SpWIO
Command-line Interface for SpaceWire

User Manual

SpWIO Version 17
Manual Version 6, Revision 3, subject to change

This program is designed as a means to interact with a SpaceWire interface and/or script a series of data transmissions and to show data received. The program is intermediate between the Graphical User Interface program (SpaceWireUI) and the Application Programmer Interface (API).

SpWIO allows the user to control the transmission of data on multiple ports on one unit or on any number of units including an arbitrary mix of all 4Links' Ethernet based interface products.

For a quick reference sheet, please see the last page.

Revision history

Version 6, Rev 3, Barry Cook and Paul Walker

- Added SRR
- Additional examples for protocol plug-ins.
- Corrected Event reporting parameters
- Formatting of examples made consistent with other recent manuals

Version 5, Rev 1 – Barry Cook, 20081016:

- Added support for protocol plug-ins.
- Added SPG.

Version 4, Rev 0 – Barry Cook, 20070618:

- Added support for Option PG – Packet Generator / Checker.
- Added support for delta TimeTags

Version 3, Rev 0 – Barry Cook, 20061103:

- Added extended support for DSI.

Version 2, Rev 2 – Barry Cook, Paul Walker, 20060503:

- Added description and examples of comments.

Version 2 – Barry Cook, Paul Walker, 20060426: Added explanations of new capabilities including:

- change to '/' as parameter indicator instead of '-' or '/'. Note that this is the only change that may require a change of usage from the user;
- local (individual) control of each link;
- control of viewable tokens;
- delta changes to viewable tokens and time tag placement;
- delta times on synchronization barrier release;
- comprehensive implementation of changes to command-line parameters from within the program;
- inputting (nested) command and/or data files from within the program;
- abbreviating the console output for large packets;
- outputting and inputting binary files;
- adding delay at the end of each line of an input file.

Version 1 – Barry Cook, edited by Paul Walker, 20060303

Contents

Introduction

This program is designed as a means to interact with a SpaceWire interface and/or script a series of data transmissions and to show data received. The program is intermediate between the Graphical User Interface program (SpaceWireUI) and the Application Programmer Interface (API).

SpWIO allows the user to control the transmission of data on multiple ports on one unit or on any number of units including an arbitrary mix of all 4Links' Ethernet based products.

The program

The program is written in Java (for portability) and supplied as a single consolidated file - SpWIO.jar

A Java run-time environment (JRE) is needed - and is almost always found included with current operating systems; it is also available from Sun Microsystems.

Usage is typically:

```
java -jar SpWIO.jar <parameters>
```

where the parameters may be one or more of:

<parameters>	default	Description
/i <file>	console	Take command input from file;
/d <dd>	0 ms	Delay of dd milliseconds on CR for each line of input file;
/l <file>	no log	Keep a record of activity in a log file;
/a <nn>	0	Abbreviate received packet: if log file, dump binary of packet to file;
/q [y][n][t][f]	n	Quiet mode, Yes, No, True, False;
/t <label>	"	Transmit only sequences labelled <label>;
/v [(delta)event-list]	xtabc	Tokens to be viewed/reported to the console and log file
/x [(delta)event-list]		Exceptions to errors / ignore tokens
/f […]	7	Initial flow-control credit and flow control behaviour
/ew [(delta)event-list]		Event/Waveform cause
/es [source(s)]		Event/Waveform source
/m [dnl] [sf]	n	Set link mode [disabled, normal, legacy], [slow,fixed] on all ports;
/s <number>	10Mb/s	Link transmit speed on all ports;
/w [(delta)event-list]	never	When to add a timetag (start-, middle-, end-of-packet);
/u [<name>=]<IP-address>		Specify a unit, and give it a name.
/p <name>		Add a plug-in for protocol <name>

The parameter indicator must be a slash symbol ('`/`'), the hyphen ('`-`') is used for delta changes to parameters.

The parameter indicator may be upper or lower case. There must be a space before the parameter indicator but otherwise the interpreter is very tolerant e.g. '`/s20`' is the same as '`/s 20`' is the same as '`/S20`' etc.

Link control parameters (`/m, /s, /w, /f, /v, /x, /ew, /es`) set values that are valid until the end of the line, or until changed by a later use of the same parameter. These values are used by units specified after them, so that when a unit is specified it takes previously set values for link mode, speed, viewable tokens and timetag settings.

`<IP-address>` may be an explicit number or symbolic. Renaming it allows the control file content to be in terms of the name and independent of the actual IP address.

A typical example might be:

```
java -jar SpWIO.jar /s 50 /u 192.168.0.62
```

This attaches the EtherSpaceLink unit with IP address 192.168.0.62, starting its links in normal mode at 50Mb/s. Control will be from lines of text typed at the 'console' i.e. the command line.

Link parameters

When set on the command line, link parameters will be applied to all links on the unit. Individual link parameters may be set in normal operation be selecting the link and then specifying the parameter values.

Link Mode

Links may be set to:

- '/md' – disabled;
- '/ml' – legacy mode (for use with IEEE1355 or other pre-ECSS devices);
- '/mn' – normal (ECSS) mode.

Select ONE mode of operation.

Additionally, in DSI version 1.3 or greater and SRR version 0.3 or greater, one of two more modes settings may additionally be specified for an active link mode

- '/mns' (or '/mls') – slow mode (for use 2Mb/s links – also specify '/s 2');
- '/mnf' (or '/mlf') – fixed speed mode where the link continues to run at the default start-up speed (10±1Mb/s) irrespective of the overall link speed selected. This may be used to run some links at 10Mb/s while others run at the selected speed.

Link Speed

Link speed may be set to any value supported by the hardware, values are in Mb/s and decimal numbers may be used where appropriate.

Others

Other parameters are dependant on the presence of options and are covered in later, device and option specific, sections – see sections DSI and ESL.

Unit attachment

Each unit (from the EtherSpaceLink family) to be controlled must be specified in its own '/u' entry. The operating mode of the unit is that PREVIOUSLY given by link parameters. The operating mode may be changed from unit to unit, e.g:

```
... /s 50 /u 192.168.0.24 /u 192.168.0.56 /s 100 /u 192.168.0.62
```

results in units at 192.168.0.24 and 192.168.0.56 running at 50Mb/s and the unit at 192.168.0.62 running at 100Mb/s.

The unit may be specified simply by its IP address, which may be a dotted notation address or a symbolic name known to the operating system. It is possible to assign a name to the unit so that the name, rather than the IP address, can be used in command; this allows the command script to be written independently of the actual device/IP-address, e.g

```
... /u UUT=192.168.0.62
```

allows the name 'UUT' to be used to refer to the unit at address 192.168.0.62.

All 4LInks EtherSpaceLink products use a known port number for Ethernet access, but forthcoming software will require the use different port numbers. In these cases the port number can be attached to the IP-address following a colon, e.g

```
192.168.0.62:1234
```

refers to a unit at IP address 192.168.0.62, port number 1234.

Attaching multiple EtherSpaceLink units

Several units can be addressed by the one instance of the SpWIO program. See the section on Mixed Unit Behaviour below.

Console output

Console output may be suppressed if quiet mode is enabled with '`/q y`' (yes) or '`/q t`' (true). Console output will be present if quiet mode is disabled with '`/q n`' (no) or '`/q f`' (false).

Console output may also be redirected to a file or piped to another program.

The parameter `/a nn` (abbreviate received packet – see below for File output) controls the size of received packet that is displayed on the console view.

File output

All output and input data can be stored in a log file if one is specified with the `/l <log_file>` parameter. The contents of this file may subsequently be used as input to this program for selective re-transmission.

Large packets are not conveniently displayed on the screen and so packets longer than a threshold are abbreviated in the console view and in the log file. The parameter `/a nn` (abbreviate received packet) prints only the first nn bytes of a received packet (default is 10). If a log file is specified, packets larger than nn bytes are dumped (binary, no EOP) into files whose names are derived from the log file name, the unit name, port number and a sequence number.

