operation.failed - see hostio.inc or iserver
  --           documentation for further details
  -- Notes:    To flush means to write out any internally buffered data to
  --           the stream that it is associated with.
  --           The stream remains open.

PROC so.flush (CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result)

  -- As sp.flush

PROC sp.seek (CHAN OF SP fs, ts, VAL INT32 streamid, offset, origin, BYTE result)

  -- Purpose:  To set the file position for the specified stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream
  --           associated with the file the position of which is to be set
  -- In:       offset - the offset from origin of the new position for reading
  --           or writing.  For a binary file the new position will be offset
  --           bytes, perhaps negative, from the position defined by origin.
  --           For a text file offset must be zero or a value returned by
  --           so.tell; in the latter case origin must be spo.start, and
  --           offset greater than or equal to zero.
  -- In:       origin - one of
  --             spo.start    the start of the file
  --             spo.current  the current position in the file
  --             spo.end      the end of the file
  -- Out:      result - equal to spr.ok if the file position was successfully
  --           set; otherwise takes on a value indicating what the problem was:
  --             spr.bad.origin        invalid origin
  --           >=spr.operation.failed  the seek failed - see hostio.inc or
  --                                   iserver documentation for further details

PROC so.seek (CHAN OF SP fs, ts, VAL INT32 streamid, offset, origin, BYTE result)

  -- As sp.seek

PROC sp.tell (CHAN OF SP fs, ts, VAL INT32 streamid,
              INT32 position, BYTE result)

  -- Purpose:  To return the current file position for the specified stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the open stream associated
  --           whith the file the reading/writing position of which is desired
  -- Out:      position - the current file position
  -- Out:      result - equal to spr.ok if the file position determined;
  --           otherwise >= spr.operation.failed - refer to hostio.inc or
  --           iserver documentation for further details
```

```
PROC so.tell (CHAN OF SP fs, ts, VAL INT32 streamid,
              INT32 position, BYTE result)

 -- As sp.tell

PROC sp.eof (CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result)

  -- Purpose:  To test whether the specified stream has end of file status
  --           or not.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to test
  -- Out:      result - equals spr.ok if end of file status is set;
  --           >= spr.operation.failed if the end of file status has not
  --           been set

PROC so.eof (CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result)

  -- As sp.eof

PROC sp.ferror (CHAN OF SP fs, ts, VAL INT32 streamid,
                INT32 error.no, INT length, []BYTE message, BYTE result)

  -- Purpose:  To test for the occurrence of an error on the specified stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to be tested
  --           for error
  -- Out:      error.no - a host defined error number
  -- Out:      length - the number of bytes in message.  Equal to zero
  --           if result >= spr.operation.failed.
  -- Out:      message - if result is not spr.operation.failed then this
  --           contains a ( possibly null ) message corresponding to the last
  --           file error on the specified stream; this message is
  --           [message FROM 0 FOR length]
  -- Out:      result - equals spr.ok if there was an error on the specified
  --           stream and the message corresponding to it fits into message;
  --           equals spr.buffer.overflow if there was an error on the
  --           specified stream but the message corresponding to it is too
  --           large to fit into message, in which case it is truncated to fit;
  --           >= spr.operation.failed if there was no error on the
  --           specified stream

PROC so.ferror (CHAN OF SP fs, ts, VAL INT32 streamid,
                INT32 error.no, INT length, []BYTE message, BYTE result)

 -- As sp.ferror

PROC sp.remove (CHAN OF SP fs, ts, VAL []BYTE name, BYTE result)
```

```
  -- Purpose:  To delete the specified file.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       name - the name of the file to be deleted; the filename must
  --           fill all of the bytes of name
  -- Out:      result - equals spr.ok if the file was removed; otherwise
  --           takes on a value indicating what went wrong:
  --             spr.bad.packet.size   name too large,
  --                                     (SIZE name) > sp.max.removename.size
  --             spr.bad.name          name is null
  --           >=spr.operation.failed  the delete failed - see hostio.inc or
  --                                   iserver documentation for further details
  -- Notes:    No path environment variable is used; what is in name is all
  --           that is considered.

PROC so.remove (CHAN OF SP fs, ts, VAL []BYTE name, BYTE result)

  -- As sp.remove

PROC sp.rename (CHAN OF SP fs, ts, VAL []BYTE oldname, newname, BYTE result)

  -- Purpose:  To rename the specified file.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       oldname - the present name of the file to be renamed
  -- In:       newname - the desired name of the file to be renamed
  -- Out:      result - equals spr.ok if the renaming was successful;
  --           otherwise takes on a value indicating what went wrong:
  --             spr.bad.packet.size   the combined lengths of oldname and
  --                                   newname is too large - it must be
  --                                   less than sp.max.renamename.size
  --             spr.bad.name          either or both of the filenames are null
  --           >=spr.operation.failed  the renaming failed - see hostio.inc or
  --                                   iserver documentation for further details

PROC so.rename (CHAN OF SP fs, ts, VAL []BYTE oldname, newname, BYTE result)

  -- As sp.rename

PROC sp.getkey (CHAN OF SP fs, ts, BYTE key, result)

  -- Purpose:  To read a single character from the keyboard, waiting for one
  --           if there is none.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      key - the key that was read
  -- Out:      result - equals spr.ok if the read was successful; otherwise
  --           >= spr.operation.failed - see hostio.inc or iserver
  --           documentation for further details
  -- Notes:    The key is not echoed to the screen.
```

```
PROC so.getkey (CHAN OF SP fs, ts, BYTE key, result)

  -- As sp.getkey

PROC sp.pollkey (CHAN OF SP fs, ts, BYTE key, result)

  -- Purpose:  To read a single character from the keyboard, without waiting
  --           for one if there is none.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      key - the key that was read
  -- Out:      result - equals spr.ok if there was a key to read and the read
  --           was successful; otherwise >= spr.operation.failed - refer to
  --           hostio.inc or iserver documentation for further details
  -- Notes:    The key is not echoed to the screen.

PROC so.pollkey (CHAN OF SP fs, ts, BYTE key, result)

  -- As sp.pollkey

PROC sp.getenv (CHAN OF SP fs, ts, VAL []BYTE name,
                INT length, []BYTE value, BYTE result)

  -- Purpose:  To return the contents of an environment variable from
  --           whatever host is being used.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       name - a string containing the name of the desired
  --           environment variable.  name should not have more than
  --           sp.max.getenvname.size characters in it.
  -- Out:      length - if result is spr.ok, the number of bytes in value,
  --           starting at value[ 0 ]; (SIZE value) if result is
  --           spr.buffer.overflow; zero otherwise.
  -- Out:      value - the contents of the environment variable, if
  --           result is spr.ok; truncated environment variable if result
  --           is spr.buffer.overflow; undefined otherwise.
  -- Out:      result - spr.ok if the environment string successfully returned;
  --           some other value otherwise, indicating the sort of problem:
  --             spr.bad.name          the specified name is a null string
  --             spr.bad.packet.size   (SIZE name) > sp.max.getenvname.size
  --             spr.buffer.overflow   environment string too large for value;
  --                                   it is truncated to fit
  --           >=spr.operation.failed  could not read environment string - see
  --                                   hostio.inc or iserver documentation for
  --                                   further details
  -- Notes:    The largest size the returned contents of an environment
  --           variable can be is limited by the functionality of the server,
  --           not by this routine.

PROC so.getenv (CHAN OF SP fs, ts, VAL []BYTE name,
                INT length, []BYTE value, BYTE result)
```

```
  -- As sp.getenv

PROC sp.time (CHAN OF SP fs, ts, INT32 localtime, UTCtime, BYTE result)

  -- Purpose:  To retrieve the local and UTC time from the host system.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      localtime - the local time
  -- Out:      UTCtime - the Coordinated Universal Time, if available; zero
  --           if not available
  -- Out:      result - spr.ok if operation successful; otherwise
  --           >= spr.operation.failed - refer to hostio.inc or iserver
  --           documentation for details
  -- Notes:    Both times are expressed as the number of seconds that have
  --           elapsed since midnight on the 1st of January, 1970, and are
  --           given as unsigned INT32s
  --           UTC time used to be known as Greenwich Mean Time ( GMT )

PROC so.time (CHAN OF SP fs, ts, INT32 localtime, UTCtime)

  -- Purpose:  To retrieve the local and UTC time from the host system.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      localtime - the local time
  -- Out:      UTCtime - the Coordinated Universal Time, if available; zero
  --           if not available
  -- Notes:    Both times are expressed as the number of seconds that have
  --           elapsed since midnight on the 1st of January, 1970, and are
  --           given as unsigned INT32s
  --           UTC time used to be known as Greenwich Mean Time ( GMT )

PROC sp.system (CHAN OF SP fs, ts, VAL []BYTE command,
                INT32 status, BYTE result)

  -- Purpose:  To execute a command on the host system.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       command - the command to be executed, which may be null
  -- Out:      status - if command is not null and result is spr.ok then this
  --           is the return value of the command, which is host dependent
  -- Out:      result - equal to spr.ok if a host command processor exists;
  --           otherwise:
  --             spr.bad.packet.size   command too large,
  --                                   (SIZE command) > sp.max.systemcommand.size
  --           >=spr.operation.failed  the operation failed - see
  --                                   hostio.inc or iserver documentation
  --                                   for further details

PROC so.system (CHAN OF SP fs, ts, VAL []BYTE command,
                INT32 status, BYTE result)

  -- As sp.system
```

```
PROC sp.exit (CHAN OF SP fs, ts, VAL INT32 status, BYTE result)

  -- Purpose:  To terminate the server.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       status - a value which the server passes directly on to the
  --           host environment, except in the following two cases:
  --             sps.success - a host specific success result is passed on
  --             sps.failure - a host specific failure result is passed on
  -- Out:      result - equals spr.ok if the operation succeeded; otherwise
  --           >= spr.operation.failed - see hostio.inc or iserver documentation
  --           for further details


PROC so.exit (CHAN OF SP fs, ts, VAL INT32 status)

  -- Purpose:  To terminate the server.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       status - a value which the server passes directly on to the
  --           host environment, except in the following two cases:
  --             sps.success - a host specific success result is passed on
  --             sps.failure - a host specific failure result is passed on


PROC sp.commandline (CHAN OF SP fs, ts, VAL BYTE all,
                     INT length, []BYTE string, BYTE result)

  -- Purpose:  To return the command line passed to the server on invocation.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       all - either
  --             sp.short.commandline, remove the server's own name, its
  --                                   recognised options and their parameters
  --           or
  --             sp.whole.commandline, the entire command line
  -- Out:      length - the number of bytes returned in string, starting
  --           from string[ 0 ], being zero in the case of
  --           result >= spr.operation.failed
  -- Out:      string - contains the requested command line if result = spr.ok;
  --           contains a truncation of the requested command line if
  --           result = spr.buffer.overflow; is undefined otherwise.  There
  --           is no restriction on the size of string.
  -- Out:      result - equal to spr.ok if the command line was successfully
  --           returned; otherwise takes on a value indicating what went wrong:
  --             spr.buffer.overflow   string is not long enough to contain
  --                                   the command line, but the latter has
  --                                   been truncated to fit
  --           >=spr.operation.failed  the operation failed - refer to
  --                                   hostio.inc or iserver documentation
  --                                   for further details
```

```
PROC so.commandline (CHAN OF SP fs, ts, VAL BYTE all,
                     INT length, []BYTE string, BYTE result)

  -- As sp.commandline


PROC sp.core (CHAN OF SP fs, ts, VAL INT32 offset,
              INT bytes.read, []BYTE data, BYTE result)

  -- Purpose:  To return the contents of the root transputer's memory as
  --           peeked from the transputer when the server was invoked with
  --           the analyse option.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       offset - this gives the offset from base of memory
  --           ( (MOSTNEG INT) ) of the memory segment to be read ( so offset
  --           should be non-negative ); result will be >= spr.operation.failed
  --           if offset is larger than the amount of memory that was peeked
  -- Out:      bytes.read - the number of bytes read, which will be
  --           [data FROM 0 FOR bytes.read], if result = spr.ok; otherwise
  --           will be zero
  -- Out:      data - contains the contents of the memory read; (SIZE data)
  --           is the amount of memory in bytes that is requested to be read
  -- Out:      result - equal to spr.ok if the peeked memory was successfully
  --           returned; otherwise takes on a value indicating what the problem
  --           was:
  --             spr.bad.packet.size   data is too large,
  --                                     (SIZE data) > sp.max.corerequest.size
  --           >=spr.operation.failed  the operation failed or the transputer
  --                                   was not analysed when the server was
  --                                   invoked or offset is greater than the
  --                                   amount of memory peeked - refer to
  --                                   hostio.inc or iserver documentation for
  --                                   further details
  -- Notes:    If offset + (SIZE data) is larger than the total memory peeked
  --           then only those bytes from offset up to the end of that memory
  --           peeked are returned in data.
  --           If both offset and SIZE data are zero, the routine fails if
  --           the memory was not peeked, succeeds otherwise.

PROC so.core (CHAN OF SP fs, ts, VAL INT32 offset,
              INT bytes.read, []BYTE data, BYTE result)

  -- As sp.core


PROC sp.version (CHAN OF SP fs, ts,
                 BYTE version, host, os, board, result)

  -- Purpose:  To return identification information about the server and the
  --           host that it is running on.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      version - on division by 10 this gives the version of the server
```

```
  -- Out:      host - identifies the host via sph. constants
  -- Out:      os - identifies the host environment via spo. constants
  -- Out:      board - identifies the interface board via spb. constants
  -- Out:      result - spr.ok if the request was successful; otherwise
  --           >= spr.operation.failed - refer to hostio.inc or iserver
  --           documentation for further details
  -- Notes:    If any of the information ( except result ) has the value zero
  --           then that information is not available.
  --           The parameter version cannot distinguish between e.g. 1.41 and
  --           1.42.

