1.6: Streamio library

Tony Debling, SGS-Thomson Microelectronics Ltd, 1000 Aztec West, Almondsbury, Bristol BS12 4SQ, England. June 7, 1995.
Converted to HTML via RTFtoHTML, PERL5 and by hand by Dave Beckett <*D.J.Beckett@ukc.ac.uk*>.

WARNING: This has been converted from RTF format and may have mistakes compared to the printed version. If you find any in this file, please send them to me, and I will update it.

Dave Beckett <*D.J.Beckett@ukc.ac.uk*>

Library: streamio.lib

The streamio library contains routines for reading and writing to files and to the terminal at a higher level of abstraction than the hostio library. The file `streamio.inc` defines the `KS` and `SS` protocols and constants used by the streamio library routines. The result value from many of the routines in this library can take a value . `spr.operation.failed` which is a server dependent failure result. It has been left open with the use of . because future server implementations may give more failure information back via this byte. Names for result values can be found in the file `hostio.inc`.

The streamio routines can be classified into three main groups:

- Stream processes
- Stream input procedures
- Stream output procedures.

Stream input and output procedures are used to input and output characters in keystream `KS` and screen stream `SS` protocols. `KS` and `SS` protocols must be converted to the server protocol before communicating with the host.

Stream processes convert streams from keyboard or screen protocol to the server protocol `SP` or to related data structures. They are used to transfer data from the stream input and output routines to the host. Stream processes can be run as parallel processes serving stream input and output routines called in sequential code. For example, the following code clears the screen of a terminal supporting ANSI escape sequences:

```
CHAN OF SS scrn :
PAR
  so.scrstream.to.ANSI(fs, ts, scrn)
  SEQ
    ss.goto.xy(scrn, 0, 0)
    ss.clear.eos(scrn)
    ss.write.endstream(scrn)
```

The key stream and screen stream protocols are identical to those used in the IMS D700 Transputer Development System (TDS) and facilitate the porting of programs between the TDS and the toolset.

1.6.1: Naming conventions

Procedure names always begin with a prefix derived from the first parameter. Stream processes, where the **SP** channel (listed first) is used in combination with either the **KS** or **SS** protocols, are prefixed with `**so.**'. Stream input routines, which use only the **KS** protocol are prefixed with `**ks.**', and stream output routines, which use only the **SS** protocol, are prefixed with `**ss.**'. The **KS** to **SS** conversion routine, which actually uses both protocols, is prefixed for convenience with `**ks.**'.

1.6.2: Stream processes

Procedure	Parameter Specifiers
so.keystream.from.kbd	CHAN OF SP fs, ts,
CHAN OF KS	
	keys.out,
CHAN OF BOOL stopper,	
VAL INT	
	ticks.per.poll
so.keystream.from.file	CHAN OF SP fs, ts,
CHAN OF KS	
	keys.out,
VAL []BYTE filename,	
BYTE	
	result
so.keystream.from.stdin	CHAN OF SP fs, ts,
CHAN OF KS	
	keys.out,
BYTE result	
ks.keystream.sink	CHAN OF KS keys
ks.keystream.to.scrstream	CHAN OF KS keyboard,
CHAN OF SS scrn	
ss.scrstream.sink	CHAN OF SS scrn
so.scrstream.to.file	CHAN OF SP fs, ts,
CHAN OF SS scrn,	
VAL	
	[]BYTE filename,
BYTE result	
so.scrstream.to.stdout	CHAN OF SP fs, ts,
CHAN OF SS	
	scrn,
BYTE result	
ss.scrstream.to.array	CHAN OF SS scrn,
[]BYTE buffer	
ss.scrstream.from.array	CHAN OF SS scrn,
VAL []BYTE buffer	
ss.scrstream.fan.out	CHAN OF SS
	scrn,
screen.out1,	
screen.out2	
ss.scrstream.copy	CHAN OF SS scrn.in, scrn.out
so.scrstream.to.ANSI	CHAN OF SP fs, ts,
CHAN OF SS scrn	
so.scrstream.to.TVI920	CHAN OF SP fs, ts,
CHAN OF SS scrn	
ss.scrstream.multiplexor	[]CHAN OF SS screen.in,
CHAN OF SS	
	screen.out,
CHAN OF INT stopper	