File input

Control lines may be left as the default settings, may be set interactively or, if specified, from a text file. The text file may be accessed via the /i parameter, by redirecting data from a file or by piping text from another program.

Text files can be nested, so that one text file can include /i parameters inputting other files.

It can be useful to delay transmission to allow time for a response before the next transmission. The parameter `/d dd` adds a pause of dd milliseconds at each Carriage Return encountered in the input file.

Active control sequences may be selected with the '`/t`' parameter. It is followed by a label (future: a list of labels) and only sequences preceded by that label will result in any action. This can be used to select, for example, the packets received by a particular port of a particular unit.

Setting command-line parameters from within the SpWIO program

Most of the command-line parameters can be set from within the SpWIO program. The syntax is simply to enclose the parameter(s) and their arguments in parentheses. For example:

```
(/s 399 /m d /m n /i test.txt)
```

would set the speed to 399Mbits/s, disable and then re-enable the SpaceWire links, and then input a test file.

Running another program from the SpWIO program

It may be useful to run programs from within SpWIO. The program can be a batch file or executable, and can include parameters. Syntax is to use the full filename of the program, including its file type extension, followed by parameters in parentheses without a space between the filename extension and the opening bracket. An example is:

```
delay_ns.bat(800)
```

which is described in more detail in the examples of using SpWIO

Delta changes to parameters

The /v and /w options may be explicitly set, or changed with delta commands:

`/w se` sets time tagging to s and e
`/w -e` removes e tags but leaves others as they are
`/w +m` adds m tags but leaves others as they are
`/w+e-m` adds e and removes m

Protocol plug-ins

Protocol plug-ins – dynamically loadable interpreters for specific protocols – can be used to convert raw packet data to/from human readable format. These may be declared on the command line with the command line parameter:

```
/p <name>
```

or at any point within a session with the command:

```
(/p <name>)
```

The plug-in is selected with session commands starting with the protocol name and containing any required parameters (an empty parameter set gives a short summary of usage)

```
<name>( <parameters> )
```

Received packets are submitted to the plug-ins which, if they recognise the format of the packet, interpret the raw data and generate a readable output on the screen.

See the protocol plug-in section for more information and descriptions of current protocol plug-ins.

Exiting the program

The program will exit when it detects an end-of-file in the input stream - a delay of about 1 second is included to allow receipt of any last data.

Console input is terminated by '`ctrl-Z`' followed by 'return'.

'`ctrl-C`' can also be used to end the program - but it is possible that the log file will not be complete.

Control data format

Input, whether from the command line or a file is primarily a sequence of data to be transmitted on a SpaceWire link. Other information has to be provided in some circumstances, for example which port to use on a multi-port unit, or which unit to use.

Selecting a unit / SpaceWire port

`@<name>` selects the unit with name (or IP-address) `<name>` for subsequent data `@<port>` selects the port (a number) for subsequent data

A single unit with a single port requires neither selector.

A single unit with multiple ports requires a port selector.

Multiple units require a unit selector, and possibly also a port selector if the selected unit has multiple ports.

Normal Data

Normal data is a sequence of one or more numeric values and end-of-packet indicators.

By default, a numeric value occupies one byte and its interpretation depends on its format:

```
0oo    Octal number      \
0xhh   Hexadecimal number \
#hh    Hexadecimal number /
dd     Decimal number    /
```

`oo` represents one or more octal digits (0..7)
`hh` represents one or more hexadecimal digits (0..F)
`dd` represents one or more decimal digits (0..9)
`x` is the letter x, lower or upper-case
`#` is the # character

e.g. in the following, each value represents the decimal number 33:

```
041    0x21    #21    33
```

Negative numbers are not supported.

Numbers may be separated by one or more spaces, tabs, newlines, commas, semicolons or dots.

e.g. the following sequences are equivalent, each represents the same 4-bytes:

```
12 34 56 78
12,34,56,78
12, 34, 56, 78
12;34;56;78
12,,34;;;56...78
12.34.56.78
```

A number may include a suffix indicating that it represents more that one byte:

`s`	(lower case)	short-word, 16-bits, 2-bytes little-endian
`S`	(upper case)	short-word, 16-bits, 2-bytes big-endian
`w`	(lower case)	word, 32-bits, 4-bytes little-endian
`W`	(upper case)	word, 32-bits, 4-bytes big-endian

little-endian numbers are transmitted least-significant byte first, big-endian numbers are transmitted most-significant byte first. e.g.

```
#12345678w      -->     #78 #56 #34 #12
#12345678W      -->     #12 #34 #56 #78
#1234s          -->     #34 #12
#1234S          -->     #12 #34
1w              -->     #01 #00 #00 #00
1W              -->     #00 #00 #00 #01
```

ASCII character strings are contained within pairs of single-quote characters, the escape sequence `\'` is used to represent the single-quote character itself. Characters are transmitted from left to right.

> [The double-quote pair currently behaves the same way, but it may be redefined for 16-bit Unicode characters - unless a suffix is used for the purpose - TBD.]

Packets are terminated by `EOP` (or `eop`) for normal end-of-packet or `EEP` (or `eep`) for error-end-of-packet.

Example packet:

```
#12 #34 192.168.0.56 1077W 1063W 7 8 9 eop
```

Comments

Comments may be added either on the command line or in an input file.

`//` is used as a delimiter for a comment thereafter until the end of line.

The paired delimiters

`/*` at the start of the comment and, at the end of the comment `*/`

may be used for a comment in the middle of a line or for a comment of several lines.

Labelling and selective transmission

Labels, consisting of a word followed by a colon, may be inserted in the control stream. Transmission will only occur after a label is recognised and will cease on encountering another, different, label. This allows multiple sequences to be stored in a file and different sequences to be selected.

One significant use of this is that a log file will contain both transmitted and received data sequences (labelled "Tx:" and "Rx:", respectively). This program can be used to replay either the Tx or the Rx sequence separately:

```
... /i logfile /t Tx
... /i logfile /t Rx
```

Displayed data format

Data transmitted will appear as lines in the form:

```
Tx:@<unit>@<port> #<value> ...
```

All values will be presented as hexadecimal bytes, regardless of the format of the original request. This allows the precise sequence of data sent to be seen.

If only one unit has been attached, the `@<unit>` indicator will not appear.

Data received will be shown similarly, but marked Rx:

```
Rx:@<unit>@<port> #<value> ...
```

The same format is used in the log file, if there is one.

ESL: Ethernet to SpaceWire Link

Option TT TimeTags

This is not supported in this version of SpWIO.

Timetags are controlled by the /w (when) parameter. By default, all timetags are disabled. Its argument is one or more of:

- 's' – the start of a packet;
- 'm' – the middle bytes of a packet;
- 'e' – the end of a packet (EOP or EEP).

Zero or more qualifiers may be used (e.g. '/wse' (or '/wes') selects timetags on the start and end of packets) and delta values may be specified. (e.g. '/w+m' also selects timetags on the middle bytes of packets)

Option TC Time Code Generator

This is not supported in this version of SpWIO.

Option SF Store and Forward

This is not supported in this version of SpWIO.

Option ER Error Reporting

This is not supported in this version of SpWIO.

Option EW Error Waveform

This is not supported in this version of SpWIO.

DSI: Diagnostic SpaceWire Interface

Timing of data arriving over a network cannot be guaranteed but the 4Links Diagnostic SpaceWire Interface, DSI, provides mechanisms to overcome this issue.

Store and forward

To ensure that all the data required to be sent together have arrived at the unit before transmission begins it can be temporarily stored in a buffer and then released when it has all arrived. The buffer for each output port is 1000 bytes long but this must include control as well as data bytes. Data to be grouped in this way is bracketed by `[` and `]` e.g.

```
[ 1 2 3 4 eop ]
```

Option ER Event Reporting

Only data, EOP and EEP tokens are normally displayed. Link control and error tokens are normally hidden but can be made visible with the `/v` command. Error conditions, if they occur mid-packet are normally replaced by EEP but if made visible are seen as they are and no EEP is substituted. Possible arguments are:

- `'a'` – ESC-ESC;
- `'b'` – ESC-EOP;
- `'c'` – ESC-EEP;
- `'p'` – parity error;
- `'t'` – time code;
- `'f'` – flow control token (except for the first flow-control token received when a link starts);
- `'x'` – timeout.