PROC so.version (CHAN OF SP fs, ts,
                 BYTE version, host, os, board)

  -- Purpose:  To return identification information about the server and the
  --           host that it is running on.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      version - on division by 10 this gives the version of the server
  -- Out:      host - identifies the host via sph. constants
  -- Out:      os - identifies the host environment via spo. constants
  -- Out:      board - identifies the interface board via spb. constants
  -- Notes:    If any of the information has the value zero then that
  --           information is not available.
  --           The parameter version cannot distinguish between e.g. 1.41 and
  --           1.42.

PROC sp.buffer (CHAN OF SP fs, ts, from.user, to.user, CHAN OF BOOL stopper)

  -- Purpose:  To act as a communication buffer.
  -- Channels: fs - input to this routine
  --           ts - output from this routine
  --           from.user - input to this routine
  --           to.user - output from this routine
  --           stopper - input to this routine; TRUE or FALSE received will
  --           terminate this routine
  -- Notes:    No more than sp.max.packet.data.size bytes can be buffered.
  --           Primary input to this routine must be on the guard input channels
  --           from.user or stopper.
  --           Input on from.user is stored, output on ts, then a reply is
  --           expected on fs; this reply is stored and then output on to.user,
  --           and all this must finish before any new input can be received on
  --           from.user or stopper.

PROC so.buffer (CHAN OF SP fs, ts, from.user, to.user, CHAN OF BOOL stopper)

  -- As sp.buffer

PROC sp.multiplexor (CHAN OF SP fs, ts,
                     []CHAN OF SP from.user, to.user,
                     CHAN OF BOOL stopper)
```

```
  -- Purpose:  To multiplex any number of pairs of SP protocol channels
  --           onto a single pair of SP protocol channels.
  -- Channels: single pair:
  --             fs - input to this routine
  --             ts - output from this routine
  --           multiple pairs:
  --             from.user - input to this routine
  --             to.user - output from this routine
  --           termination channel:
  --             stopper - input to this routine: TRUE or FALSE received
  --             on this channel will cause the routine to terminate; this
  --             channel has the highest priority amongst the channels
  -- Notes:    It is an error if the number of channels in from.user is not
  --           equal to that in to.user.
  --           It is permissible for from.user ( and hence to.user ) to be
  --           null arrays.
  --           No more than sp.max.packet.data.size bytes can be buffered as
  --           they passes through this routine.
  --           Primary input to this routine must be on the guard input channels
  --           from.user[ i ] or stopper.
  --           Input on from.user[ i ] is stored, output on ts, then a reply is
  --           expected on fs; this reply is stored and then output on
  --           to.user[ i ], and all this must finish before any new input can
  --           be received on any of the from.user channels or stopper.
  --           To attempt some degree of fairness, there is a heirarchy of
  --           priorities from from.user with index i for
  --           SIZE from.user, using modulo SIZE from.user on the indexes,
  --           with i starting from 0 and incrementing by one after each input
  --           accepted.

PROC so.multiplexor (CHAN OF SP fs, ts,
                     []CHAN OF SP from.user, to.user,
                     CHAN OF BOOL stopper)

  -- As sp.multiplexor


PROC sp.pri.multiplexor (CHAN OF SP fs, ts,
                         []CHAN OF SP from.user, to.user,
                         CHAN OF BOOL stopper)

  -- Purpose:  To multiplex any number of pairs of SP protocol channels
  --           onto a single pair of SP protocol channels.
  -- Channels: single pair:
  --             fs - input to this routine
  --             ts - output from this routine
  --           multiple pairs:
  --             from.user - input to this routine
  --             to.user - output from this routine
  --             There is a relative priority to the input channels:
  --             from.user[ i ] is of higher priority than from.user[ j ]
  --             where i < j.
  --           termination channel:
  --             stopper - input to this routine: TRUE or FALSE received
  --             on this channel will cause the routine to terminate; this
```

```
  --             channel has the lowest priority of all input channels.
  -- Notes:    It is an error if the number of channels in from.user is
  --           different to that in to.user.
  --           It is permissible for the number of channels in from.user ( and
  --           hence also to.user ) to be zero.
  --           No more than sp.max.packet.data.size bytes can be buffered as
  --           they passes through this routine.
  --           Primary input to this routine must be on the guard input channels
  --           from.user[ i ] or stopper.
  --           Input on from.user[ i ] is stored, output on ts, then a reply is
  --           expected on fs; this reply is stored and then output on
  --           to.user[ i ], and all this must finish before any new input can
  --           be received on any of the from.user channels or stopper.

PROC so.pri.multiplexor (CHAN OF SP fs, ts,
                         []CHAN OF SP from.user, to.user,
                         CHAN OF BOOL stopper)

  -- As sp.pri.multiplexor

PROC sp.overlapped.buffer (CHAN OF SP fs, ts, from.user, to.user,
                           CHAN OF BOOL stopper)

  -- Purpose:  To act as two almost independent buffers.
  -- Channels: fs - input to this routine and to buffer2
  --           ts - output from this routine and from buffer1
  --           from.user - input to this routine and to buffer1
  --           to.user - output from this routine and from buffer2
  --           stopper - input to this routine; TRUE or FALSE received
  --           will cause buffer1 to stop listening on the ts channel and
  --           to terminate when buffer2 has passed through the same number
  --           of reply communications on the fs/to.user channels as
  --           buffer1 sent on the from.user/ts channels.  buffer1 terminates
  --           immediately after buffer1; the whole routine then terminates.
  -- Notes:    The dependence of the buffers is given by the fact that the
  --           number of communications through one buffer ( buffer2 ) must
  --           match the number through the other buffer ( buffer1 ) before
  --           the routine will terminate.
  --           No more than sp.max.packet.data.size bytes can be buffered in
  --           each of the buffers.

PROC so.overlapped.buffer (CHAN OF SP fs, ts, from.user, to.user,
                           CHAN OF BOOL stopper)

  -- As sp.overlapped.buffer

PROC sp.overlapped.multiplexor (CHAN OF SP fs, ts,
                                []CHAN OF SP from.user, to.user,
                                CHAN OF BOOL stopper,
                                []INT queue)

  -- Purpose:  To multiplex any number of pairs of SP protocol channels
```

```
  --           onto a single pair of SP protocol channels, where the
  --           multiplexing is overlapped so that output through this
  --           routine can continue independently of the receipt of
  --           replies.
  -- Channels: single pair:
  --             fs - input to this routine
  --             ts - output from this routine
  --           multiple pairs:
  --             from.user - input to this routine
  --             to.user - output from this routine
  --           termination channel:
  --             stopper - input to this routine: TRUE or FALSE received
  --             on this channel will cause the routine to stop listening
  --             on the from.user channels and wait until all replies have
  --             been received before terminating; stopper has the highest
  --             priority amongst the channels
  -- Out:      queue - the values in queue on entry are irrelevant but the
  --           size of it is the maximum number of communications that can
  --           be output through this routine before a reply is received and
  --           to be passed back; the values in queue on termination are
  --           equally irrelevant; queue is used for storage only; if it is
  --           of zero length then no communication can be done and the
  --           routine simply waits on stopper for an input which immediately
  --           causes termination of this routine
  -- Notes:    It is an error if the size of from.user is different to that
  --           of to.user.
  --           It is permissible for the size of from.user ( and hence
  --           to.user ) to be zero.
  --           The maximum number of bytes that can be channelled through
  --           this routine in each direction is sp.max.packet.data.size.
  --           It is assumed that replies for messages are received in the
  --           same sequence as the messages are sent.
  --           To attempt some degree of fairness, there is a heirarchy of
  --           priorities from from.user with index i for
  --           SIZE from.user, using modulo SIZE from.user on the indexes,
  --           with i starting from 0 and incrementing by one after each input
  --           accepted.

PROC so.overlapped.multiplexor (CHAN OF SP fs, ts,
                                []CHAN OF SP from.user, to.user,
                                CHAN OF BOOL stopper,
                                []INT queue)

  -- As sp.overlapped.multiplexor

PROC sp.overlapped.pri.multiplexor (CHAN OF SP fs, ts,
                                    []CHAN OF SP from.user, to.user,
                                    CHAN OF BOOL stopper,
                                    []INT queue)

  -- Purpose:  To multiplex any number of pairs of SP protocol channels
  --           onto a single pair of SP protocol channels, where the
  --           multiplexing is overlapped so that output through this
  --           routine can continue independently of the receipt of
```

```
  --           replies.
  -- Channels: single pair:
  --             fs - input to this routine
  --             ts - output from this routine
  --           multiple pairs:
  --             from.user - input to this routine
  --             to.user - output from this routine
  --             There is a relative priority to the input channels:
  --             from.user[ i ] is of higher priority than from.user[ j ]
  --             where i < j.
  --           termination channel:
  --             stopper - input to this routine: TRUE or FALSE received
  --             on this channel will cause the routine to stop listening
  --             on the from.user channels and wait until all replies have
  --             been received before terminating; this channel has the
  --             highest priority of all input channels
  -- Out:      queue - the values in queue on entry are irrelevant but the
  --           size of it is the maximum number of communications that can
  --           be output through this routine before a reply is received and
  --           to be passed back; the values in queue on termination are
  --           equally irrelevant; queue is used for storage only; if it is
  --           of zero length then no communication can be done and the
  --           routine simply waits on stopper for an input which immediately
  --           causes termination of this routine
  -- Notes:    It is an error if the size of from.user is different to that
  --           of to.user.
  --           It is permissible for the size of from.user ( and hence
  --           to.user ) to be zero.
  --           The maximum number of bytes that can be channelled through
  --           this routine in each direction is sp.max.packet.data.size.
  --           It is assumed that replies for messages are received in the
  --           same sequence as the messages are sent.

PROC so.overlapped.pri.multiplexor (CHAN OF SP fs, ts,
                                    []CHAN OF SP from.user, to.user,
                                    CHAN OF BOOL stopper,
                                    []INT queue)

  -- As sp.overlapped.pri.multiplexor

PROC so.open.temp (CHAN OF SP fs, ts, VAL BYTE type,
                   [so.temp.filename.length]BYTE filename,
                   INT32 streamid, BYTE result)

  -- Purpose:  To open a temporary file in spm.new.update mode.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       type - either spt.binary or spt.text
  -- Out:      filename - the name of the file opened
  -- Out:      streamid - if result is spr.ok, the identification number of
  --           the stream associated with the file that was opened; undefined
  --           otherwise
  -- Out:      result - spr.ok if all went well; otherwise takes on a
  --           value indicating what went wrong:
```

```
  --             spr.bad.type          invalid type
  --             spr.notok             there are already 10,000 such temp files
  --           >=spr.operation.failed  the open failed - refer to hostio.inc
  --                                   or iserver documentation for further
  --                                   details
  -- Notes:    The file will be opened in the current directory.
  --           The file name of the temporary file is chosen as follows:
  --           the first name tried is TEMP00; if there already exists such
  --           a file then TEMP01 is tried; if there already exists such
  --           a file then TEMP02 is tried; etc. up to TEMP99 when, if such
  --           a file exists then TEM100 is tried, and so on up to TEM999, at
  --           which point, if such a file exists, TE1000 is tried, and so
  --           on up to TE9999, which is the last name tried, i.e. a maximum
  --           of 10,000 possible temporary files can be created by this routine.

PROC so.test.exists (CHAN OF SP fs, ts, VAL []BYTE filename, BOOL exists)

  -- Purpose:  To test for the existence of a file.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       filename - the name of the file to test for the existence of.
  --           The name of the file must fit exactly into filename, i.e. there
  --           are (SIZE filename) characters in the name of the file.  A
  --           directory specification may form part of filename.
  -- Out:      exists - TRUE if the file exists; FALSE otherwise

PROC so.popen.read (CHAN OF SP fs, ts,
                    VAL []BYTE filename, path.variable.name,
                    VAL BYTE open.type,
                    INT full.len, []BYTE full.name,
                    INT32 stream.id, BYTE result)

  -- Purpose:  To perform a general purpose open file for reading.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       filename - a string with the name of the file to be opened.
  --           There may be an explicit directory part to the file's name.
  -- In:       path.variable.name - a string holding the name of a path
  --           environment variable; there is a limit imposed on the number of
  --           characters accepted from sp.getenv for the environment variable
  --           contents: the limit is imposed by this routine and is given
  --           by 256 ( path.string.size in the program text ).
  -- In:       open.type - either
  --             spt.binary, for a file containing raw bytes only
  --           or
  --             spt.text, for a file stored as text records separated by newlines
  -- Out:      full.len - the number of characters in full.name if result
  --           is spr.ok; undefined otherwise
  -- Out:      full.name - if result is spr.ok, [full.name FROM 0 FOR full.len]
  --           is the name of the file that was successfully opened; undefined
  --           otherwise.  full.name clearly should be at least as large as
  --           filename.
  -- Out:      streamid - if result is spr.ok, the identification number of the
  --           stream associated with the file that was opened; undefined
```

```
  --            otherwise
  -- Out:       result - spr.ok if a file was opened successfully; otherwise
  --            some value indicating what went wrong:
  --              spr.bad.packet.size     (SIZE filename) > sp.max.openname.size
  --                                                             or
  --                                      (SIZE path.variable.name) >
  --                                                       sp.max.getenvname.size
  --              spr.bad.name            null file name supplied
  --              spr.bad.type            invalid file type specified
  --              spr.full.name.too.short (SIZE full.name) not large enough to
  --                                      hold name of a file to open
  --              spr.buffer.overflow     length of environment string > 256
  --            >=spr.operation.failed    the open failed or could not read
  --                                      environment string - see hostio.inc or
  --                                      iserver documentation for more details
  -- Notes:     This procedure performs the following actions:
  --            1. An attempt is made to open the file as given by filename
  --               ( so if there is no directory part to filename the
  --               current directory will be tried );
  --            2. If the attempt in 1 is not successful and there is no
  --               explicit directory part to filename, the path is
  --               searched for the file.