Procedure definitions

so.keystream.from.kbd

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

Reads characters from the keyboard and outputs them one at a time as integers on the channel **keys.out**. It is terminated by sending either a **TRUE** or **FALSE** on the boolean channel **stopper**. The procedure polls the keyboard at an interval determined by the value of **ticks.per.poll**, in transputer clock cycles, unless keys are available, in which case they are read at full speed. It is an error if **ticks.per.poll** is less than or equal to zero. After **FALSE** is sent on the channel **stopper** the procedure sends the negative value **ft.terminated** on **keys.out**.

so.keystream.from.file

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

Reads lines from the specified text file and outputs them on **keys.out**. Terminates automatically on error or when it has reached the end of the file and all the characters have been output on the **keys.out** channel. A `**c**' is output to terminate a text line. The negative value **ft.terminated** is sent on the channel **keys.out** to mark the end of the file. The result returned can take any of the following values:

```
spr.ok                      The operation was successful.
spr.bad.packet.size         Filename too large i.e. SIZE filename
                            > sp.max.openname.size.
spr.bad.name                Null file name.
wspr.operation.failed       The open failed or reading the file
                            failed. If result >= spr.operation.failed
                            then this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual.)
```

so.keystream.from.stdin

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

As **so.keystream.from.file**, but reads from the standard input stream. The standard input stream is normally assigned to the keyboard, but can be redirected by the host operating system. End of file from keyboard will terminate this routine. The result returned may take any of the following values:

```
spr.ok                      The operation was successful.
wspr.operation.failed       Reading standard input failed. If
                            result w spr.operation.failed then
                            this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual.)
```

ks.keystream.sink

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

Reads word length quantities until **ft.terminated** is received, then terminates.

ks.keystream.to.scrstream

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

Converts key stream protocol to screen stream protocol. The value **ft.terminated** on **keyboard** terminates the procedure.

ss.scrstream.sink

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

Reads screen stream protocol and ignores it except for the stream terminator from **ss.write.endstream** which terminates the procedure.

so.scrstream.to.file

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

Creates a new file with the specified name and writes the data sent on channel **scrn** to it. The **scrn** channel uses the screen stream protocol which is used by all the stream output library routines (and is the same as the INMOS TDS screen stream protocol). It terminates on receipt of the stream terminator from **ss.write.endstream**, or on an error condition. The result returned can take any of the following values:

```
spr.ok                      The data sent on scrn was successfully
                            written to the file.
spr.bad.packet.size         Filename too large i.e. SIZE filename
                            > sp.max.openname.size.
spr.bad.name                Null file name.
wspr.operation.failed       If result >= spr.operation.failed then
                            this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual.)
```

If used in conjunction with **so.scrstream.fan.out** this procedure may be used to file a copy of everything sent to the screen.

so.scrstream.to.stdout

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

Performs the same operation as **so.scrstream.to.file**, but writes to the standard output stream. The standard output stream goes to the screen, but can be redirected to a file by the host operating system. The result returned can take any of the following values:

```
spr.ok                      The data sent on scrn was successfully
                            written to standard output.
wspr.operation.failed       If result >= spr.operation.failed then
                            this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual.)
```

ss.scrstream.to.array

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

Buffers a screen stream whose total size does not exceed the capacity of `buffer`, for debugging purposes or subsequent onward transmission using `so.scrstream.from.array`. The procedure terminates on receipt of the stream terminator from `ss.write.endstream`.

ss.scrstream.from.array

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

Regenerates a screen stream buffered in `buffer` by a previous call of `so.scrstream.to.array`. Terminates when all buffered data has been sent.

ss.scrstream.fan.out

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

Sends copies of everything received on the input channel `scrn` to two output channels. The procedure terminates on receipt of the stream terminator from `ss.write.endstream` without passing on the terminator.

ss.scrstream.copy

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

Copies screen stream protocol input on `scrn.in` to `scrn.out`. Terminates on receipt of the end-stream terminator from `ss.write.endstream`, which is not passed on.