Zero or more qualifiers may be used (e.g. `'/vptf'` selects visible parity errors, time-codes and flow-control tokens) and delta values may be specified. (e.g. `'/v+x'` also selects timeouts)

Option SO Synchronized output

Synchronization in the SpWIO program provides a convenient way to access the DSI's mechanism whereby outputs on different ports can synchronized so that data can be transmitted at the same time. One to all the ports may be synchronized.

We use the techniques developed for low/zero jitter time codes to synchronize a bit edge to the same clock edge across all ports involved. The skew between ports is just that caused by wiring and propagation delay differences and is typically well below 1ns.

The process of synchronization is similar to a race in which those wishing to compete first join the list of competitors. Each then moves to a starting barrier and as soon as all are assembled the barrier is lifted and everyone starts at the same time.

Multiple port outputs can be closely synchronized with a barrier synchronization and uses the control characters `+` (to join a synchronization), `-` (to resign from a synchronization) and `|` (the barrier itself).

Delays, during which the D and S lines of the SpaceWire link are held unchanging can be inserted with one or more `=` characters. Each `=` inserts 4 (may change to 2 at some time in the future) transmit bit periods of unchanging D and S. This is especially useful to introduce controlled skew after a barrier synchronization but may also be used for other purposes, e.g. to test a link's timeout period.

Example of a closely controlled timing sequence:

```
@1+@2+@3+@4+[|4 4 4 eop]@3[|3 3 3 eop]@2[|2 2 2 eop]@1[|==1 1 1 eop]
```

Port 1 will output 1 1 1 EOP

Port 2 will output 2 2 2 EOP
Port 3 will output 3 3 3 EOP
Port 4 will output 4 4 4 EOP

Each of the packets will start at the same time, except that the output from port 1 will be delayed by eight transmit bit-times.

Option EI Error Insertion

It is possible to insert explicit tokens that would not normally form part of a correctly formed data stream.

`FCT` inserts a flow-control character, normally only inserted by the SpaceWire state-machine.
`ESC` inserts an escape character. Not all insertions of ESC form errors:

`ESC ESC`	error
`ESC EOP`	error
`ESC EEP`	error
`ESC FCT`	acceptable as this is a NULL token
`ESC data`	acceptable as this is a time-code

`~` inverts the parity bit and should cause an error on the link. There is no explicit facility to change an arbitrary bit since this can be achieved by changing a data value combined with inverting the parity bit.

Each `=` inserts a delay of 4 (may change to 2 at some time in the future) transmit bit periods- excessive delay should cause a timeout on the link.

The SpaceWire state-machine can be controlled with the `/f` command. Its qualifiers allow control of the number of flow-control tokens issued at link start and subsequent behaviour with regard to flow-control. Qualifiers are:

- `nn` – a decimal number between 0 and 15 – the number of FCT's to send when a link starts;
- `'.'` – don't automatically send any more FCT's;
- `'i'` – ignore flow control credit – always send data whether credit allows this or not.

ECSS operation requires the transmission of 1 to 7 FCT's at link start. No FCT should result in the link not starting and more than 7 is an ECSS error.

The `'.'` qualifier allows the user to control the amount of FCT credit by explicitly sending all FCT's after the link has started.

The `'i'` qualifier allows the user to send data even when there is no credit to do so – in order to test the link behaviour (ECSS requires it to reset).

Received tokens and sequences that normally force link reset (in accordance with ECSS requirements) can be ignored by the SpaceWire state machine. The `/x` command specifies which are to be ignored:

- `'a'` – ESC-ESC;
- `'b'` – ESC-EOP;
- `'c'` – ESC-EEP;
- `'p'` – parity error;
- `'v'` – excess data – more data is received than credit was given;
- `'w'` – excess FCT – more than 56-bytes of credit has been received;
- `'x'` – timeout.

Zero or more qualifiers may be used (e.g. `'/xp'` ignores parity errors) and delta values may be specified. (e.g. `'/x+t'` also ignores timeouts)

Option EW Events / Waveforms

Each SpaceWire link is monitored by a waveform capture unit that may be triggered on a number of events. Triggering may be from events on the associated link or on other links or external events.

Each link can generate an event which is an event placed in the transmit data stream OR one or more of the events selected with the /ew command whose qualifiers are:

- 'a' – ESC-ESC;
- 'b' – ESC-EOP;
- 'c' – ESC-EEP;
- 'd' – 'nchars' – data, EOP and EEP;
- 'e' – EOP;
- 'f' – FCT;
- 'g' – EEP;
- 'i' – first null, received when a link starts;'
- 'm' – mid character in packet;
- 'p' – parity error;
- 's' – start of packet;
- 't' – time codes;
- 'v' – excess data – more data is received than credit was given;
- 'w' – excess FCT – more than 56-bytes of credit has been received;
- 'x' – timeout.

An event may be placed in the transmit data stream so that an event is generated when this point in the stream comes to be transmitted. This may be used to trigger a waveform capture on the transmit data in addition to the normal triggers on received data. Use the keyword EVENT in the transmit data stream.

Each link generates a trigger based on the above selection. Each waveform capture is initiated by a logical OR of triggers selected with the /es command whose qualifiers are:

- '1' – trigger from link 1;
- '2' – trigger from link 2;
- '3' – trigger from link 3;
- '4' – trigger from link 4;
- '5' – trigger from link 5;
- '6' – trigger from link 6;
- '7' – trigger from link 7;
- '8' – trigger from link 8;
- 'A' – trigger from rising edge on SMA 1-2;
- 'B' – trigger from rising edge on SMA 3-4;
- 'C' – trigger from rising edge on SMA 5-6;
- 'D' – trigger from rising edge on SMA 7-8;
- '0' – trigger when the synchronising barrier is lifted.

SMA inputs (on the rear panel) are (currently) set with a detection threshold of 0.5 volts.

It is possible to trigger more than one waveform capture on a single event. For example, an event on one link can be set to capture waveforms on all links.

Option TT TimeTags

Time tags are controlled by the /w (when) parameter. By default, all time tags are disabled. Its qualifier is one or more of:

- 's' – the start of a packet;
- 'm' – the middle bytes of a packet;
- 'e' – the end of a packet (EOP);
- 'g' – the error end of a packet (EEP);
- 'p' – parity error;
- 't' – time code;
- 'f' – flow control token;
- 'x' – timeout;
- 'b' – ESC-EOP;
- 'c' – ESC-EEP;
- 'a' – ESC-ESC;
- 'z' – Enable delta time tags (see below).

Zero or more qualifiers may be used (e.g. '/wse' (or '/wes') selects time tags on the start and end of packets) and delta values may be specified. (e.g. '/w+m' also selects time tags on the middle bytes of packets)

Delta Time Tags (DSI version 1.0 or later)

When enabled, a timetag that is reported soon after another timetag is sent as a delta time – this reduces the amount of data that needs to be transferred back to the host.

A delta time tag can be reported if the time interval is less than ~6.5µs when a 3-byte code is used in place of the 9-byte code used for an absolute time tag.

SPG: SpaceWire Packet Generator and Packet Checker

custom-designed processor that handles packet data at line speed can be inserted into the data-path in each direction – Packet Generator in the direction sending to the SpaceWire link and Packet Checker in the direction from the link.

General usage

Most often the generator and checker will use pre-defined programs. These can be included for compilation with the instruction:

```
PG( (/I program_file_name) )
```

Nothing is sent out of the SpaceWire link at this time. The generator/checker must be started in order to send the specified data:

```
PG( run )
```

For a complete description of the capabilities of the generator/checker and of the programming language used see the companion guide – "PGPC User Manual".