PROC so.parse.command.line (CHAN OF SP fs, ts,
                            VAL []BYTE option.strings,
                            VAL []INT    option.parameters.required,
                            []BOOL       option.exists,
                            [][2]INT     option.parameters,
                            INT error.len, []BYTE line)

  -- Purpose:  To read the server command line and parse it for specified
  --           options and associated parameters.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       option.strings - a list of all possible options.  These options
  --           must not be longer than 256 bytes and any letters should be in
  --           upper case, even though the options as given by the user on the
  --           command line are case insensitive.  To read a parameter that
  --           has no preceding option ( such as a filename ) then the first
  --           option string should contain only spaces.  Because all of
  --           these option strings must be the same length, trailing
  --           spaces should be used to pad.
  -- In:       option.parameters.required - indicates if the corresponding
  --           option ( in option.strings ) requires a parameter.  The
  --           permissible values are:
  --             spopt.never     never takes a parameter
  --             spopt.maybe     optionally takes a parameter
  --             spopt.always    must take a parameter
  -- Out:      option.exists - TRUE if the corresponding option was present on
  --           the command line; otherwise FALSE
  -- Out:      option.parameters - if an option was followed by a parameter
  --           then the position in the array line where the parameter starts
  --           and the length of the parameter are given by the first and
  --           second elements respectively in the corresponding element of
  --           this parameter, as long as an error did not occur
```

```
  -- Out:      error.len - zero if no error occurs whilst the command line is
  --           being parsed; otherwise greater than zero and is the number of
  --           bytes in line ( which will contain an error message )
  -- Out:      line - contains the command line as supplied by the server if
  --           there was no error ( in which case error.len is zero );
  --           otherwise contains an error message ( the length of which is
  --           given by error.len ).  There is no limitation on the size of
  --           line - if it is not long enough for an error message then that
  --           message is truncated to fit; if not long enough for the
  --           command line, then an error message is put into line.

PROC so.write.string (CHAN OF SP fs, ts, VAL []BYTE string)

  -- Purpose:  To write a string to standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       string - the string to be written, which can be any length

PROC so.fwrite.string (CHAN OF SP fs, ts, VAL INT32 streamid,
                       VAL []BYTE string, BYTE result)

  -- Purpose:  To write a string to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to write
  --           the string to
  -- In:       string - the string to be written, which can be any length
  -- Out:      result - spr.ok if the string was written; spr.notok if not
  --           all of the string was written

PROC so.write.char (CHAN OF SP fs, ts, VAL BYTE char)

  -- Purpose:  To write a single character to standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       char - the single character to write

PROC so.fwrite.char (CHAN OF SP fs, ts, VAL INT32 streamid, VAL BYTE char,
                     BYTE result)

  -- Purpose:  To write a single character to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to write to
  -- In:       char - the single character to write
  -- Out:      result - spr.ok if the character was written; otherwise spr.notok

PROC so.write.string.nl (CHAN OF SP fs, ts, VAL []BYTE string)

  -- Purpose:  To write a string appended by a newline to standard out.
  -- Channels: fs - from server
```

```
  --           ts - to server
  -- In:       string - the string to be written, which can be any length
  -- Notes:    The newline sequence is appended by this routine.

PROC so.fwrite.string.nl (CHAN OF SP fs, ts, VAL INT32 streamid,
                          VAL []BYTE string, BYTE result)

  -- Purpose:  To write a string appended by a newline to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to write
  --           the string to
  -- In:       string - the string to be written, which can be any length
  -- Out:      result - spr.ok if the string was successfully written;
  --           spr.notok if not all of the string was written;
  --           if >= spr.operation.failed refer to hostio.inc or iserver
  --           documentation for further details
  -- Notes:    The newline sequence is appended by this routine.

PROC so.write.nl (CHAN OF SP fs, ts)

  -- Purpose:  To write a newline to standard out.
  -- Channels: fs - from server
  --           ts - to server

PROC so.fwrite.nl (CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result)

  -- Purpose:  To write a newline to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to write to
  -- Out:      result - spr.ok if the newline was successfully written; otherwise
  --           >= spr.operation.failed - refer to hostio.inc or iserver
  --           documentation for further details

PROC so.fwrite.int (CHAN OF SP fs, ts, VAL INT32 streamid, VAL INT n, width,
                    BYTE result)

  -- Purpose:  To write to the specified file stream an integer as decimal
  --           ASCII digits, padded out with leading spaces and an optional
  --           sign to the specified field width.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to which
  --           the string representing the integer is to be sent
  -- In:       n - the integer that is desired to be written
  -- In:       width - the desired field width of the string
  -- Out:      result - equal to spr.ok if the string was written all right;
  --           otherwise not all of the string could be written, in which
  --           case result takes on a value of spr.notok
  -- Notes:    If the field width is too small for the number, then it is
  --           widened as necessary; a zero value for the field width will
```

```
  --           give minimum width; a negative value is an error.

PROC so.write.int (CHAN OF SP fs, ts, VAL INT n, width)

  -- Purpose:  To write to standard out an integer as decimal
  --           ASCII digits, padded out with leading spaces and an
  --           optional sign to the specified field width.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       n - the integer that is desired to be written
  -- In:       width - the desired field width of the string
  -- Notes:    If the field width is too small for the number, then it is
  --           widened as necessary; a zero value for the field width will
  --           give minimum width; a negative field width is an error.

PROC so.fwrite.int32 (CHAN OF SP fs, ts, VAL INT32 streamid, n, VAL INT width,
                      BYTE result)

  -- As so.fwrite.int, but for INT32s

PROC so.write.int32 (CHAN OF SP fs, ts, VAL INT32 n, VAL INT width)

  -- As so.write.int, but for INT32s

PROC so.fwrite.int64 (CHAN OF SP fs, ts, VAL INT32 streamid,
                      VAL INT64 n, VAL INT width, BYTE result)

  -- As so.fwrite.int, but for INT64s

PROC so.write.int64 (CHAN OF SP fs, ts, VAL INT64 n, VAL INT width)

  -- As so.write.int, but for INT64s

PROC so.fwrite.hex.int (CHAN OF SP fs, ts, VAL INT32 streamid,
                        VAL INT n, width, BYTE result)

  -- Purpose:  To write an integer as hexadecimal ASCII characters
  --           preceded by the '#' character to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to write to
  -- In:       n - the integer to write out
  -- In:       width - the field width to use when writing the hex;
  --           if width is larger than the size of the number then the
  --           number is padded with leading 0's or F's as appropriate;
  --           if width is smaller than the size of the number then the
  --           number is truncated at the left to width characters;
  --           width does not take the '#' into account;
  --           a negative field width is an error
  -- Out:      result - spr.ok if the integer was successfully written;
```

```
  --           otherwise not all of the string could be written, in which case
  --           result takes on a value of spr.notok

PROC so.write.hex.int (CHAN OF SP fs, ts, VAL INT n, width)

  -- Purpose:  To write an integer as hexadecimal ASCII characters
  --           preceded by the '#' character to standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       n - the integer to write out
  -- In:       width - the field width to use when writing the hex;
  --           if width is larger than the size of the number then the
  --           number is padded with leading 0's or F's as appropriate;
  --           if width is smaller than the size of the number then the
  --           number is truncated at the left to width characters;
  --           width does not take the '#' into account;
  --           a negative field width is an error

PROC so.fwrite.hex.int32 (CHAN OF SP fs, ts, VAL INT32 streamid, n,
                          VAL INT width, BYTE result)

  -- As so.fwrite.hex.int, but for INT32s

PROC so.write.hex.int32 (CHAN OF SP fs, ts, VAL INT32 n, VAL INT width)

  -- As so.write.hex.int, but for INT32s

PROC so.fwrite.hex.int64 (CHAN OF SP fs, ts, VAL INT32 streamid,
                          VAL INT64 n, VAL INT width, BYTE result)

  -- As so.fwrite.hex.int, but for INT64s

PROC so.write.hex.int64 (CHAN OF SP fs, ts, VAL INT64 n, VAL INT width)

  -- As so.write.hex.int, but for INT64s

PROC so.fwrite.real32 (CHAN OF SP fs, ts, VAL INT32 streamid,
                       VAL REAL32 r, VAL INT Ip, Dp, BYTE result)

  -- Purpose:  To write a single precision real as decimal ASCII characters
  --           to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to write
  --           the real to
  -- In:       r - a single precision real
  -- In, In:   Ip, Dp - formatting values; see REAL32TOSTRING for their
  --           effect
  -- Out:      result - spr.ok if the string representing the real was written;
  --           otherwise not all of the characters were written in which case
```

```
  --           spr.notok
  -- Notes:    Behaves as an invalid process if the string to write is longer
  --           than 24 characters.

PROC so.write.real32 (CHAN OF SP fs, ts, VAL REAL32 r, VAL INT Ip, Dp)

  -- Purpose:  To write a single precision real as decimal ASCII characters
  --           to standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       r - a single precision real
  -- In, In:   Ip, Dp - formatting values; see REAL32TOSTRING for their
  --           effect
  -- Notes:    Behaves as an invalid process if the string to write is longer
  --           than 24 characters.

PROC so.fwrite.real64 (CHAN OF SP fs, ts, VAL INT32 streamid,
                       VAL REAL64 r, VAL INT Ip, Dp, BYTE result)

  -- Purpose:  To write a double precision real as decimal ASCII characters
  --           to a stream.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       streamid - the identification number of the stream to write
  --           the real to
  -- In:       r - a double precision real
  -- In, In:   Ip, Dp - formatting values; see REAL64TOSTRING for their
  --           effect
  -- Out:      result - spr.ok if the string representing the real was written;
  --           otherwise not all of the characters were written in which case
  --           spr.notok
  -- Notes:    Behaves as an invalid process if the string to write is longer
  --           than 30 characters.

PROC so.write.real64 (CHAN OF SP fs, ts, VAL REAL64 r, VAL INT Ip, Dp)

  -- Purpose:  To write a double precision real as decimal ASCII characters
  --           to standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       r - a double precision real
  -- In, In:   Ip, Dp - formatting values; see REAL64TOSTRING for their
  --           effect
  -- Notes:    Behaves as an invalid process if the string to write is longer
  --           than 30 characters.

PROC so.read.echo.any.int (CHAN OF SP fs, ts, INT n, BOOL error)

  -- Purpose:  To read an integer in either ( two's complement ) hexadecimal
  --           form or in decimal form from the keyboard, and to echo it
  --           to standard out.
  -- Channels: fs - from server
```

```
  --            ts - to server
  -- Out:       n - if error is FALSE, the value of the integer read;
  --            otherwise undefined
  -- Out:       error - FALSE if everything was in order; TRUE otherwise:
  --            1)  if the integer overflows the INT range;
  --            2a) if it is a hex integer, if there is a non-hex character
  --                in the string read from the keyboard ( other than #, $,
  --                or % in the first character position )
  --            2b) if it is a decimal integer, if there is a non-numeric
  --                character in the string read from the keyboard ( other
  --                than a + or - in the first character position )
  -- Notes:     The hexadecimal number must begin with one of '#', '$' or '%';
  --            a sign is not allowed ( nor necessary since the most significant
  --            bit indicates the sign ).
  --            A '#' and '$' mean exactly the same thing: the following
  --            digits form a hex integer.
  --            A '%' means add the given hex integer to MOSTNEG INT using
  --            modulo arithmetic to give the result.
  --            Both upper and lower case hex digits are permissible.
  --            A sign is allowed in the first character position if it is a
  --            decimal integer that is entered.
  --            The number typed at the keyboard must be terminated by pressing
  --            'RETURN'.

PROC so.read.echo.int (CHAN OF SP fs, ts, INT n, BOOL error)

  -- Purpose:  To read a string containing a representation of a decimal
  --           integer from the keyboard, and to echo it to standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      n - if error is FALSE, the value of the integer read; otherwise
  --           undefined
  -- Out:      error - FALSE if nothing went wrong in the operation; otherwise
  --           TRUE:
  --           1) if the integer read overflows the INT range
  --           2) if a non-numeric character is found in the input string ( other
  --           than a + or - in the first character position )
  -- Notes:    The integer may commence with a + or a - sign.
  --           The number typed at the keyboard must be terminated by pressing
  --           'RETURN'.

PROC so.read.echo.int32 (CHAN OF SP fs, ts, INT32 n, BOOL error)

  -- As so.read.echo.int, but for INT32s

PROC so.read.echo.int64 (CHAN OF SP fs, ts, INT64 n, BOOL error)

  -- As so.read.echo.int, but for INT64s

PROC so.read.echo.hex.int (CHAN OF SP fs, ts, INT n, BOOL error)

  -- Purpose:  To read an integer in ( two's complement ) hexadecimal form
```

```
  --           from the keyboard, and to echo it to standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      n - if error is FALSE, the value of the hexadecimal number read;
  --           otherwise undefined
  -- Out:      error - FALSE if everything was in order; TRUE otherwise:
  --           1) if the integer overflows the INT range;
  --           2) if there is a non-hex character in the string read from
  --              the keyboard ( other than #, $, or % in the first character
  --              position )
  -- Notes:    The hexadecimal number must begin with one of '#', '$' or '%';
  --           a sign is not allowed ( nor necessary since the most significant
  --           bit indicates the sign ).
  --           A '#' and '$' mean exactly the same thing: the following
  --           digits form a hex integer.
  --           A '%' means add the given hex integer to MOSTNEG INT using
  --           modulo arithmetic to give the result.
  --           Both upper and lower case hex digits are permissible.
  --           The number typed at the keyboard must be terminated by pressing
  --           'RETURN'.