so.scrstream.to.ANSI

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

Converts screen stream protocol into a stream of `BYTE`s according to the requirements of ANSI terminal screen protocol. Not all of the screen stream commands are supported. The following tags are ignored:

```
st.ins.char  st.reset  st.terminatest.help  st.claim
st.key.raw  st.key.cooked  st.release  st.initialise
```

The procedure terminates on receipt of the stream terminator from `ss.write.endstream`.

so.scrstream.to.TVI920

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

Converts screen stream protocol into a stream of `BYTE`s according to the requirements of TVI-920 (and compatible) terminals. Not all of the screen stream commands are supported. The following tags are ignored:

```
st.reset st.terminate st.help st.initialise st.key.raw st.key.cooked
```

```
st.release st.claim
```

The procedure terminates on receipt of the stream terminator from `ss.write.endstream`.

ss.scrstream.multiplexor

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

This procedure multiplexes up to 256 screen stream channels onto a single screen stream channel. 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. The procedure is terminated by the receipt of any integer on the channel `stopper`. For *n* channels, each channel is guaranteed to be able to pass on a message for every *n* messages that pass through the multiplexor. This is achieved by cycling from the lowest index of `screen.in`. However, `stopper` always has highest priority.

1.6.3: Stream input

These routines read characters and strings from the input stream, in `KS` protocol.

Procedure	Parameter Specifiers
ks.read.char	CHAN OF KS source, INT char
ks.read.line	CHAN OF KS source, INT len,
[]BYTE	
	line, INT char
ks.read.int	CHAN OF KS source,
INT number, char	
ks.read.int64	CHAN OF KS source,
INT64 number, INT	
	char
ks.read.real32	CHAN OF KS source,
REAL32 number,	
	INT char
ks.read.real64	CHAN OF KS source,
REAL64 number,	
	INT char

Procedure definitions

ks.read.char

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

Returns in `char` the next word length quantity from `source`.

ks.read.line

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

Reads text into the array `line` up to but excluding `*c', or up to and excluding any error code. Any `*n' encountered is thrown away. `len` gives the number of characters in `line`. If there is an error its code is returned as `char`, otherwise the value of `char` will be `INT `*c'. If the array is filled before a `*c' is encountered all further characters are ignored.

ks.read.int

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

Skips input up to a digit, **#**, **+** or **-**, then reads a sequence of digits to the first non-digit, returned as **char**, and converts the digits to an integer in **number**. **char** must be initialized to the first character of the input. If the first significant character is a `**#**' then a hexadecimal number is input, thereby allowing the user the option of which number base to use. The hexadecimal may be in upper or lower case. **char** is returned as **ft.number.error** if the number overflows the **INT** range.

ks.read.int64

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

As **ks.read.int**, but for 64-bit integers.

ks.read.real32

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

Skips input up to a digit, **+** or **-**, then reads a sequence of digits with optional decimal point and exponent) up to the first invalid character, returned as **char**. Converts the digits to a floating point value in **number**. **char** must be initialized to the first character of the input. If there is an error in the syntax of the real, if it is +/- infinity, or if more than 24 characters read then **char** is returned as **ft.number.error**.

ks.read.real64

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

As **ks.read.real32**, but for 64-bit real numbers. Allows for reading up to 30 characters.

1.6.4: Stream output

These routines write text, numbers and screen control codes to an output stream in ss protocol.

```
Procedure                   Parameter Specifiers
ss.write.char               CHAN OF SS scrn, VAL BYTE char
ss.write.nl                 CHAN OF SS scrn
ss.write.string             CHAN OF SS scrn, VAL []BYTE str
ss.write.endstream          CHAN OF SS scrn
ss.write.text.line          CHAN OF SS scrn, VAL []BYTE str
ss.write.int                CHAN OF SS scrn,
VAL INT number, width
ss.write.int64              CHAN OF SS scrn, VAL INT64 number,
VAL INT width
ss.write.hex.int            CHAN OF SS scrn,
VAL INT number, width
ss.write.hex.int64          CHAN OF SS scrn, VAL INT64 number,
VAL INT width
ss.write.real32             CHAN OF SS scrn, VAL REAL32 number,
VAL INT Ip,
                            Dp
ss.write.real64             CHAN OF SS scrn, VAL REAL64 number,
VAL INT Ip,
```

```
                              Dp
ss.goto.xy                    CHAN OF SS scrn, VAL INT x, y
ss.clear.eol                  CHAN OF SS scrn
ss.clear.eos                  CHAN OF SS scrn
ss.beep                       CHAN OF SS scrn
ss.up                         CHAN OF SS scrn
ss.down                       CHAN OF SS scrn
ss.left                       CHAN OF SS scrn
ss.right                      CHAN OF SS scrn
ss.insert.char                CHAN OF SS scrn, VAL BYTE ch
ss.delete.chr                 CHAN OF SS scrn
ss.delete.chl                 CHAN OF SS scrn
ss.ins.line                   CHAN OF SS scrn
ss.del.line                   CHAN OF SS scrn
```