Direct Usage

It is possible to directly include programs for the generator/checker in SpWIO commands. For example, in order to transmit a packet consisting of the values 1 to 5 and an end-of-packet token we could issue the command:

```
PG( Generator { !{ 1 2 3 4 5 EOP } } )
```

The command may be spread over more than one line:

```
PG(
  Generator
    {
      !{ 1 2
         3 4
         5 EOP
       }
    }
  )
```

Nothing is sent out of the SpaceWire link at this time. The generator must be started in order to send the specified data:

```
PG( run )
```

The generator will stop itself at the end of this packet transmission and revert to being transparent – data can then be sent directly to the link, as if the Packet Generator was not present.

The same packet can be resent by issuing another start command. There is no need to re-issue the packet content if it is unchanged. For example:

```
PG( run )
PG( run )
PG( run )
```

sends the packet three more times.

SRR: SpaceWire RMAP Responder

In the SpaceWire RMAP (Remote Memory Access Protocol) Responder, there is an implementation of an RMAP target for each SpaceWire port. Each RMAP target has its own memory area and the user can, via Ethernet, access this memory as well as controlling reporting levels.

General usage

Commands from the user are sent with the SPGRMAP protocol (common to an software implementation of RMAP on SPG) – see the SPGRMAP plug-in description for more information.

By default, the RMAP responder is enabled. It can be disabled with the SpWIO command:

```
PG( stop )
```

And re-started with the command:

```
PG( run )
```

Mixed-unit behaviour

The program can be used with any of the products in the EtherSpaceLink interface family, such as the ESL-RF201, the ESL-RG401/8, the DSI-RG408xms, and also with other products that will be developed in the family. The program can also be used with any number of any mix of these products.

For example, the fragment of a command line:

```
/s 200 /u RF201=192.168.0.15 /s 249 /u RG401_8=192.168.0.59 /u DSI=192.168.1.73
```

could access three separate units, each being a different model from the family:

- An EtherSpaceLink ESL-RF201, set to a transmit speed of 200Mbits/s, at 192.168.0.15
- An EtherSpaceLink ESL-RG401/8, set to a transmit speed of 249Mbits/s, at 192.168.0.59
- A Diagnostic SpaceWire interface, DSI-RG408, also set to a transmit speed of 249Mbits/s, at 192.168.1.73

Not all of 4Links products support all the features that the program provides access to. In the event of any request for action from a unit that does not support that action, the request will be ignored. For example, attempting to transmit data from port 2 on a unit that has only one port will simply result in that data being ignored.

Protocol Plug-ins

Protocol plug-ins are dynamically loadable programs to format clear-text requests into detailed protocol packets for transmission and a corresponding received packet interpreter.

A separate program is used for transmit and receive and thus either or both may be installed.

Plug-ins are loaded by a request, either on the command line when starting SpWIO or in a session command during operation.

```
/p <name>
```

Two files (`ProtocolTx_<name>.class` and `ProtocolRx_<name>.class`) are loaded from the current directory.

The transmit plug-in is called when a command of the form

```
<name>( <parameters> )
```

is seen.

The receive plug-in is offered to all received packets and tests these packets to see if they can be identified as being of the form expected. Recognised packets are interpreted and plain-text descriptions produced for display.

The following sections describe each protocol plug-in in more detail.

Remote Memory Access Protocol (RMAP)

Requesting the protocol without any parameters:

```
Rmap()
```

generates a summary parameter listing.

RMAP ({W(rite) <value value ...> | R(ead) <number-of-bytes>} @ <address> [F(ixed-address)] [P(ath) <address…>] [S(ource-path) <address_bytes>] [T(ransaction-identifier) <n>] [K(ey) <n>] [A(cknowledge)] [V(erify)])

Each parameter may be given as the whole word or may be abbreviated down to as little as a single letter.

The parameters are:

Parameter	Function
W(rite) <value value …> *(Either this or Read is required)*	Create a write data packet containing data values <value value …>
R(ead) <number-of-bytes> *(Either this or Write is required)*	Create a read data request packet for <number-of-bytes>
@ <address> *(Required)*	Specify the start address for the read or write
F(ixed-address) *(Optional)*	The "increment address" flag is set unless this parameter is given
P(ath) <address…> *(Optional)*	Routing path address to the RMAP target. This may be one or more physical and/or logical address bytes. If not specified, a single logical address byte value 254 is used.
S(ource-path) <address-bytes> *(Optional)*	Return routing path address from the RMAP target to this unit. This may be one or more physical and/or logical address bytes. If not specified, a single logical address byte value 254 is used.
T(ransaction-identifier) <n> *(Optional)*	The transaction identifier is set to <n> If not specified, transaction identifiers increment automatically
K(ey) <n> *(Optional)*	The transaction key is set to <n> If not specified, the key is set to zero.
A(cknowledge) *(Optional)*	Request an acknowledgement for write requests. If not specified, not acknowledge is requested.
V(erify) *(Optional)*	Request that write packets have their data verified before the write takes place. If not specified, not verify request is made. (Optional)

SPGRMAP

Requesting the protocol without any parameters:

```
spgrmap()
```

generates a summary parameter listing.

SPGRMAP ({W(rite) <value value ...> | R(ead) <number-of-bytes>} @ <address> [WRL <n>] [RRL <n>] [ERL <n>] [ORL <n>] [TRL <n>]

Each parameter may be given as the whole word or may be abbreviated down to as little as a single letter.

The parameters are:

Parameter	Function
W(rite) <value value …> *(Optional)*	Create a write data packet containing data values <value value …>
R(ead) <number-of-bytes> *(Optional)*	Create a read data request packet for <number-of-bytes>
@ <address> *(Required if Write or Read)*	Specify the start address for the read or write
WRL <n> *(Optional)* *Default is <n> = 0*	Set the RMAP Write Report Level to <n> where <n> can be one of: 0: Don't report RMAP writes; 1: Report each RMAP write; 2: Report each RMAP write with its address and length; 3: Report each RMAP write with its address and length and the data bytes written.
RRL <n> *(Optional)* *Default is <n> = 0*	Set the RMAP Read Report Level to <n> where <n> can be one of: 0: Don't report RMAP reads; 1: Report each RMAP read; 2: Report each RMAP read with its address and length; 3: Report each RMAP read with its address and length and the data bytes written.
ERL <n> *(Optional)* *Default is <n> = 0*	Set the RMAP Error Report Level to <n> where <n> can be one of: 0: Don't report RMAP errors; 1: Report only RMAP header errors; 2: Report all RMAP errors.
ORL <n> *(Optional)* *Default is <n> = 0*	Set the RMAP "Other" Report Level to <n> ("Other" is packets received from SpaceWire other than RMAP packets) where <n> can be one of: 0: Pass all non-RMAP packets to the PC (unchanged); 1: Report each non-RMAP packet received; 2: Discard each non-RMAP packet received.
TRL <n> *(Optional)* *Default is <n> = 0*	Set the RMAP Time-code Report Level to <n> where <n> can be one of: 0: Discard all time codes received from SpaceWire; 1: Report Time-codes as SPGRMAP packets; 2: Pass Time-codes to the PC/user.

Examples of using the SpWIO program

The following log shows a number of examples of using the SpWIO program: to send and receive packets, to use the barrier synchronization of packet transmission, to generate special sequences of characters including errors, to record and replay log files, to change program parameters from within the program, and to call an external program from within SpWIO.

In the examples,

what the user types is `blue console font`
what the program responds is in `red console font`
and explanation is in normal text

Also in the examples, port 1 is connected via a short (30cm) cable to port 2, and port 3 by a similar cable to port 4.