PROC so.read.echo.hex.int32 (CHAN OF SP fs, ts, INT32 n, BOOL error)

  -- As so.read.echo.hex.int, but for INT32s

PROC so.read.echo.hex.int64 (CHAN OF SP fs, ts, INT64 n, BOOL error)

  -- As so.read.echo.hex.int, but for INT64s

PROC so.read.echo.real32 (CHAN OF SP fs, ts, REAL32 n, BOOL error)

  -- Purpose:  To read a string containing a representation of a single
  --           precision real in occam syntax ( though a + or - is allowed
  --           as first character ) from the keyboard and to echo it to
  --           standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      n - if error is FALSE, the IEEE single precision format of the
  --           real that was read, otherwise undefined
  -- Out:      error - FALSE if the real was read and converted to binary
  --           without problem; TRUE otherwise - if the string read does not
  --           conform to the syntax of an occam real number, other than
  --           the fact that a plus or minus sign is allowed as the first
  --           character
  -- Notes:    The number typed at the keyboard must be terminated by pressing
  --           'RETURN'.

PROC so.read.echo.real64 (CHAN OF SP fs, ts, REAL64 n, BOOL error)

  -- Purpose:  To read a string containing a representation of a double
  --           precision real in occam syntax ( though a + or - is allowed
  --           as first character ) from the keyboard and to echo it to
```

```
  --           standard out.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      n - if error is FALSE, the IEEE double precision format of the
  --           real that was read, otherwise undefined
  -- Out:      error - FALSE if the real was read and converted to binary
  --           without problem; TRUE otherwise - if the string read does not
  --           conform to the syntax of an occam real number, other than
  --           the fact that a plus or minus sign is allowed as the first
  --           character
  -- Notes:    The number typed at the keyboard must be terminated by pressing
  --           'RETURN'.

PROC so.read.line (CHAN OF SP fs, ts, INT len, []BYTE line, BYTE result)

  -- Purpose:  To read a line from the keyboard without echoing it.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      len - the number of bytes in the parameter line that form
  --           the line that was read
  -- Out:      line - if result = spr.ok, the line that was read is in the
  --           first len bytes, the remaining bytes being undefined or if
  --           the supplied byte array was not large enough for the read
  --           line, the read line is truncated to fit; otherwise undefined
  -- Out:      result - spr.ok if the line was successfully read; otherwise
  --           >= spr.operation.failed - see hostio.inc or iserver
  --           documentation for further details
  -- Notes:    The line to be read is considered terminated by a carriage
  --           return, ie ASCII ^M, which has value #0D = 13.  This
  --           carriage return is not included in the line, and nor is a
  --           newline ( ASCII ^J, #0A = 10 ) if present in the input.
  --           If an error occurs whilst reading the line this routine
  --           terminates immediately.

PROC so.read.echo.line (CHAN OF SP fs, ts, INT len, []BYTE line, BYTE result)

  -- Purpose:  To read a line from the keyboard, echoing it to standard
  --           out.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      len - the number of bytes in line that form the line that
  --           was read
  -- Out:      line - if result = spr.ok, the line that was read is in the
  --           first len bytes, the remaining bytes being undefined or if
  --           the supplied byte array was not large enough for the read
  --           line, the read line is truncated to fit; otherwise undefined
  -- Out:      result - spr.ok if no problems encountered; otherwise
  --           >= spr.operation.failed - refer to hostio.inc or iserver
  --           documentation for further details
  -- Notes:    The line to be read is considered terminated by a carriage
  --           return, ie ASCII ^M, which has value #0D = 13.  This
  --           carriage return is not included in line, and nor is a
  --           newline ( ASCII ^J, #0A = 10 ) if present in the input.
  --           Carriage return and newline are not echoed.
```

```
  --           If an error occurs whilst reading the line this routine
  --           terminates immediately.

PROC so.ask (CHAN OF SP fs, ts, VAL []BYTE prompt, replies,
             VAL BOOL display.possible.replies, echo.reply,
             INT reply.number)

  -- Purpose:  To provide a general purpose routine for interactively asking
  --           a user of a program a question requiring only a one character
  --           answer.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       prompt - the prompt to be displayed on the screen to the user,
  --           but any list of possible replies and a question mark will be
  --           taken care of outside of this prompt
  -- In:       replies - the possible single character replies allowed, which
  --           do not necessarily need to be printable
  -- In:       display.possible.replies - if TRUE the printable replies are
  --           displayed ( in upper case ) inside round brackets with commas
  --           inbetween each character, this display being one space after the
  --           given prompt; if FALSE then this is not done
  -- In:       echo.reply - if TRUE then the reply typed at the keyboard
  --           by the user is echoed if it is printable, and it is echoed as an
  --           upper case letter if a letter; if FALSE the reply typed is not
  --           echoed
  -- Out:      reply.number - an integer corresponding to the reply typed, where
  --           0 is associated with the first possible reply, 1 with the second
  --           possible reply, etc., where the order is as in replies
  -- Notes:    After the prompt and possibly the printable replies, " ? " is
  --           output and a valid key is waited for from the keyboard; invalid
  --           keys are ignored.

PROC sp.send.packet (CHAN OF SP ts, VAL []BYTE packet, BOOL error)

  -- Purpose:  To send an SP protocol packet.
  -- Channel:  ts - the channel on which to send the packet
  -- In:       packet - the packet to send
  -- Out:      error - FALSE if the packet size is greater than or equal to
  --           sp.min.packet.data.size, and is less than or equal to
  --           sp.max.packet.data.size, and is even; TRUE otherwise, in which
  --           case the packet is not sent

PROC sp.receive.packet (CHAN OF SP fs, INT16 length, []BYTE packet, BOOL error)

  -- Purpose:  To receive an SP protocol packet.
  -- Channel:  fs - the channel on which to receive the packet
  -- Out:      length - the number of bytes in the received packet
  -- Out:      packet - the received packet, left justified
  -- Out:      error - TRUE if length is greater than sp.max.packet.data.size;
  --           FALSE otherwise

PROC so.time.to.date (VAL INT32 input.time, [so.date.len]INT date)
```

```
  -- Purpose:  To convert the number of seconds since the midnight before
  --           1st January 1970 into the associated time and date.
  -- In:       input.time - the number of seconds since the midnight before
  --           1st January 1970
  -- Out:      date - an array of six integers where the elements are,
  --                index                   data
  --                  0              seconds past the minute
  --                  1              minutes past the hour
  --                  2              the hour in the 24 hour clock
  --                  3              the day of the month
  --                  4              the month, where January is 1
  --                  5              the year expressed fully, i.e. 4 digits

PROC so.date.to.ascii (VAL [so.date.len]INT date, VAL BOOL long.years,
                       VAL BOOL days.first, [so.time.string.len]BYTE string)

  -- Purpose:  To format a time and date given in the form of six integers.
  -- In:       date - an array of six integers where the elements are,
  --                index                   data
  --                  0              seconds past the minute
  --                  1              minutes past the hour
  --                  2              the hour in the 24 hour clock
  --                  3              the day of the month
  --                  4              the month, where January is 1
  --                  5              the year expressed fully, i.e. 4 digits
  -- In:       long.years - TRUE if a four digit year is desired; FALSE if
  --           a two digit year is desired, the digits being the tens and
  --           units, followed by two spaces
  -- In:       days.first - TRUE if date to be written with the days preceding
  --           the month ( i.e. British format ); FALSE if the month is to
  --           precede the days ( i.e. U.S. format )
  -- Out:      string - an ASCII string representing the time and date as
  --           follows:
  --             "HH:MM:SS DD/MM/YY  " -- long.years = FALSE; days.first = TRUE
  --             "HH:MM:SS DD/MM/YYYY" -- long.years = TRUE; days.first = TRUE
  --             "HH:MM:SS MM/DD/YY  " -- long.years = FALSE; days.first = FALSE
  --             "HH:MM:SS MM/DD/YYYY" -- long.years = TRUE; days.first = FALSE
  --           where HH:MM:SS is hours, minutes and seconds,
  --           and DD/MM/YY is day, month and year

PROC so.time.to.ascii (VAL INT32 time, VAL BOOL long.years,
                       VAL BOOL days.first, [so.time.string.len]BYTE string)

  -- Purpose:  To convert the number of seconds since the midnight before
  --           1st January 1970 into a formatted time and date
  -- In:       time - the number of seconds since the midnight before
  --           1st January 1970
  -- In:       long.years - TRUE if a four digit year is desired; FALSE if
  --           a two digit year is desired, the digits being the tens and
  --           units, followed by two spaces
  -- In:       days.first - TRUE if date to be written with the days preceding
  --           the month ( i.e. British format ); FALSE if the month is to
  --           precede the days ( i.e. U.S. format )
  -- Out:      string - an ASCII string representing the time and date as
```

```
  --           follows:
  --             "HH:MM:SS DD/MM/YY  " -- long.years = FALSE; days.first = TRUE
  --             "HH:MM:SS DD/MM/YYYY" -- long.years = TRUE; days.first = TRUE
  --             "HH:MM:SS MM/DD/YY  " -- long.years = FALSE; days.first = FALSE
  --             "HH:MM:SS MM/DD/YYYY" -- long.years = TRUE; days.first = FALSE
  --           where HH:MM:SS is hours, minutes and seconds,
  --           and DD/MM/YY is day, month and year

PROC so.today.date (CHAN OF SP fs, ts, [so.date.len]INT date)

  -- Purpose:  To give the time and date, in the form of six integers, of
  --           when this routine was called.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      date - an array of six integers where the elements are,
  --                index                  data
  --                  0              seconds past the minute
  --                  1              minutes past the hour
  --                  2              the hour in the 24 hour clock
  --                  3              the day of the month
  --                  4              the month, where January is 1
  --                  5              the year expressed fully, i.e. 4 digits
  -- Notes:    If the time and date are unavailable all elements of date are
  --           set to zero.
  --           Local time is used.

PROC so.today.ascii (CHAN OF SP fs, ts, VAL BOOL long.years, days.first,
                     [so.time.string.len]BYTE string)

  -- Purpose:  To give the time and date, in formatted form, of
  --           when this routine was called.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       long.years - TRUE if a four digit year is desired; FALSE if
  --           a two digit year is desired, the digits being the tens and
  --           units, followed by two spaces
  -- In:       days.first - TRUE if date to be written with the days preceding
  --           the month ( i.e. British format ); FALSE if the month is to
  --           precede the days ( i.e. U.S. format )
  -- Out:      string - an ASCII string representing the time and date as
  --           follows:
  --             "HH:MM:SS DD/MM/YY  " -- long.years = FALSE; days.first = TRUE
  --             "HH:MM:SS DD/MM/YYYY" -- long.years = TRUE; days.first = TRUE
  --             "HH:MM:SS MM/DD/YY  " -- long.years = FALSE; days.first = FALSE
  --             "HH:MM:SS MM/DD/YYYY" -- long.years = TRUE; days.first = FALSE
  --           where HH:MM:SS is hours, minutes and seconds,
  --           and DD/MM/YY is day, month and year
  -- Notes:    If the time and date are unavailable string is filled with spaces.
  --           Local time is used.
```

9 snglmath.lib

15 entry points

Entry Point	Where Documented
ALOG	occam 2 Reference Manual, Appendix N
ALOG10	”
EXP	”
POWER	”
SIN	”
COS	”
TAN	”
ASIN	”
ACOS	”
ATAN	”
ATAN2	”
SINH	”
COSH	”
TANH	”
RAN	”

10 dblmath.lib

15 entry points

Entry Point	Where Documented
DALOG	occam 2 Reference Manual, Appendix N
DALOG10	”
DEXP	”
DPOWER	”
DSIN	”
DCOS	”
DTAN	”
DASIN	”
DACOS	”
DATAN	”
DATAN2	”
DSINH	”
DCOSH	”
DTANH	”
DRAN	”

11 tbmaths.lib

30 entry points

Entry Point	Where Documented

ALOG	occam 2 Reference Manual, Appendix N
ALOG10	”
EXP	”
POWER	”
SIN	”
COS	”
TAN	”
ASIN	”
ACOS	”
ATAN	”
ATAN2	”
SINH	”
COSH	”
TANH	”
RAN	”
DALOG	”
DALOG10	”
DEXP	”
DPOWER	”
DSIN	”
DCOS	”
DTAN	”
DASIN	”
DACOS	”
DATAN	”
DATAN2	”
DSINH	”
DCOSH	”
DTANH	”
DRAN	”

12 msdos.lib

6 entry points

Constants referred to in the following specifications are to be found in the include files **msdos.inc** and **hostio.inc**.