Procedure definitions

ss.write.char

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

Sends the ASCII value **char** on **scrn**, in **scrstream** protocol, to the current position in the output line.

ss.write.nl

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

Sends "***c*n**" to **scrn**.

ss.write.string

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

Sends all characters in **str** to **scrn**.

ss.write.endstream

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

Sends a special stream terminator value to **scrn**.

ss.write.text.line

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

Sends all of **str** to **scrn** ensuring that, whether or not the last character of **str** is `***c**', the last two characters sent are "***c*n**".

ss.write.int

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

Converts **number** into a sequence of ASCII decimal digits padded out with leading spaces and an optional sign to the specified field width, **width**, if necessary. If the number cannot be represented in **width** characters it is widened as necessary; a zero

value for `width` will give minimum width. The converted number is sent to `scrn`. A negative value for `width` is an error.

ss.write.int64

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

As `ss.write.int` but for 64-bit integers.

ss.write.hex.int

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

Converts number into a sequence of ASCII hexadecimal digits, using upper case letters, preceded by `#'. The total number of characters sent is always `width + 1`, padding out with `0' or `F' on the left if necessary. The number is truncated at the left if the field is too narrow, thereby allowing the less significant part of any number to be printed. The converted number is sent to `scrn`. A negative value for `width` is an error.

ss.write.hex.int64

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

As `ss.write.hex.int` but for 64-bit integer values.

ss.write.real32

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

Converts `number` into an ASCII string formatted using `Ip` and `Dp`, as described for `REAL32TOSTRING` (see section 1.8). The converted number is sent to `scrn`. If the formatted form of `number` is larger than 24 characters then this procedure acts as an invalid process.

ss.write.real64

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

As for `ss.write.real32` but for 64-bit real values. See section 1.8, `REAL32TOSTRING` for details of the formatting effect of `Ip` and `Dp`. If the formatted form of number is larger than 30 characters then this procedure acts as an invalid process.

ss.goto.xy

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

Sends the cursor to screen position (x,y). The origin (0,0) is at the top left corner of the screen.

ss.clear.eol

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

Clears screen from the cursor position to the end of the current line.

ss.clear.eos

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

Clears screen from the cursor position to the end of the current line and all lines below.

ss.beep

```
PROC ss.beep (CHAN OF SS scrn)
```

Sends a bell code to the terminal.

ss.up

```
PROC ss.up (CHAN OF SS scrn)
```

Sends a command to the terminal to move the cursor one line up the screen.

ss.down

```
PROC ss.down (CHAN OF SS scrn)
```

Sends a command to the terminal to move the cursor one line down the screen.

ss.left

```
PROC ss.left (CHAN OF SS scrn)
```

Sends a command to the terminal to move the cursor one place left.

ss.right

```
PROC ss.right (CHAN OF SS scrn)
```

Sends a command to the terminal to move the cursor one place right.

ss.insert.char

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

Sends a command to the terminal to move the character at the cursor and all those to the right of it one place to the right and inserts `char` at the cursor. The cursor moves one place right.

ss.delete.chr

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

Sends a command to the terminal to delete the character at the cursor and move the rest of the line one place to the left. The cursor does not move.

ss.delete.chl

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

Sends a command to the terminal to delete the character to the left of the cursor and

move the rest of the line one place to the left. The cursor also moves one place left.

ss.ins.line

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

Sends a command to the terminal to move all lines below the current line down one line on the screen, losing the bottom line. The current line becomes blank.

ss.del.line

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

Sends a command to the terminal to delete the current line and move all lines below it up one line. The bottom line becomes blank.