Usage listing

Typing the command to run the program without any parameters generates the usage listing:

```
C:\dsi>java -jar SpWIO.jar

//-------------------------------------------------------------------------------------------//
//  4Links.SpWIO.v13 - Usage: SpWIO <general commands>                                       //
//                                                                                           //
//  Exchange data with one or more EtherSpacelinks.                                          //
//                                                                                           //
//  Input is from the console, unless an input file                                          //
//  is given or input redirection is used.                                                   //
//                                                                                           //
//  Output is to the console, unless output redirection                                      //
//  is used or quiet mode is selected.                                                       //
//-------------------------------------------------------------------------------------------//
//  GENERAL COMMANDS                                                                         //
//                              default                                                      //
//                                                                                           //
//  /i  <file>                  console  Take command input from file                        //
//  /l  <file>                  no log   Keep a record of activity in <file> which must not  //
//                                         already exist                                     //
//  /l  <file>!                          as above - over-writes existing file (if any)       //
//  /l  <file>+                          as above - appends to existing file (if any)        //
//  /q  [y][n][t][f]            n        quiet mode;                                        //
//  /t  <label>                 ''       transmit only sequences labelled <label>           //
//                                                                                           //
//  /m  [dnl] [sf]              n        Set link mode [disabled,normal,legacy] [slow,fixed] //
//  /s  <number>                10Mb/s   Link speed;                                        //
//  /v  [(delta)event_list]     xtabc    View events (otherwise invisible)                  //
//  /w  [(delta)event_list]              When to add a timetag                              //
//  /x  [(delta)event_list]              Ignore events                                      //
//  /ew [(delta)event_list]              Event cause                                        //
//  /es [(delta)trigger_list]            Waveform trigger source                            //
//  /f  [.I0-15]                         Flow control behaviour                             //
//                                                                                           //
//  /u  [<name>=]<IP-address>            Specify a unit and (optional) give it a name       //
//                                                                                           //
//  /d  <ddd>                   0        Delay, at each end-of-line, for <ddd> ms           //
//                                       - useful in input files to allow any response to    //
//                                       arrive before sending the next data                 //
//                                                                                           //
//  /a  <nn>                    0        Abbreviate long received packets, show only the    //
//                                       first nn bytes. A value of 0 results in all bytes   //
//                                       being shown. If a log file has been specified       //
//                                       the complete packet will be stored, in binary,      //
//                                       in a file whose name is derived from the log file   //
//                                       name.                                               //
//                                                                                           //
//  /p  <Name>                           Add a protocol plug-in from one or both of the files //
//                                       ProtocolRx_<Name>.class                             //
//                                       ProtocolTx_<Name>.class                             //
//                                                                                           //
//  list may give explicit settings (e.g. '/w s') or deltas (e.g. '/w+m' or '/w-s' or        //
//   '/w+me-s'etc.)                                                                          //
//                                                                                           //
//  event characters (upper- or lower-case) that may appear in an event_list:                //
//    a  ESC-ESC                                                                             //
```

```
//    b  ESC-EOP                                                                    //
//    c  ESC-EEP                                                                    //
//    d  nchar = data byte                                                          //
//    e  EOP                                                                        //
//    f  FCT                                                                        //
//    g  EEP                                                                        //
//    i  first-NULL                                                                 //
//    m  mid-bytes in packet                                                        //
//    n  NULL                                                                       //
//    p  parity-error                                                               //
//    s  first-byte of packet                                                       //
//    t  tchar = time code                                                          //
//    v  excess data                                                                //
//    w  excess FCT                                                                 //
//    x  timeout                                                                    //
//                                                                                  //
//  trigger_list (letters may be upper- or lower-case):                             //
//    0  Barrier lifting                                                            //
//    1  port 1 event                                                               //
//    2  port 2 event                                                               //
//    3  port 3 event                                                               //
//    4  port 4 event                                                               //
//    5  port 5 event                                                               //
//    6  port 6 event                                                               //
//    7  port 7 event                                                               //
//    8  port 8 event                                                               //
//    A  port SMA_12 rising edge                                                    //
//    B  port SMA_34 rising edge                                                    //
//    C  port SMA_56 rising edge                                                    //
//    D  port SMA_78 rising edge                                                    //
//------------------------------------------------------------------------------------//
```

Running the program with default parameters

The only parameter that must be set on the command line is the Unit IP address. It is normally useful to give the unit a name, and here we use the serial number of the DSI unit that was used for these tests, which is A59. SpWIO responds as follows:

```
C:\dsi>java -jar SpWIO.jar /u A59=192.168.0.59
// 4Links.SpWIO run by 4links on Thu Mar 02 14:43:07 GMT 2006
// Attached "A59" = 192.168.0.59 is DSI-RG408, 8-ports, link mode is normal at 10Mb/s
// Input from console
```

Help

Typing `?<return>` on the command line of the program gives a usage listing for help:

```
?
// ?l for the status etc. of the currently selected link
// ?t for help on transmit commands
// ?r for help on received data
// ?c for help on general commands
```

Taking these in turn,

The link help command:

```
?l
// @4 Link mode is normal (running), Tx buffer is empty, Rx speed is 10Mb/s
// @4 Link Tx speed is 10Mb/s
```

The transmit commands (as in the Quick Reference on the last page):

```
?t
//------------------------------------------------------------------------------//
// TRANSMIT CONTROL / DATA                                                      //
//                                                                              //
// @<unit>            Select unit for subsequent data                           //
// @<port>            Select port for subsequent data                           //
//                                                                              //
// <number>                                                                     //
//     23             Decimal byte                                              //
//    023             Octal byte       (= 19 decimal)                           //
```

```
//    #23              Hexadecimal byte (= 35 decimal)                          //
//   0x23              Hexadecimal byte (= 35 decimal)                                //
//                                                                                    //
// <number>s           (lower case s) Short word (16-bits, 2bytes) Little-endian      //
// <number>S           (upper case S) Short word (16-bits, 2bytes) Big-endian         //
// <number>w           (lower case w) Word       (32-bits, 4bytes) Little-endian      //
// <number>W           (upper case W) Word       (32-bits, 4bytes) Big-endian         //
//                                                                                    //
// 'abcd'              String of bytes, leftmost first; \' represents '               //
//                                                                                    //
// EOP                 End-of-packet                                                  //
// EEP                 Error-end-of-packet                                            //
//                                                                                    //
// ESC                 Escape                                                         //
// FCT                 Flow control token                                             //
//                                                                                    //
// ~                   Invert parity bit                                              //
//                                                                                    //
// [                   Store data temporarily                                         //
// ]                   Forward stored data                                            //
//                                                                                    //
// <                   Not yet implemented                                            //
// >                   Not yet implemented                                            //
//                                                                                    //
// +                   Join a barrier synchronization                                 //
// -                   Resign from a barrier synchronization                          //
// |                   Barrier synchronization, all synchronized ports will transmit  //
//                       at the same time                                             //
//                                                                                    //
// (xxxx)              Interpret xxxx as a command-line string (see ?c) and cause     //
//                     the specified actions to take effect at this place. Can be     //
//                     to change link speed (/s n), include a command file (/i fff),  //
//                     etc.                                                           //
//                                                                                    //
// binary(filename)    Send the (binary) content of the file 'filename'.              //
//                                                                                    //
// program(ppp)        Run the 'program' with parameters ppp and use its output as    //
//                     command input.                                                 //
//                                                                                    //
// <CR>                Return will flush the transmit buffer                          //
//------------------------------------------------------------------------------------//
```

The receive commands (also as in the Quick Reference on the last page):

```
?r
//-------------------------------------------------------------------------------------//
// RECEIVED DATA                                                                       //
//                                                                                     //
// @<unit>              Indicate which unit sourced the subsequent data                //
// @<port>              Indicate which port sourced the subsequent data                //
//                                                                                     //
// <number>                                                                            //
//    #23               Hexadecimal byte (= 35 decimal)                                //
//                                                                                     //
// EOP                  End-of-packet                                                  //
// EEP                  Error-end-of-packet                                            //
//                                                                                     //
// ESC                  Escape                                                         //
// FCT                  Flow control token                                             //
//                                                                                     //
// ESC ESC              Escape followed by escape                                      //
// ESC EOP              Escape followed by end-of-packet                               //
// ESC EEP              Escape followed by error-end-of-packet                         //
//                                                                                     //
// /*PARITY_ERROR*/                                                                   //
// /*TIMEOUT*/                                                                        //
// /*TIMECODE-0:0:00*/                                                                //
// /*123.456s*/         Timetag                                                        //
//-------------------------------------------------------------------------------------//
```

The general commands `?c` are as listed in the usage listing for the program.