```
PROC dos.send.block (CHAN OF SP fs, ts, VAL INT32 location,
                     VAL []BYTE block, INT len, BYTE result)

  -- Purpose:  To write a block of data to host memory.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       location - the start memory address at which to write
  --           the block of data; this address is arranged as the segment
  --           in the top two bytes and the offset in the lower two bytes
  --           of this parameter, both unsigned
  -- In:       block - the data to be written, that is, (SIZE block) bytes
  -- Out:      len - the number of bytes written
  -- Out:      result - spr.ok if the write was successful, otherwise
  --           takes on a value indicating what went wrong:
  --             spr.bad.packet.size   too many bytes requested to be written:
```

```
  --                              (SIZE block) > dos.max.send.block.buffer.size
  --           >=spr.operation.failed  the write failed - see hostio.inc or
  --                                   the server documentation for further
  --                                   details

PROC dos.receive.block (CHAN OF SP fs, ts, VAL INT32 location,
                        INT bytes.read, []BYTE block, BYTE result)

  -- Purpose:  To read a block of data from host memory.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       location - the start address of where to read the data
  --           from; this address is arranged as the segment in the
  --           top two bytes and the offset in the lower two bytes
  --           of this parameter, both unsigned
  -- Out:      bytes.read - if result is spr.ok, the number of bytes read;
  --           otherwise zero
  -- Out:      block - if result is spr.ok, the data read is held in the
  --           first bytes.read bytes of block; otherwise undefined.  The
  --           number of bytes requested to be read is (SIZE block).
  -- Out:      result - spr.ok if the read was successful; otherwise
  --           takes on a value indicating what went wrong:
  --             spr.bad.packet.size   too many bytes were requested to be
  --                                   read:
  --                                (SIZE block) > dos.max.receive.block.buffer.size
  --           >=spr.operation.failed  the read failed, ( so bytes.read = 0 )
  --                                   - see hostio.inc or the server
  --                                   documentation for further details

PROC dos.call.interrupt (CHAN OF SP fs, ts, VAL INT16 interrupt,
                         VAL [dos.interrupt.regs.size]BYTE register.block.in,
                         BYTE carry.flag,
                         [dos.interrupt.regs.size]BYTE register.block.out,
                         BYTE result)

  -- Purpose:  To invoke an interrupt call on the host PC, with the
  --           processor's registers initialised to desired values.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       interrupt - the interrupt number
  -- In:       register.block.in - the register values to which to
  --           initialise the PC's registers on this interrupt
  -- Out:      carry.flag - if result is spr.ok, the value of the PC's
  --           carry flag on return from the interrupt; otherwise
  --           undefined
  -- Out:      register.block.out - if result is spr.ok, the values
  --           stored in the processor's registers on return from the
  --           interrupt; otherwise undefined
  -- Out:      result - spr.ok if the interrupt was successful;
  --           otherwise >= spr.operation.failed
  -- Notes:    Each register value occupies 4 bytes:
  --                register           start position in block
  --                                 ( least significant byte )
  --                   ax                          0
```

```
  --                  bx                          4
  --                  cx                          8
  --                  dx                         12
  --                  di                         16
  --                  si                         20
  --                  cs                         24
  --                  ds                         28
  --                  es                         32
  --                  ss                         36

PROC dos.read.regs (CHAN OF SP fs, ts,
                    [dos.read.regs.size]BYTE registers,
                    BYTE result)

  -- Purpose:  To read the current values of some of the PC's
  --           registers.
  -- Channels: fs - from server
  --           ts - to server
  -- Out:      registers - a block of bytes containing the values
  --           of the registers read
  -- Out:      result - spr.ok if the registers were read
  --           successfully; otherwise >= spr.operation.failed
  -- Notes:    Each register value occupies 4 bytes:
  --                register      start position in block
  --                            ( least significant byte )
  --                  ax                    0
  --                  bx                    4
  --                  cx                    8
  --                  dx                   12

PROC dos.port.write (CHAN OF SP fs, ts, VAL INT16 port.location,
                     VAL BYTE value, BYTE result)

  -- Purpose:  To write a given value to a given port.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       port.location - the address where the port is to be found,
  --           the address being in the I/O space of the PC and hence is
  --           an unsigned number between 0 and 64K
  -- In:       value - the value to write to port.location
  -- Out:      result - spr.ok if the write was successful; otherwise
  --           >= spr.operation.failed
  -- Notes:    No check is made to ensure that the value written to the
  --           port has been correctly read by the device connected to
  --           the port ( if any ).

PROC dos.port.read (CHAN OF SP fs, ts, VAL INT16 port.location,
                    BYTE value, BYTE result)

  -- Purpose:  To read a byte from a given port.
  -- Channels: fs - from server
  --           ts - to server
  -- In:       port.location - the address where the port is to be found,
```

```
  --           the address being in the I/O space of the PC and hence is
  --           an unsigned number between 0 and 64K
  -- Out:      value - the value read from port.location
  -- Out:      result - spr.ok if the read was successful; otherwise
  --           >= spr.operation.failed
  -- Notes:    No check is made to ensure that the value received from
  --           the port ( if any ) is valid.
  --           The value returned in value is that of the given address
  --           at the moment the port is read by the host file server.
```

13 streamio.lib

45 entry points

The key stream and screen stream protocols used by the routines in this library have their origins in the Transputer Development System (TDS). The key stream protocol (**KS**) is, in fact, simply **INT**. This gives rise to some confusing terminology because the TDS and the naming of some of the routines in this library refer to characters when actually meaning an **INT**, a word length quantity. The TDS requires values outside the ASCII range for its operation and so uses word length quantities, and due to **streamio.lib** being intended to facilitate porting of the TDS to the toolset, the same terminology is used for the naming of the routines. Hence, for example, **ks.read.char** actually reads a word length quantity rather than a **BYTE**.

There is a naming convention used in this library. Procedures always begin with a prefix derived from the first parameter. Stream processes (ones which convert streams from keyboard or screen protocol to the server protocol **SP** or to related data structures), where the **SP** channel is used in combination with either the **KS** or **SS** protocols, are prefixed with '**so.**'. Stream input routines, (that is, ones which use only the **KS** protocol) are prefixed with '**ks.**', and stream output routines (that is, ones which use only the **SS** protocol) are prefixed with '**ss.**'. The single **KS** to **SS** conversion routine, which uses both protocols, is prefixed with '**ks.**' because it is the **KS** parameter which is first.

Constants referred to in the following specifications are to be found in the include files **hostio.inc** and **streamio.inc**.

```
PROC ks.keystream.to.scrstream (CHAN OF KS keyboard, CHAN OF SS scrn)

  -- Purpose:  To convert keystream protocol into screen
  --           stream protocol.
  -- Channels: keyboard - for input
  --           scrn - for output
  -- Notes:    This procedure is terminated by the receipt
  --           of ft.terminated from the keyboard stream.
  --           For keystream values above ft.tag, only
  --           ft.tag + 1 to ft.tag + 8 are dealt with,
  --           values above ft.tag + 8, and ft.tag itself are
  --           translated into st.beep.
  --           Keystream values which are negative, other than
  --           ft.terminated, are ignored.
  --           Keystream values in the range [0, ft.tag) are
  --           passed straight through, other than '*c' which
  --           is translated into '*c' followed by '*n'.
```

```
PROC ks.keystream.sink (CHAN OF KS keys)

  -- Purpose:  To read word length quantities from the given key
  --           stream until ft.terminated is received, at which
  --           point this procedure terminates.
  -- Channels: keys - for input


PROC so.keystream.from.file (CHAN OF SP fs, ts, CHAN OF KS keys.out,
                             VAL []BYTE filename, BYTE result)

  -- Purpose:  To read lines from a text file and output them on the
  --           given key stream channel.
  -- Channels: fs - from server
  --           ts - to server
  --           keys.out - for output
  -- In:       filename - the name of the file to be opened.  The name of
  --           the file must fit exactly into filename, i.e. there are
  --           (SIZE filename) characters in the name of the file.  A
  --           directory specification may form part of filename.
  -- Out:      result - spr.ok if the process was successful; otherwise
  --           it takes on a value indicating what went wrong:
  --             spr.bad.packet.size   filename too large:
  --                                   (SIZE filename) > sp.max.openname.size
  --             spr.bad.name          null file name
  --           >=spr.operation.failed  the open failed or reading the file
  --                                   failed - see hostio.inc or iserver
  --                                   documentation for further details
  -- Notes:    The value ft.terminated is sent on keys.out on termination.
  --           Termination of this procedure is either when an error occurs
  --           or when all characters in the file have been read.
  --           A '*c' is output to terminate a text line.


PROC so.keystream.from.kbd (CHAN OF SP fs, ts,
                            CHAN OF KS keys.out,
                            CHAN OF BOOL stopper,
                            VAL INT ticks.per.poll)

  -- Purpose:  To read keys from the keyboard at full speed if they
  --           are available, otherwise waiting a given interval
  --           between polls of the keyboard, and output them as
  --           integers on the given key stream.
  -- Channels: fs - from server
  --           ts - to server
  --           keys.out - for output
  --           stopper - for input; FALSE or TRUE received on this
  --           channel terminates this procedure
  -- In:       ticks.per.poll - this procedure polls the keyboard at
  --           intervals of ticks.per.poll low priority transputer
  --           clock cycles if keys are not available; a value of
  --           less than or equal to zero is an error
  -- Notes:    On termination, ft.terminated is sent on keys.out.
```

```
PROC so.keystream.from.stdin (CHAN OF SP fs, ts, CHAN OF KS keys.out,
                              BYTE result)

  -- Purpose:  To read lines from standard input and output them on the
  --           given key stream channel.
  -- Channels: fs - from server
  --           ts - to server
  --           keys.out - for output
  -- Out:      result - spr.ok if the process was successful; otherwise
  --           >= spr.operation.failed meaning that reading standard input
  --           failed - see hostio.inc or iserver documentation for further
  --           details
  -- Notes:    The value ft.terminated is sent on keys.out on termination.
  --           Termination of this procedure is either when an error occurs
  --           or when all characters from standard input have been read,
  --           ie when end of file from standard input is read.
  --           A '*c' is output to terminate a text line.


PROC ss.scrstream.copy (CHAN OF SS scrn.in, scrn.out)

  -- Purpose:  To copy the output of one screen stream to the
  --           input of another.
  -- Channels: scrn.in - for input
  --           scrn.out - for output
  -- Notes:    Receipt of st.endstream on scrn.in will terminate
  --           this procedure, without the stream terminator
  --           being passed on.


PROC ss.scrstream.fan.out (CHAN OF SS scrn, screen.out1, screen.out2)

  -- Purpose:  To copy everything received on the given screen
  --           stream input channel to both of the given screen
  --           stream output channels.
  -- Channels: scrn - for input
  --           screen.out1 - for output
  --           screen.out2 - for output
  -- Notes:    Receipt of st.endstream on scrn will terminate
  --           this procedure, without sending the stream
  --           terminator on.


PROC ss.scrstream.sink (CHAN OF SS scrn)

  -- Purpose:  To ignore all but st.endstream of screen stream protocol,
  --           and to terminate when it does receive st.endstream.
  -- Channels: scrn - for input


PROC ss.scrstream.to.array (CHAN OF SS scrn, []BYTE buffer)

  -- Purpose:  To put the data received on given screen stream
  --           channel into given array.
  -- Channels: scrn - for input
```

```
  -- Out:      buffer - the array to which the data received on
  --           scrn is placed; the data is stored in the form of
  --           TDS2 tt. tags followed as necessary by further data
  --           bytes; if buffer overflows then this procedure acts
  --           as an invalid process
  -- Notes:    This procedure terminates when st.endstream is read
  --           on channel scrn.  This terminating st.endstream is
  --           stored in buffer as tt.endstream.

PROC ss.scrstream.from.array (CHAN OF SS scrn, VAL []BYTE buffer)

  -- Purpose:  To output the data in the given array on the given
  --           screen stream channel.
  -- Channels: scrn - for output
  -- Out:      buffer - the array from which the data is to be
  --           sent on scrn; the data should be in the form of
  --           TDS2 tt. tags, followed as necessary by further
  --           data bytes.
  -- Notes:    This procedure terminates when tt.endstream is found
  --           in buffer.  This tt.endstream is not sent out ( as
  --           st.endstream or anything else ).

PROC so.scrstream.to.file (CHAN OF SP fs, ts, CHAN OF SS scrn,
                           VAL []BYTE filename, BYTE result)

  -- Purpose:  To write the data sent on the given screen stream to
  --           a file.
  -- Channels: fs - from server
  --           ts - to server
  --           scrn - for input
  -- In:       filename - the name of the file to be write to.  The name
  --           of the file must fit exactly into filename, i.e. there are
  --           (SIZE filename) characters in the name of the file.  A
  --           directory specification may form part of filename.  If a
  --           file of the same name exists it is overwritten.
  -- Out:      result - spr.ok if the data sent on scrn was successfully
  --           written to the file filename; otherwise it takes on a
  --           value indicating what went wrong:
  --             spr.bad.packet.size   filename too large
  --                                     (SIZE filename) > sp.max.openname.size
  --             spr.bad.name          null file name
  --           >=spr.operation.failed  see hostio.inc or iserver documentation
  --                                   for further details
  -- Notes:    This routine terminates on receipt of st.endstream on scrn.
  --           What is written to the file is the bytes that follow tags
  --           ( for those tags that have them ), with the exception that
  --           the st.out.int tag has its following INT32 truncated to a BYTE.

PROC so.scrstream.to.stdout (CHAN OF SP fs, ts, CHAN OF SS scrn, BYTE result)

  -- Purpose:  To write the data sent on the given screen stream to
  --           standard out.
  -- Channels: fs - from server
```

```
  --           ts - to server
  --           scrn - for input
  -- Out:      result - spr.ok if the data sent on scrn was successfully
  --           written to standard out; otherwise is >= spr.operation.failed
  --           in which case see hostio.inc or iserver documentation
  --           for further details
  -- Notes:    This routine terminates on receipt of st.endstream on scrn.
  --           What is written to the file is the bytes that follow tags
  --           ( for those tags that have them ), with the exception that
  --           the st.out.int tag has its following INT32 truncated to a BYTE,
  --           and that the st.beep tag is transformed into '*#07'.

PROC ss.scrstream.multiplexor ([]CHAN OF SS screen.in,
                               CHAN OF SS screen.out,
                               CHAN OF INT stopper)

  -- Purpose:  To multiplex up to 256 screen stream channels onto
  --           a single such channel.
  -- Channels: screen.in - an array of input channels
  --           screen.out - the single output channel
  --           stopper - input to this routine; any integer received
  --           on this channel will terminate this routine; this
  --           channel has highest priority
  -- Notes:    It is an error if there are more than 256 channels in
  --           screen.in.
  --           It is permissible for screen.in to be a null array.
  --           Each change of input channel directs output to the
  --           next line of the screen, and each such line is
  --           annotated at the left with the array index of the
  --           channel used followed by '>'.
  --           The tag st.endstream is ignored.
  --           To attempt some degree of fairness, let the implementation
  --           give priority in a heirarchy from screen.in with index i for
  --           SIZE screen.in, using modulo SIZE screen.in on the indexes;
  --           starting from 0 and incrementing it by one after each input
  --           accepted.

PROC ks.read.char (CHAN OF KS source, INT char)

  -- Purpose:  To obtain the value of the next word length quantity
  --           from the given keystream channel.
  -- Channels: source - for input
  -- Out:      char - the value of the word obtained

PROC ks.read.line (CHAN OF KS source, INT len, []BYTE line, INT char)

  -- Purpose:  To read a line of text from the given keystream channel.
  -- Channels: source - for input
  -- Out:      len - the number of characters that have been put
  --           into line
  -- Out:      line - the line that was read is [ line FROM 0 FOR len ]
  -- Out:      char - the word that terminated the line
  -- Notes:    Although characters ( ie BYTE's ) are inserted into line,
```

```
  --           it is word length quantities that are read from source.
  --           The line of text is deemed to be terminated by INT '*c'
  --           or any negative value.
  --           The word that terminated the line is not included
  --           in line.
  --           Any word read with a value greater than 255 ( the
  --           maximum that a byte can hold ) is ignored.
  --           Any '*n' read is ignored.
  --           If the array line is filled before a valid termination
  --           character is encountered, then all further words
  --           are ignored.

PROC ks.read.int (CHAN OF KS source, INT number, char)

  -- Purpose:  To read a decimal or hexadecimal integer from the given
  --           keystream.
  -- Channels: source - for input
  -- Out:      number - if char is not ft.number.error or negative,
  --           the value of the integer read; otherwise undefined
  -- In/Out:   char - on entry: the first 'character' from the input
  --           to be read; on exit: ft.number.error if the integer
  --           read overflowed the range of INT; a negative value
  --           ( other than ft.number.error ) indicating an input
  --           error; otherwise the 'character' that terminated the
  --           integer
  -- Notes:    A distinction must be made here between a character held
  --           in a byte and a character held in a word length quantity.
  --           Here the former is referred to as a character and the
  --           latter as a 'character' ( note the quotes ).
  --           Upper and lower case letters are permissible in a
  --           hexadecimal integer.
  --           All input up to a plus sign, a minus sign, a hash symbol,
  --           decimal digit, or a negative valued word is skipped.  If
  --           it is the hash that is encountered first then it is a
  --           hexadecimal integer ( in two's complement form ) that is
  --           expected.  If it is a negative valued word that is
  --           encountered then an input error has occurred and that
  --           value is returned in char.
  --           The integer is terminated when anything other than a valid
  --           digit ( hex and/or decimal, as the case may be ) is read,
  --           and it is this terminating quantity that is returned in char
  --           if the integer read is not invalid.

PROC ks.read.int64 (CHAN OF KS source, INT64 number, INT char)

  -- Purpose:  To read a 64-bit decimal or hexadecimal integer from the
  --           given keystream.
  -- Channels: source - for input
  -- Out:      number - if char is not ft.number.error, the value of the
  --           integer read; otherwise undefined
  -- In/Out:   char - on entry: the first 'character' from the input
  --           to be read; on exit: ft.number.error if the integer
  --           read overflowed the range of INT64; otherwise the
  --           'character' that terminated the integer
```

```
  -- Notes:    A distinction must be made here between a character held
  --           in a byte and a character held in a word length quantity.
  --           Here the former is referred to as a character and the
  --           latter as a 'character' ( note the quotes ).
  --           Upper and lower case letters are permissible in a
  --           hexadecimal integer.
  --           All input up to a plus sign, a minus sign, a hash symbol,
  --           decimal digit, or a negative valued word is skipped.  If
  --           it is the hash that is encountered first then it is a
  --           hexadecimal integer ( in two's complement form ) that is
  --           expected.  If it is a negative valued word that is
  --           encountered then an input error has occurred and that
  --           value is returned in char.
  --           The integer is terminated when anything other than a valid
  --           digit ( hex and/or decimal, as the case may be ) is read,
  --           and it is this terminating quantity that is returned in char
  --           if the integer read is not invalid.

PROC ks.read.real32 (CHAN OF KS source, REAL32 number, INT char)

  -- Purpose:  To read a single precision real number from the given
  --           keystream.
  -- Channels: source - for input
  -- Out:      number - if char on exit is not ft.number.error or less
  --           than zero then the real number read in single precision
  --           IEEE format; otherwise undefined
  -- In/Out:   char - on entry: the first 'character' from the input
  --           to be read; on exit: ft.number.error if the 'characters'
  --           read did not form an occam syntax real number ( with
  --           optional plus or minus sign as first 'character',
  --           and permissible INT 'e' rather than INT 'E' ) or there
  --           were more than 24 'characters' read or the 'characters'
  --           formed an infinity; less than zero ( other than
  --           ft.number.error ) if there was an input error ( eg end
  --           of file read ); otherwise it is the 'character' that
  --           terminated the sequence of 'characters' read
  -- Notes:    A distinction must be made here between a character held
  --           in a byte and a character held in a word length quantity.
  --           Here the former is referred to as a character and the
  --           latter as a 'character' ( note the quotes ).
  --           All input up to a plus sign, a minus sign or a decimal digit
  --           is skipped.
  --           The real is terminated when anything other than a valid real
  --           'character' is encountered, and it is this 'character' that
  --           is returned in char if char is not set to ft.number.error.

PROC ks.read.real64 (CHAN OF KS source, REAL64 number, INT char)

  -- Purpose:  To read a double precision real number from the given
  --           keystream.
  -- Channels: source - for input
  -- Out:      number - if char on exit is not ft.number.error or less
  --           than zero then the real number read in double precision
  --           IEEE format; otherwise undefined
```

```
  -- In/Out:   char - on entry: the first 'character' from the input
  --           to be read; on exit: ft.number.error if the 'characters'
  --           read did not form an occam syntax real number ( with
  --           optional plus or minus sign as first 'character',
  --           and permissible INT 'e' rather than INT 'E' ) or there
  --           were more than 30 'characters' read or the 'characters'
  --           formed an infinity; less than zero ( other than
  --           ft.number.error ) if there was an input error ( eg end
  --           of file read ); otherwise it is the 'character' that
  --           terminated the sequence of 'characters' read
  -- Notes:    A distinction must be made here between a character held
  --           in a byte and a character held in a word length quantity.
  --           Here the former is referred to as a character and the
  --           latter as a 'character' ( note the quotes ).
  --           All input up to a plus sign, a minus sign or a decimal digit
  --           is skipped.
  --           The real is terminated when anything other than a valid real
  --           'character' is encountered, and it is this 'character' that
  --           is returned in char if char is not set to ft.number.error.

PROC ss.write.int (CHAN OF SS scrn, VAL INT number, field )

  -- Purpose:  To write to the given screen stream an integer as decimal
  --           ASCII digits, padded out with leading spaces to the
  --           specified field width.
  -- Channels: scrn - for output
  -- In:       number - the integer that is desired to be written
  -- In:       field - the desired field width of the string
  -- Notes:    If the field width is too small for the number, then it is
  --           widened as necessary; a zero value for the field width will
  --           give minimum width; a negative field width is an error.

PROC ss.write.hex.int (CHAN OF SS scrn, VAL INT number, field )

  -- Purpose:  To write an integer as hexadecimal ASCII characters
  --           preceded by the '#' character to the given screen stream.
  -- Channels: scrn - for output
  -- In:       number - the integer to write out
  -- In:       field - the field width to use when writing the hex;
  --           if field is larger than the size of the number then the
  --           number is padded with leading 0's or F's as appropriate;
  --           if field is smaller than the size of the number then the
  --           number is truncated at the left to field characters;
  --           field does not take the '#' into account.
  --           A negative field width is an error.

PROC ss.write.int64 (CHAN OF SS scrn, VAL INT64 number, VAL INT field )

  -- As ss.write.int, but for INT64s

PROC ss.write.hex.int64 (CHAN OF SS scrn, VAL INT64 number, VAL INT field )
```

```
  -- As ss.write.hex.int, but for INT64s

PROC ss.write.real32 (CHAN OF SS scrn, VAL REAL32 number, VAL INT Ip, Dp)

  -- Purpose:  To format into ASCII characters and then write to the
  --           given screen stream a single precision real number.
  -- Channels: scrn - for output
  -- In:       number - the IEEE single precision real to format
  --           and write out
  -- In, In:   Ip, Dp - formatting values; see REAL32TOSTRING for
  --           their effect
  -- Notes:    If Ip, Dp and number are such that the resulting
  --           formatted form is longer than 24 characters this
  --           procedure will act as an invalid process.

PROC ss.write.real64 (CHAN OF SS scrn, VAL REAL64 number, VAL INT Ip, Dp)

  -- Purpose:  To format into ASCII characters and then write to the
  --           given screen stream a double precision real number.
  -- Channels: scrn - for output
  -- In:       number - the IEEE double precision real to format
  --           and write out
  -- In, In:   Ip, Dp - formatting values; see REAL64TOSTRING for
  --           their effect
  -- Notes:    If Ip, Dp and number are such that the resulting
  --           formatted form is longer than 30 characters this
  --           procedure will act as an invalid process.

PROC ss.write.char (CHAN OF SS scrn, VAL BYTE char)

  -- Purpose:  To send a character to the given screen stream.
  -- Channels: scrn - for output
  -- In:       char - the byte to send on the channel

PROC ss.write.string (CHAN OF SS scrn, VAL []BYTE str)

  -- Purpose:  To send the given string on the given screen stream.
  -- Channels: scrn - for output
  -- In:       str - a string all the characters of which are sent
  --           on scrn.

PROC ss.write.nl (CHAN OF SS scrn)

  -- Purpose:  To send "*c*n" on the given screen stream.
  -- Channels: scrn - for output

PROC ss.write.endstream (CHAN OF SS scrn)

  -- Purpose:  To write a st.endstream tag on the given screen stream
  -- Channels: scrn - for output
```

```
PROC ss.write.text.line (CHAN OF SS scrn, VAL []BYTE str)

  -- Purpose:  To send a text line on the given screen stream.
  -- Channels: scrn - for output
  -- In:       str - the string of characters to send.  This
  --           string may be terminated by '*c' or not, but
  --           in either case, the last two characters written
  --           are "*c*n".


PROC so.scrstream.to.ANSI (CHAN OF SP fs, ts, CHAN OF SS scrn)

  -- Purpose:  To convert output in screen stream protocol to
  --           output using ANSI screen conventions.
  -- Channels: fs - from server
  --           ts - to server
  --           scrn - for input
  -- Notes:    Receipt of st.endstream will terminate this procedure.
  --           ANSI screen convertions are set out in ANSI X3.64-1979
  --           "Additional controls for use with american national
  --           standard code for information interchange."


PROC so.scrstream.to.TVI920 (CHAN OF SP fs, ts, CHAN OF SS scrn)

  -- Purpose:  To convert output in screen stream protocol to
  --           output using TVI920 screen conventions.
  -- Channels: fs - from server
  --           ts - to server
  --           scrn - for input
  -- Notes:    Receipt of st.endstream will terminate this procedure.