Running the program with parameters as required

The command line starts with the java command, followed by parameters. In this example, the link speed is set to 150Mbits/s, the link mode to normal, time tags are set on the start and end of packet. These parameters are then used by the unit which is at IP address 192.168.0.59 and which can be referred to in the program as A59.

The program responds to the command with an expansion of the parameters.

```
C:\dsi>java -jar SpWIO.jar /s 150 /m n /w se /u A59=192.168.0.59
    // 4Links.SpWIO.v13 run by 4links on Thu Mar 02 15:12:06 GMT 2006
    // /s 150 /m n /w se /u A59=192.168.0.59
    // Attached "A59" = 192.168.0.59 is DSI-RG408, 8-ports, link mode is normal at 150Mb/s
    // Input from console
```

Sending individual packets

The following command sends the packet 1 1 1 EOP to port 1.

```
    @1 1 1 1 eop
    Tx:@1 #01 #01 #01 EOP
    Rx:@2 /*21 106.628 478 138 7s*/ #01 #01 #01 /*21 106.628 478 392 0s*/ EOP
```

Notice that the difference between the time tag at the start of packet and the end of packet is over 250ns, which is longer than the 200ns we would expect for a 3-Byte packet at 150Mbits/s. This has probably occurred because the packet starts being sent before all the data is in the buffer. We can ensure that the packet is in the buffer by enclosing the packet in square brackets:

```
    @1 [1 1 1 eop]
    Tx:@1 [ #01 #01 #01 EOP ]
    Rx:@2 /*21 127.513 344 344 0s*/ #01 #01 #01 /*21 127.513 344 544 0s*/ EOP
```

and the packet duration is now 200ns as expected.

Packets can similarly be sent on other ports:

```
    @2 [2 2 2 eop]
    Tx:@2 [ #02 #02 #02 EOP ]
    Rx:@1 /*21 158.106 762 790 7s*/ #02 #02 #02 /*21 158.106 762 990 7s*/ EOP
    @3 [ 3 3 3 eop]
    Tx:@3 [ #03 #03 #03 EOP ]
    Rx:@4 /*21 191.749 682 337 3s*/ #03 #03 #03 /*21 191.749 682 537 3s*/ EOP
    @4 [4 4 4 eop]
    Tx:@4 [ #04 #04 #04 EOP ]
    Rx:@3 /*21 219.678 359 938 7s*/ #04 #04 #04 /*21 219.678 360 138 7s*/ EOP
```

and the packet duration for all these packets is the 200ns.

Terminating the program

The clean way to terminated the program is to type '`ctrl-Z`' followed by '`return`'.

Sending multiple, synchronized, packets

The SO option is required for this operation.

Please note that the minimum transmit speed currently supported by the DSI products for synchronized outputs is 100Mb/s.

These examples were run at a link speed of 100Mbits/s.

First of all, we set up a group of ports that are to be synchronized:

```
@1+ @2+ @3+ @4+
Tx:@1 +
Tx:@2 +
Tx:@3 +
Tx:@4 +
```

[SpWIO is able to synchronize one or more ports on one or more devices – provided those devices all have the SO option and are all synchronized to each other via a suitable cable.]

Then we send a packet to each port of the synchronized set. In this example, for clarity of explanation, the separate packets are each typed on a separate line.

```
@1 [ | 1 2 3 eop ]
Tx:@1 [ | #01 #02 #03 EOP ]
@2 [ | 4 5 6 eop ]
Tx:@2 [ | #04 #05 #06 EOP ]
@3 [ | 7 8 9 eop ]
Tx:@3 [ | #07 #08 #09 EOP ]
@4 [ | 10 11 12 eop ]
Tx:@4 [ | #0A #0B #0C EOP ]
// Barrier lifted at 359.988 834 799 0s
Rx:@1 /*359.988 834 800 0s (Barrier+1.0ns)*/ #04 #05 #06 /*359.988 835 100 0s
    (Barrier+301.0ns)*/ EOP
Rx:@2 /*359.988 834 800 0s (Barrier+1.0ns)*/ #01 #02 #03 /*359.988 835 100 0s
    (Barrier+301.0ns)*/ EOP
Rx:@3 /*359.988 834 800 0s (Barrier+1.0ns)*/ #0A #0B #0C /*359.988 835 100 0s
    (Barrier+301.0ns)*/ EOP
Rx:@4 /*359.988 834 800 0s (Barrier+1.0ns)*/ #07 #08 #09 /*359.988 835 100 0s
    (Barrier+301.0ns)*/ EOP
```

As soon as the last packet arrives at the barrier, the synchronization barrier is released and all the packets are transferred.

The cables between ports 1 and 2, and between ports 3 and 4, are both short, approximately 30cm, and so there is a very short delay between the barrier release and the start of packet. In the above example, all the packets are lined up apparently precisely with each other, and each packet takes 300ns from start to end, corresponding with data characters at 100Mbits/s.

In the following example, again all the packets are lined up precisely with each other, but the sampling quantization gives an apparently longer delay on both start and end of packet. The duration of the three data characters is 300ns as in the previous example. (the packets sent are identical to those shown above and the console log for the packets is not repeated here.)

```
// Barrier lifted at 558.104 542 365 6s
Rx:@1 /*558.104 542 368 0s (Barrier+2.4ns)*/ #04 #05 #06 /*558.104 542 668 0s
    (Barrier+302.4ns)*/ EOP
Rx:@2 /*558.104 542 368 0s (Barrier+2.4ns)*/ #01 #02 #03 /*558.104 542 668 0s
    (Barrier+302.4ns)*/ EOP
Rx:@3 /*558.104 542 368 0s (Barrier+2.4ns)*/ #0A #0B #0C /*558.104 542 668 0s
    (Barrier+302.4ns)*/ EOP
Rx:@4 /*558.104 542 368 0s (Barrier+2.4ns)*/ #07 #08 #09 /*558.104 542 668 0s
    (Barrier+302.4ns)*/ EOP
```

The time tags are not always so well aligned as in the previous two examples, because the sampling may detect some packets before the sampling edge and some after, in which case they appear to be delayed by a sample period, as in the next example:

```
// Barrier lifted at 390.965 208 148 3s
Rx:@1 /*390.965 208 149 3s (Barrier+1.0ns)*/ #04 #05 #06 /*390.965 208 450 7s
    (Barrier+302.4ns)*/ EOP
```

```
Rx:@2 /*390.965 208 150 7s (Barrier+2.4ns)*/ #01 #02 #03 /*390.965 208 450 7s
    (Barrier+302.4ns)*/ EOP
Rx:@3 /*390.965 208 149 3s (Barrier+1.0ns)*/ #0A #0B #0C /*390.965 208 450 7s
    (Barrier+302.4ns)*/ EOP
Rx:@4 /*390.965 208 149 3s (Barrier+1.0ns)*/ #07 #08 #09 /*390.965 208 449 3s
    (Barrier+301.0ns)*/ EOP
```

The synchronization barrier release signal is sampled in the same way as the receipt of characters, and is aligned so that, on average, it is coincident with data received by a zero-length cable. For very short cables, such as loop-backs, it is possible for the barrier release to appear to be after the start of a packet.