PROC ss.goto.xy (CHAN OF SS scrn, VAL INT x, y)

  -- Purpose:  To place the cursor on the terminal screen at
  --           the given position.
  -- Channels: scrn - for output
  -- In, In:   x, y - the coordinates of where to place the
  --           cursor, where the origin (0, 0) is at the top
  --           left hand corner of the screen


PROC ss.clear.eol (CHAN OF SS scrn)

  -- Purpose:  To clear the terminal screen from the cursor
  --           position to the end of the current line.
  -- Channels: scrn - for output


PROC ss.clear.eos (CHAN OF SS scrn)

  -- Purpose:  To clear the terminal screen from the cursor
  --           position to the end of the current line and
```

```
  --           then to the end of the screen.
  -- Channels: scrn - for output

PROC ss.beep (CHAN OF SS scrn)

  -- Purpose:  To sound the computer's bell.
  -- Channels: scrn - for output

PROC ss.up (CHAN OF SS scrn)

  -- Purpose:  To move the cursor on the terminal screen one
  --           line up.
  -- Channels: scrn - for output

PROC ss.down (CHAN OF SS scrn)

  -- Purpose:  To move the cursor on the terminal screen one
  --           line down.
  -- Channels: scrn - for output

PROC ss.left (CHAN OF SS scrn)

  -- Purpose:  To move the cursor on the terminal screen one
  --           place to the left.
  -- Channels: scrn - for output

PROC ss.right (CHAN OF SS scrn)

  -- Purpose:  To move the cursor on the terminal screen one
  --           place to the right.
  -- Channels: scrn - for output

PROC ss.insert.char (CHAN OF SS scrn, VAL BYTE ch)

  -- Purpose:  To move the character at the cursor and all
  --           those to the right of it on the terminal screen
  --           one place to the right and then to insert the given
  --           character at the cursor.  The cursor moves one
  --           place to the right.
  -- Channels: scrn - for output
  -- In:       ch - the character to insert

PROC ss.delete.chr (CHAN OF SS scrn)

  -- Purpose:  To delete the character at the cursor on the
  --           terminal screen and move the rest of the
  --           line one place to the left. The cursor does
  --           not move.
  -- Channels: scrn - for output
```

```
PROC ss.delete.chl (CHAN OF SS scrn)

  -- Purpose:  To delete the character to the left of the cursor
  --           on the terminal screen and move the rest of the
  --           line one place to the left. The cursor also moves
  --           one place to the left.
  -- Channels: scrn - for output


PROC ss.ins.line (CHAN OF SS scrn)

  -- Purpose:  To move all lines below the current line on the terminal
  --           down one line, losing the bottom line, with the inserted
  --           line blank.
  -- Channels: scrn - for output


PROC ss.del.line (CHAN OF SS scrn)

  -- Purpose:  To delete the current line on the terminal and move
  --           all lines below it up one line.
  -- Channels: scrn - for output
  -- Notes:    The bottom line of the terminal screen becomes blank.
```

14 string.lib

27 entry points

```
BOOL FUNCTION is.in.range (VAL BYTE char, bottom, top)

  -- Purpose:  To determine whether the value of a byte lies in the
  --           inclusive range between two others.
  -- Returned: TRUE if the value of char lies in the range
  --           [value of bottom, value of top]; FALSE otherwise
  -- In:       char - the byte the value of which is to be tested to
  --           see whether it lies in a given range
  -- In:       bottom - the lowermost limit of the test range
  -- In:       top - the uppermost limit of the test range


BOOL FUNCTION is.upper (VAL BYTE char)

  -- Purpose:  To determine whether the value of a byte lies in the
  --           inclusive range that delimits upper case ASCII characters.
  -- Returned: TRUE if the value of char corresponds to an upper case
  --           ASCII character; FALSE otherwise
  -- In:       char - the byte the value of which is to be tested


BOOL FUNCTION is.lower (VAL BYTE char)
```

```
  -- Purpose:  To determine whether the value of a byte lies in the
  --           inclusive range that delimits lower case ASCII characters.
  -- Returned: TRUE if the value of char corresponds to a lower case
  --           ASCII character; FALSE otherwise
  -- In:       char - the byte the value of which is to be tested

BOOL FUNCTION is.digit (VAL BYTE char)

  -- Purpose:  To determine whether the value of a byte lies in the
  --           inclusive range that delimits ASCII decimal digits.
  -- Returned: TRUE if the value of char corresponds to a decimal digit
  --           according to the ASCII code; FALSE otherwise
  -- In:       char - the byte the value of which is to be tested

BOOL FUNCTION is.hex.digit (VAL BYTE char)

  -- Purpose:  To determine whether the value of a byte corresponds to
  --           the ASCII value of any hexadecimal digit.
  -- Returned: TRUE if the value of char corresponds to a hexadecimal
  --           digit according to the ASCII code, where upper or
  --           lower case letters are allowed; FALSE otherwise
  -- In:       char - the byte the value of which is to be tested

BOOL FUNCTION is.id.char (VAL BYTE char)

  -- Purpose:  To determine whether the value of a byte corresponds
  --           to the ASCII code of any legal occam identifier character.
  -- Returned: TRUE if the value of char corresponds to the ASCII code of
  --           any legal occam identifier character
  -- In:       char - the byte the value of which is to be tested

PROC to.upper.case ([]BYTE str)

  -- Purpose:  To convert all lower case alphabetic characters in a given
  --           string to upper case.
  -- In/Out:   str - the string the lower case characters of which are to
  --           be converted to upper case
  -- Notes:    Assumes the ASCII character set.
  --           Characters which are not lower case letters remain unchanged.

PROC to.lower.case ([]BYTE str)

  -- Purpose:  To convert all upper case alphabetic characters in a given
  --           string to lower case.
  -- In/Out:   str - the string the upper case characters of which are to
  --           be converted to lower case
  -- Notes:    Assumes the ASCII character set.
  --           Characters which are not upper case letters remain unchanged.

INT FUNCTION compare.strings (VAL []BYTE str1, str2)
```

```
  -- Purpose:  To determine the lexicographical ordering of two strings.
  -- Returned: 0 for exact equality ( of both length and content )
  --           1 for str2 is leading substring of str1
  --          -1 for str1 is leading substring of str2
  --           2 for str1 "later" than str2
  --          -2 for str2 "later" than str1
  -- In, In:   str1, str2 - the strings to be compared
  -- Notes:    Lexicographical ordering is that which uses the ordinal values
  --           of the characters for comparison in sequence along the strings.
  --           Here the ordinal values are the ASCII values.

BOOL FUNCTION eqstr (VAL []BYTE s1, s2)

  -- Purpose:  To determine if two strings are identical or not.
  -- Returned: TRUE if the two strings are identical in length and content;
  --           FALSE otherwise
  -- In, In:   s1, s2 - the strings to be compared

PROC str.shift ([]BYTE str, VAL INT start, len, shift, BOOL not.done)

  -- Purpose:  To shift a substring.
  -- In/Out:   str - on entry: a string containing the substring to be shifted
  --           in positions [str FROM start FOR len]; on exit: the string once
  --           the substring has been shifted, the only bytes of string that
  --           have changed being those that the substring was shifted into
  -- In:       start - the index of str of the first character of the substring
  --           to be shifted
  -- In:       len - the number of characters in the substring to be shifted
  -- In:       shift - the number of places to the right to move the substring
  --           by, so that a negative number for shift will move the substring
  --           left
  -- Out:      not.done - TRUE if any elements of the substring are shifted off
  --           either end of str ( though no access is made to invalid locations
  --           of str ); FALSE if the shifted substring is entirely within str

PROC delete.string (INT len, []BYTE str, VAL INT start, size, BOOL not.done)

  -- Purpose:  To remove a substring from a string.
  -- In/Out:   len - on entry: the number of significant characters in str;
  --           on exit: if not.done is FALSE, the number of significant
  --           characters in str, being size subtracted from the entry
  --           value of len; otherwise the same as it was on entry
  -- In/Out:   str - on entry: the string from which it is desired to delete
  --           a substring; on exit: if not.done is FALSE, the original
  --           string with substring deleted, where deleted means that the gap
  --           created by the deletion is filled from the left with those
  --           significant characters left in str that were originally after
  --           the end of the deleted substring, and so that the number of
  --           significant characters remaining in str is len on exit;
  --           otherwise the same as it was on entry
  -- In:       start - the index of str of the first character of the
  --           substring to be deleted
  -- In:       size - the number of characters in the substring to be deleted
```

```
  -- Out:      not.done - TRUE if size is less than zero, start is less than
  --           zero, or (start + size) is greater than the entry value of len;
  --           FALSE otherwise.
  --           If TRUE then len and str are unchanged from their original
  --           values.

PROC insert.string (VAL []BYTE new.str, INT len,
                    []BYTE str, VAL INT start,
                    BOOL not.done)

  -- Purpose:  To insert a string into another string.
  -- In:       new.str - the string to be inserted
  -- In/Out:   len - on entry: the number of significant characters in str;
  --           on exit: the number of significant characters in str
  -- In/Out:   str - on entry: the string into which new.str is to be
  --           inserted; on exit: the original str with new.str inserted,
  --           where any overflow of str at the high index results in
  --           truncation at the high index
  -- In:       start - the index of str at which the first character of
  --           new.str should go
  -- Out:      not.done - TRUE if start < 0, start > len, len < 0, new.str
  --           had to be truncated to fit, or if any significant characters
  --           of the original str could not be retained within str after the
  --           insertion; otherwise FALSE
  -- Notes:    If new.str can be fully inserted from the desired starting
  --           position, then any significant characters in str that were
  --           originally after str[ start ] are moved to the right by
  --           SIZE new.str, with not.done being set to TRUE if any of these
  --           characters are moved off the end of str ( though no invalid
  --           accesses are made ), FALSE otherwise

INT FUNCTION string.pos (VAL []BYTE search, str)

  -- Purpose:  To determine where the first occurrence is of a string
  --           within another string.
  -- Returned: the lowest index of str at which begins a substring exactly
  --           matching the string search was found, or -1 if no such
  --           substring found
  -- In:       search - the string to search for in the string str
  -- In:       str - the string in which to search for the string search
  -- Notes:    The searching is case sensitive.

INT FUNCTION char.pos (VAL BYTE search, VAL []BYTE str)

  -- Purpose:  To determine where the first occurrence of a character is
  --           in a given string.
  -- Returned: the lowest index of str at which a byte exactly matching
  --           search was found, or -1 if no such byte found
  -- In:       search - the character to search for in the string str
  -- In:       str - the string in which to search for the character search
  -- Notes:    The searching is case sensitive.
```

```
INT, BYTE FUNCTION search.match (VAL []BYTE possibles, str)

  -- Purpose:  To search a string for the first occurrence of any one of
  --           a given number of characters.
  -- Returned: From left to right:
  --           - if a match found, this gives the lowest index of str at
  --             which the match occurs; otherwise -1
  --           - if a match found, this gives the byte of possibles which
  --             was found; otherwise 255( BYTE )
  -- In:       possibles - a string each byte of which is to be individually
  --           checked for in str
  -- In:       str - the string in which to search for any of the various
  --           bytes contained in possibles


INT, BYTE FUNCTION search.no.match (VAL []BYTE possibles, str)

  -- Purpose:  To search a string for the first occurrence of a character
  --           which does not match any of a given number of characters.
  -- Returned: From left to right:
  --           - if no match found, this gives the lowest index of str at
  --             which the lack of a match occurs; otherwise -1
  --           - if no match found, this gives the byte of str which
  --             was found not to match; otherwise 255( BYTE )
  -- In:       possibles - a string each byte of which is to be individually
  --           checked for non-existence in str
  -- In:       str - the string in which to search for a byte which is not
  --           identical to any of the various bytes contained in possibles


PROC append.char (INT len, []BYTE str, VAL BYTE char)

  -- Purpose:  To write a byte into a string.
  -- In/Out:   len - on entry: the byte char will be written at str[ len ];
  --           on exit: len will be one greater than it was on entry
  -- Out:      str - the string to write char to
  -- In:       char - the byte which is to be written into str
  -- Notes:    If accessing str[ len ], for len on entry, is invalid then
  --           this routine acts as an invalid process.


PROC append.text (INT len, []BYTE str, VAL []BYTE text)

  -- Purpose:  To concatenate two strings.
  -- In/Out:   len - on entry: the index of str where the first character of
  --           text is to go; on exit: the index of str immediately after the
  --           last character of text inserted, or SIZE str if the last
  --           character of text was placed in the last position of str
  -- Out:      str - the concatenation of what str was on entry with text,
  --           where text is placed in positions str[ len ] to
  --           str[ len + (SIZE text) - 1 ] inclusive, where len here is that
  --           on entry
  -- In:       text - the string to be concatenated with str, text being the
  --           second string
  -- Notes:    If str is not long enough to hold the concatenation then
```

```
  --           this routine acts as an invalid process.

PROC append.int (INT len, []BYTE str, VAL INT number, field)

  -- Purpose:  To convert an integer into its representation as ASCII decimal
  --           digits, with leading spaces if desired, and write this into
  --           a given string.
  -- In/Out:   len - on entry: the index of str at which the first character
  --           of the ASCII conversion of number is to be written; on exit:
  --           the index of str immediately after where the last character of
  --           the ASCII conversion of number was written, or SIZE str if
  --           this last character was written into the last positon of str
  -- Out:      str - the string into which the ASCII conversion of number is
  --           to be written
  -- In:       number - the integer to be converted to an ASCII representation
  --           and then written into str
  -- In:       field - the field width of the ASCII representation of number:
  --           if number cannot be represented in field characters then the
  --           representation is widened as necessary; if field is larger
  --           than necessary then padding spaces are added on the left; it
  --           is an error if field is negative
  -- Notes:    If str overflows then this routine acts as an invalid process.
  --           The conversion of number will include a minus sign if applicable.

PROC append.hex.int (INT len, []BYTE str, VAL INT number, width)

  -- Purpose:  To convert an integer into its representation as ASCII
  --           hexadecimal characters and write this into a given string.
  -- In/Out:   len - on entry: the index of str at which the first character
  --           of the ASCII hexadecimal of number is to be written; on exit:
  --           the index of str immediately after where the last character of
  --           the ASCII hexadecimal of number was written, or SIZE str if
  --           this last character was written into the last positon of str
  -- Out:      str - the string into which the hexadecimal ASCII form of
  --           number is to be written
  -- In:       number - the integer to be converted to an ASCII hexadecimal
  --           representation and then written into str
  -- In:       width - the field width of the ASCII hexadecimal representation
  --           of number: if number cannot be represented in width characters
  --           then the representation is truncated at the left as necessary;
  --           otherwise the representation is padded on the left with 0's or
  --           F's to make up width characters; it is an error if width is
  --           negative
  -- Notes:    If str overflows then this routine acts as an invalid process.
  --           The conversion of number includes a # as the first character,
  --           so that the representation is always ( width + 1 ) characters.
  --           Any hexadecimal characters which are letters will be in upper
  --           case.

PROC append.real32 (INT len, []BYTE str,
                    VAL REAL32 number, VAL INT Ip, Dp)

  -- Purpose:  To write an ASCII representation of a single precision
```

```
  --           real number into a given string.
  -- In/Out:   len - on entry: the index of str at which the first character
  --           of the representation of number is to be placed; on
  --           exit: the index of str of the byte immediately following the
  --           last character in str of the representation of number,
  --           or (SIZE str) if the last character of the representation
  --           was placed into the last byte of str
  -- Out:      str - the string into which to place the ASCII representation
  --           of number
  -- In:       number - a single precision real number in IEEE format
  -- In, In:   Ip, Dp - formatting values for the real number; see
  --           REAL32TOSTRING for their effect
  -- Notes:    If str overflows this routine acts as an invalid process.

PROC append.real64 (INT len, []BYTE str,
                    VAL REAL64 number, VAL INT Ip, Dp)

  -- Purpose:  To write an ASCII representation of a double precision
  --           real number into a given string.
  -- In/Out:   len - on entry: the index of str at which the first character
  --           of the representation of number is to be placed; on
  --           exit: the index of str of the byte immediately following the
  --           last character in str of the representation of number,
  --           or (SIZE str) if the last character of the representation
  --           was placed into the last byte of str
  -- Out:      str - the string into which to place the ASCII representation
  --           of number
  -- In:       number - a double precision real number in IEEE format
  -- In, In:   Ip, Dp - formatting values for the real number; see
  --           REAL64TOSTRING for their effect
  -- Notes:    If str overflows this routine acts as an invalid process.

PROC append.int64 (INT len, []BYTE str, VAL INT64 number, VAL INT field)

  -- Purpose:  To convert a 64-bit integer into its representation as ASCII
  --           decimal digits, with leading spaces if desired, and write
  --           this into a given string.
  -- In/Out:   len - on entry: the index of str at which the first character
  --           of the ASCII conversion of number is to be written; on exit:
  --           the index of str immediately after where the last character of
  --           the ASCII conversion of number was written, or SIZE str if
  --           this last character was written into the last positon of str
  -- Out:      str - the string into which the ASCII conversion of number is
  --           to be written
  -- In:       number - the 64-bit integer to be converted to an ASCII
  --           representation and then written into str
  -- In:       field - the field width of the ASCII representation of number:
  --           if number cannot be represented in field characters then the
  --           representation is widened as necessary; if field is larger
  --           than necessary then padding spaces are added on the left; it
  --           is an error for field to be negative
  -- Notes:    If str overflows then this routine acts as an invalid process.
  --           The conversion of number will include a minus sign if applicable.
```

```
PROC append.hex.int64 (INT len, []BYTE str, VAL INT64 number, VAL INT width)

  -- Purpose:  To convert a 64-bit integer into its representation as ASCII
  --           hexadecimal characters and write this into a given string.
  -- In/Out:   len - on entry: the index of str at which the first character
  --           of the ASCII hexadecimal of number is to be written; on exit:
  --           the index of str immediately after where the last character of
  --           the ASCII hexadecimal of number was written, or SIZE str if
  --           this last character was written into the last positon of str
  -- Out:      str - the string into which the hexadecimal ASCII form of
  --           number is to be written
  -- In:       number - the 64-bit integer to be converted to an ASCII
  --           hexadecimal representation and then written into str
  -- In:       width - the field width of the ASCII hexadecimal representation
  --           of number: if number cannot be represented in width characters
  --           then the representation is truncated at the left as necessary;
  --           otherwise the representation is padded on the left with 0's or
  --           F's to make up width characters; a negative value for width
  --           is an error
  -- Notes:    If str overflows then this routine acts as an invalid process.
  --           The conversion of number includes a # as the first character,
  --           so that the representation is always ( width + 1 ) characters.
  --           Any hexadecimal characters which are letters will be in upper
  --           case.

PROC next.word.from.line (VAL []BYTE line, INT ptr, len, []BYTE word, BOOL ok)

  -- Purpose:  To determine the next word in a given line, skipping leading
  --           spaces and tabs.
  -- In:       line - a string containing the line from which a word is
  --           desired to be noted.  The string is considered to be of
  --           length (SIZE line).
  -- In/Out:   ptr - on entry: the index of line from which to start the search
  --           for a word, i.e. the search begins at line[ ptr ] with ptr
  --           increasing.  On exit: if ok is FALSE on entry, then unchanged;
  --           if ok is TRUE on entry, then is either the index of the space
  --           or tab immediately after the found word or is >= (SIZE line)
  --           ( where it is only ever greater than if it was passed in as
  --           such ), whether ok is subsequently set to FALSE or not.
  -- Out:      len - if ok is FALSE on entry, then 0; if ok is TRUE on entry
  --           then gives the length of the first word found after the given
  --           starting position ( whether ok is subsequently set to FALSE or
  --           not ), which in the case of no word found is 0
  -- Out:      word - if ok is FALSE on entry, then undefined; if ok is TRUE
  --           on entry, then this contains the found word from line in the
  --           first len bytes, the remaining bytes being undefined, or if
  --           not large enough to contain the word or no word found it is
  --           undefined and ok is set to FALSE, though len gives the correct
  --           length of the word found
  -- In/Out:   ok - on entry: if FALSE, then len is set to 0, ptr and ok
  --           remain unchanged, and word is undefined; otherwise a search
  --           for a word is carried out.  On exit: if FALSE on entry then
  --           FALSE; if TRUE on entry: FALSE if no word found; FALSE if a
```

```
  --           word found that was too large to fit into word; otherwise TRUE
  -- Notes:    Leading spaces and ( horizontal ) tabs ( from line[ ptr ], ptr
  --           on entry ) are skipped.
  --           A word continues until a space or tab or the end of the string
  --           is encountered.

PROC next.int.from.line (VAL []BYTE line, INT ptr, number, BOOL ok)

  -- Purpose:  To determine the next integer in a given line, skipping leading
  --           spaces and tabs.
  -- In:       line - a string containing the line from which an integer is
  --           desired to be noted.  The string is considered to be of
  --           length (SIZE line).
  -- In/Out:   ptr - on entry: the index of line from which to start the search
  --           for an integer, i.e. the search begins at line[ ptr ] with ptr
  --           increasing.  On exit: if ok is FALSE on entry, then unchanged;
  --           if ok is TRUE on entry, then is either the index of the space
  --           or tab immediately after the found integer or is >= (SIZE line)
  --           ( where it is only ever greater than if it was passed in as
  --           such ), whether ok is subsequently set to FALSE or not.
  -- Out:      number - if ok is FALSE on entry, then undefined; if ok is
  --           TRUE on entry: if ok TRUE on exit, the integer read,
  --           otherwise, undefined
  -- In/Out:   ok - on entry: if FALSE, ptr and ok remain unchanged, and
  --           number is undefined; otherwise a search for an integer is
  --           carried out.  On exit: if FALSE on entry then FALSE; if TRUE
  --           on entry: FALSE if there were no non-space or non-tab characters
  --           before the end of the string; FALSE if the first sequence of
  --           non-space, non-tab characters do not form an integer; FALSE if
  --           an integer found that overflowed the range of INT; otherwise TRUE
  -- Notes:    Leading spaces and ( horizontal ) tabs ( from line[ ptr ], ptr
  --           on entry ) are skipped.
  --           The first sequence of characters found after skipping spaces and
  --           tabs is taken to be the integer desired; the integer continues
  --           until a space or tab or the end of the string is encountered.
  --           A + or - are permissible as the first character of the integer.
```

15 xlink.lib

5 entry points

```
PROC Reinitialise (CHAN OF ANY c)

  -- Purpose:  To re-initialise a given channel.
  -- Channels: c - the channel to re-initialise
  -- Notes:    If c is a channel on a hardware link then that link's
  --           hardware is reset.
  --           If this routine is used on a channel on which communication
  --           is not finished then the error flag is set, and subsequent
  --           behaviour is undefined.
```

```
PROC InputOrFail.t (CHAN OF ANY c, []BYTE mess, TIMER t,
                    VAL INT time, BOOL aborted)

  -- Purpose:  To provide, through a time-out, for communication
  --           failure on a channel expecting input.
  -- Channels: c - the channel over which an input communication is
  --           expected
  -- Out:      mess - if aborted is FALSE, the received message over
  --           channel c; otherwise undefined
  -- In/Out:   t - a timer providing a clock to use for the time-out
  -- In:       time - the absolute time of when the time-out for the
  --           communication over channel c should occur
  -- Out:      aborted - TRUE if the communication timed-out; FALSE if
  --           the communication successfully took place.  One cannot be
  --           sure that the communication was not successful if aborted
  --           is TRUE, because it is just possible that the communication
  --           terminated successfully between when the time-out occurred
  --           and the resetting of the channel.
  -- Notes:    If the time-out occurs then the channel c is reset and this
  --           procedure terminates.

PROC InputOrFail.c (CHAN OF ANY c, []BYTE mess, CHAN OF INT kill, BOOL aborted)

  -- Purpose:  To provide, through an abort control channel, for
  --           communication failure on a channel expecting input.
  -- Channels: c - the channel over which an input communication is
  --           expected
  --           kill - an abort control channel: any integer received on
  --           this channel will cause the channel c to be reset and this
  --           procedure to terminate
  -- Out:      mess - if aborted is FALSE, the received message over
  --           channel c; otherwise undefined
  -- Out:      aborted - TRUE if the communication was signalled to be
  --           aborted; FALSE if the communication successfully took place.
  --           One cannot be sure that the communication was not successful
  --           if aborted is TRUE, because it is just possible that the
  --           communication terminated successfully between when the
  --           abort signal was received and the resetting of the channel.

PROC OutputOrFail.t (CHAN OF ANY c, VAL []BYTE mess, TIMER t,
                     VAL INT time, BOOL aborted)

  -- Purpose:  To provide, through a time-out, for communication
  --           failure on a channel expecting to output.
  -- Channels: c - the channel over which an output communication is
  --           to be made
  -- Out:      mess - the message to output over channel c
  -- In/Out:   t - a timer providing a clock to use for the time-out
  -- In:       time - the absolute time of when the time-out for the
  --           communication over channel c should occur
  -- Out:      aborted - TRUE if the communication timed-out; FALSE if
  --           the communication successfully took place.  One cannot be
  --           sure that the communication was not successful if aborted
```

```
  --           is TRUE, because it is just possible that the communication
  --           terminated successfully between when the time-out occurred
  --           and the resetting of the channel.
  -- Notes:    If the time-out occurs then the channel c is reset and this
  --           procedure terminates.

PROC OutputOrFail.c (CHAN OF ANY c, VAL []BYTE mess, CHAN OF INT kill,
                     BOOL aborted)

  -- Purpose:  To provide, through an abort control channel, for
  --           communication failure on a channel expecting to output.
  -- Channels: c - the channel over which an output communication is
  --           to be made
  --           kill - an abort control channel: any integer received on
  --           this channel will cause the channel c to be reset and this
  --           procedure to terminate
  -- In:       mess - the message to be output over channel c
  -- Out:      aborted - TRUE if the communication was signalled to be
  --           aborted; FALSE if the communication successfully took place.
  --           One cannot be sure that the communication was not successful
  --           if aborted is TRUE, because it is just possible that the
  --           communication terminated successfully between when the
  --           abort signal was received and the resetting of the channel.
```

16 streamco.lib

53 entry points

This library provides a simplified interface to **streamio.lib**. It is intended to help TDS users to port their programs to the toolset.

Entry Point	Where Documented
keystream.to.screen	Transputer Development System, Chapter 14
write.len.string	”
write.full.string	”
newline	”
read.echo.char	”
read.echo.hex.int	”
read.echo.int	”
read.echo.text.line	”
read.echo.int64	”
read.echo.hex.int64	”
get.real.with.del	”
read.echo.real32	”
read.echo.real64	”
read.hex.int	”
read.hex.int64	”
scrstream.to.array	”
scrstream.from.array	”
scrstream.fan.out	”

scrstream.sink	"
scrstream.copy	"
keystream.sink	"
read.char	"
write.char	"
write.text.line	"
write.endstream	"
read.int	"
read.int64	"
read.real32	"
read.real64	"
write.int	"
write.hex.int	"
write.int64	"
write.hex.int64	"
write.real32	"
write.real64	"
goto.xy	"
clear.eol	"
clear.eos	"
beep	"
up	"
down	"
left	"
right	"
insert.char	"
delete.chr	"
delete.chl	"
ins.line	"
del.line	"
read.line	see below
scrstream.to.file	"
keystream.from.file	"
scrstream.to.ANSI	"
scrstream.to.TVI920	"

```
PROC read.line (CHAN OF INT source, INT len, []BYTE line, INT char)

  -- As ks.read.line in streamio.lib


PROC scrstream.to.file (CHAN OF ANY scrn, CHAN OF SP fs, ts,
                        VAL[]BYTE filename, INT result)

  -- As so.scrstream.to.file in streamio.lib with the result being
  -- returned as an INT


PROC keystream.from.file (CHAN OF SP fs, ts, CHAN OF INT kbd,
                          VAL []BYTE filename, INT result)

  -- As so.keystream.from.file in streamio.lib with the result being
  -- returned as an INT


PROC scrstream.to.ANSI (CHAN OF ANY scrn, CHAN OF SP fs, ts)
```

```
  -- As so.scrstream.to.ANSI in streamio.lib

PROC scrstream.to.TVI920 (CHAN OF ANY scrn, CHAN OF SP fs, ts)

  -- As so.scrstream.to.TVI920 in streamio.lib
```

17 Index

Entry Point	Library	Page
`write.real32`	streamco	63
`write.real64`	streamco	63
`write.text.line`	streamco	63