Synchronization with delays

In this example, delays are added after the synchronization barrier on the packets from all but port 1. The delay is signified by the = signs, representing 4-bit times, and, at the current link speed of 100Mbits/s (a bit-time of 10ns) the two = signs equate to eight bit-times or 80ns and the three = signs equate to 12 bit-times or 120ns. This can be seen from the time tags, with the start of packets from ports 2 and 4 being delayed by 80ns and the end of packet from ports 3 and 4 being delayed by 120ns:

```
@1 [ | 1 2 3 eop ]
Tx:@1 [ | #01 #02 #03 EOP ]
@2 [ | == 4 5 6 eop ]
Tx:@2 [ | == #04 #05 #06 EOP ]
@3 [ | 7 8 9 === eop ]
Tx:@3 [ | #07 #08 #09 === EOP ]
@4 [ | == 10 11 12 === eop ]
Tx:@4 [ | == #0A #0B #0C === EOP ]
// Barrier lifted at 254.719 577 049 6s
Rx:@2 /*254.719 577 050 7s (Barrier+ 1.1ns)*/ #01 #02 #03 /*254.719 577 350 7s
    (Barrier+301.1ns)*/ EOP
Rx:@1 /*254.719 577 130 7s (Barrier+81.1ns)*/ #04 #05 #06 /*254.719 577 430 7s
    (Barrier+381.1ns)*/ EOP
Rx:@4 /*254.719 577 050 7s (Barrier+ 1.1ns)*/ #07 #08 #09 /*254.719 577 470 7s
    (Barrier+421.1ns)*/ EOP
Rx:@3 /*254.719 577 130 7s (Barrier+81.1ns)*/ #0A #0B #0C /*254.719 577 550 7s
    (Barrier+501.1ns)*/ EOP
```

Removing the synchronization barrier

The ports can resign from the synchronization as follows:

```
@1- @2- @3- @4-
Tx:@1 -
Tx:@2 -
Tx:@3 -
Tx:@4 -
```

Injecting special character sequences including errors

Escape followed by FCT is a Null character, so should not generate an error:

```
[esc fct]
Tx:@4 [ ESC FCT /*=NULL*/ ]
```

Receipt of Null characters is not reported, so the lack of a received character is expected in this case.

Similarly, a time code should not generate an error:

```
[esc 1]
Tx:@4 [ ESC #01 /*=ECSS_TIMECODE-0:0:1*/]
Rx:@3 ESC #01 /*=ECSS_TIMECODE-0:0:1*/
```

Here, the time code is received and reported

The EI option is required for the following operations.

The following sequences do generate errors, as can be seen from the reports. The detected error on port 3 causes port 3 to disconnect, whereupon port 4 detects and reports a timeout:

```
[esc eop]
Tx:@4 [ ESC EOP ]
Rx:@3 ESC EOP /*=ECSS_ERROR*/
Rx:@4 /*TIMEOUT*/

[esc eep]
Tx:@4 [ ESC EEP ]
Rx:@3 ESC EEP /*=ECSS_ERROR*/
Rx:@4 /*TIMEOUT*/

[esc esc]
Tx:@4 [ ESC ESC ]
Rx:@3 ESC ESC /*=ECSS_ERROR*/
Rx:@4 /*TIMEOUT*/

esc
Tx:@4 ESC
Rx:@3 ESC ESC /*=ECSS_ERROR*/
Rx:@4 /*TIMEOUT*/
```

This last error appears to be received as two Escape characters, because it is inserted between Null characters which include an Escape character.

And an example of a parity error:

```
[1 2 ~3 eop]
Tx:@4 [ #01 #02 ~ #03 EOP ]
Rx:@3 /*101 160.223 607 776 0s*/ #01 #02 ~ /*=PARITY_ERROR*/ EEP
Rx:@4 /*TIMEOUT*/
```

The first two characters are received, but the third character, with the parity error, is not received. As the packet terminates on the error, without an EOP, the EEP character is generated to terminate the packet and notify the receiver that the packet is potentially corrupt.

Recording log files

The following sequence generated the file logfile.txt:

```
C:\dsi>java -jar SpWIO.jar /s 150 /m n /w se /u A59=192.168.0.59 /l logfile.txt
// 4Links.SpWIO run by 4links on Fri Mar 03 13:51:12 GMT 2006
// /s 150 /m n /w se /u A59=192.168.0.59 /l logfile.txt
// Attached "A59" = 192.168.0.59 is DSI-RG408, 8-ports, link mode is normal at 150Mb/s
// Input from console
@1 [1 2 3 eop]
Tx:@1 [ #01 #02 #03 EOP ]
Rx:@2 /*102 657.560 746 357 3s*/ #01 #02 #03 /*102 657.560 746 557 3s*/ EOP
@2 [4 5 6 eop]
Tx:@2 [ #04 #05 #06 EOP ]
Rx:@1 /*102 669.797 597 470 7s*/ #04 #05 #06 /*102 669.797 597 670 7s*/ EOP
@3 [7 8 9 eop]
Tx:@3 [ #07 #08 #09 EOP ]
Rx:@4 /*102 686.523 624 410 7s*/ #07 #08 #09 /*102 686.523 624 610 7s*/ EOP
@4 [10 11 12 eop]
Tx:@4 [ #0A #0B #0C EOP ]
Rx:@3 /*102 704.506 230 616 0s*/ #0A #0B #0C /*102 704.506 230 816 0s*/ EOP
^Z
// Input exhausted - exiting
```

e log file itself reports the console output from the program, without the user's commands:

```
C:\dsi>type logfile.txt
// 4Links.SpWIO run by 4links on Fri Mar 03 13:51:12 GMT 2006
// /s 150 /m n /w se /u A59=192.168.0.59 /l logfile.txt
// Attached "A59" = 192.168.0.59 is DSI-RG408, 8-ports, link mode is normal at 150Mb/s
// Input from console
Tx:@1 [ #01 #02 #03 EOP ]
Rx:@2 /*102 657.560 746 357 3s*/ #01 #02 #03 /*102 657.560 746 557 3s*/ EOP
Tx:@2 [ #04 #05 #06 EOP ]
Rx:@1 /*102 669.797 597 470 7s*/ #04 #05 #06 /*102 669.797 597 670 7s*/ EOP
Tx:@3 [ #07 #08 #09 EOP ]
Rx:@4 /*102 686.523 624 410 7s*/ #07 #08 #09 /*102 686.523 624 610 7s*/ EOP
Tx:@4 [ #0A #0B #0C EOP ]
Rx:@3 /*102 704.506 230 616 0s*/ #0A #0B #0C /*102 704.506 230 816 0s*/ EOP
// Input exhausted - exiting
```

Packets too large to be shown on the screen are recorded separately in binary files, as described in a later section.

Replaying log files

The following command replays the recording of what was sent:

```
C:\dsi>java -jar SpWIO.jar /s 150 /m n /w se /u A59=192.168.0.59 /i logfile.txt /t Tx
// 4Links.SpWIO run by 4links on Fri Mar 03 13:57:45 GMT 2006
// /s 150 /m n /w se /u A59=192.168.0.59 /i logfile.txt /t Tx
// Attached "A59" = 192.168.0.59 is DSI-RG408, 8-ports, link mode is normal at 150Mb/s
// Input from "logfile.txt"
Tx:@1 [ #01 #02 #03 EOP ]
Tx:@2 [ #04 #05 #06 EOP ]
Tx:@3 [ #07 #08 #09 EOP ]
Tx:@4 [ #0A #0B #0C EOP ]
// Input exhausted - exiting
Rx:@2 /*103 021.692 714 112 0s*/ #01 #02 #03 /*103 021.692 714 312 0s*/ EOP
Rx:@1 /*103 021.693 892 520 0s*/ #04 #05 #06 /*103 021.693 892 720 0s*/ EOP
Rx:@4 /*103 021.695 044 477 3s*/ #07 #08 #09 /*103 021.695 044 677 3s*/ EOP
Rx:@3 /*103 021.696 208 326 7s*/ #0A #0B #0C /*103 021.696 208 526 7s*/ EOP
```

The following command replays the recording of what was received:

```
C:\dsi>java -jar SpWIO.jar /s 150 /m n /w se /u A59=192.168.0.59 /i logfile.txt /t Rx
// 4Links.SpWIO run by 4links on Fri Mar 03 13:58:30 GMT 2006
// /s 150 /m n /w se /u A59=192.168.0.59 /i logfile.txt /t Rx /l logfile3.txt
// Attached "A59" = 192.168.0.59 is DSI-RG408, 8-ports, link mode is normal at 150Mb/s
// Input from "logfile.txt"
Tx:@2 #01 #02 #03 EOP
Tx:@1 #04 #05 #06 EOP
Tx:@4 #07 #08 #09 EOP
Tx:@3 #0A #0B #0C EOP
// Input exhausted - exiting
Rx:@1 /*103 066.781 474 149 3s*/ #01 #02 #03 /*103 066.781 474 402 7s*/ EOP
Rx:@2 /*103 066.783 502 634 7s*/ #04 #05 #06 /*103 066.783 502 834 7s*/ EOP
Rx:@3 /*103 066.783 668 798 7s*/ #07 #08 #09 /*103 066.783 669 053 3s*/ EOP
Rx:@4 /*103 066.785 017 210 7s*/ #0A #0B #0C /*103 066.785 017 464 0s*/ EOP
```

Modifying program parameters from within the program

It is sometimes useful to be able to change parameters of the link, for example the link speed, or whether time tags are recorded or not. Parameters can be changed by putting the parameter within round brackets:

```
(/s 400)
Tx:@4 /*/s400Mb/s*/
```

changes the link speed to 400Mbits/s.

```
(/ws)
Tx:@4 /*/ws*/
[1 2 3 eop]
Tx:@4 [ #01 #02 #03 EOP ]
Rx:@3 /*102 094.082 755 014 7s*/ #01 #02 #03 EOP
```

It is even possible to add another unit. The packets following the command show that, when multiple units are attached, it is necessary to specify which unit as well as which port the command is to be sent:

```
(/u A73=192.168.1.73)
// Attached "A73" = 192.168.1.73 is DSI-RG408, 8-ports, link mode is normal at 10.0Mb/s

[1 2 3 eop]
Tx:@A59@4 [ #01 #02 #03 EOP ]
Rx:@A59@3 /*102 228.804 446 536 0s*/ #01 #02 #03 EOP

@A73 [15 14 14 eop]
Tx:@A73@1 [ #0F #0E #0E EOP ]
Rx:@A73@2 #0F #0E #0E EOP
```

A log file can be started from within the program:

```
(/l logfile.log)
// Log file is "logfile.log"
```

Inputting commands from files

It may be useful to 'Do' a set of commands in a file, such as to replay a sequence of log files or a sequence of tests. The examples given here are simple sequences of packets.

The first file, input_file_1.txt, is just a sequence of three small packets:

```
1 2 3 eop
4 5 6 eop
7 8 9 eop
```

```
(/i input_file_1.txt)
// Input from "input_file_1.txt"
Tx:@1 #01 #02 #03 EOP
Tx:@1 #04 #05 #06 EOP
Tx:@1 #07 #08 #09 EOP
// Input from "input_file_1.txt" finished
// Input from "(console)" resumes
Rx:@1 #01 #02 #03 EOP
Rx:@1 #04 #05 #06 EOP
Rx:@1 #07 #08 #09 EOP
```

In some cases it may be useful to pause, so that the response to the first packet returns before the next packet is sent. The file input_file_2.txt puts a pause of 1000 milliseconds at each carriage-return in the file, which gives the desired result. Note that a comment has been added to describe what the file does:

```
// This file contains a command to pause for 1 second at the end of each line,
// and then has three short packets.
(/d 1000)
1 2 3 eop
4 5 6 eop
7 8 9 eop
```

```
(/i input_file_2.txt)
// Input from "input_file_2.txt"
Tx:@1 #01 #02 #03 EOP
Rx:@1 #01 #02 #03 EOP
Tx:@1 #04 #05 #06 EOP
Rx:@1 #04 #05 #06 EOP
Tx:@1 #07 #08 #09 EOP
Rx:@1 #07 #08 #09 EOP
// Input from "input_file_2.txt" finished
// Input from "(console)" resumes
```

Nesting input files

It may be useful to construct packets from a variety of different sources, for example a routing heater, protocol header, packet body, and termination. Each of these can be in separate files, and the set of files can be input by another file. The example here, nested_input_file.txt uses one file to call up three more files:

```
(/i path_routing_header.txt)
(/i packet_body.csv)
(/i packet_termination.txt)
```

The path routing header is just three individual bytes to direct the packet through three routing switches, followed by the default logical header value of 254 which is recommended for use when there is not an actual logical address. A pair of delimited comments are added on the one line:

```
/* Three-bytes of path address */ 5 3 6 /* followed by one byte of logical address */ 254
```

The packet body in this example was prepared in an Excel spreadsheet and saved as a .csv file (Comma Separated Variable). The spreadsheet provides a simple way to generate a large number of values according to a simple rule, and the SpWIO program interprets the commas as separators just as if they were spaces. The packet_body.csv file, opened in a text editor, appears as:

```
0,1,2,3,4,5,6,7
8,9,10,11,12,13,14,15
16,17,18,19,20,21,22,23
24,25,26,27,28,29,30,31
32,33,34,35,36,37,38,39
40,41,42,43,44,45,46,47
48,49,50,51,52,53,54,55
56,57,58,59,60,61,62,63
```

The termination might include a checksum and the end-of-packet marker. In this example, packet_termination.txt, an arithmetic checksum is calculated (in the spreadsheet) of the packet body, and this is transmitted as a big-endian 16-bit value.

```
2016S
eop
```

The listing below is a screen/console log of running the nested_input_file.txt. Before inputting the file, a log file is set up to record the results.

```
(/l logfile.log)
// Log file is "logfile.log"
(/i nested_input_file.txt)
// Input from "nested_input_file.txt"
// Input from "path_routing_header.txt"
Tx:@1 #05 #03 #06 #FE
// Input from "path_routing_header.txt" finished
// Input from "nested_input_file.txt" resumes
// Input from "packet_body.csv"
Tx:@1 #00 #01 #02 #03 #04 #05 #06 #07
Tx:@1 #08 #09 #0A #0B #0C #0D #0E #0F
Tx:@1 #10 #11 #12 #13 #14 #15 #16 #17
Tx:@1 #18 #19 #1A #1B #1C #1D #1E #1F
Tx:@1 #20 #21 #22 #23 #24 #25 #26 #27
Tx:@1 #28 #29 #2A #2B #2C #2D #2E #2F
Tx:@1 #30 #31 #32 #33 #34 #35 #36 #37
Tx:@1 #38 #39 #3A #3B #3C #3D #3E #3F
// Input from "packet_body.csv" finished
// Input from "nested_input_file.txt" resumes
// Input from "packet_termination.txt"
Tx:@1 #07 #E0
Tx:@1 EOP
// Input from "packet_termination.txt" finished
// Input from "nested_input_file.txt" resumes
// Input from "nested_input_file.txt" finished
// Input from "(console)" resumes
Rx:@1 #05 #03 #06 #FE #00 #01 #02 #03 #04 #05 ... /* Total 70 bytes in "logfile.log_V401_1_1"
    */ EOP
```

The file is sent to a special (binary) log file because it is larger than the threshold set for displaying the whole file on the screen. The filename is constructed from the log file specified, the unit name, the output port, and a sequence number. In this case the log file was logfile.log, the unit name was V401, the port was 1 (looped back), and the sequence number 1, to make logfile.log_V401_1_1.

Binary files can be replayed, as shown in the next section.

Sending binary files

Binary files, whether generated off-line or recorded as in the previous section can be transmitted. The command for this is `binary` followed by the filename in brackets immediately following the word `binary`:

```
binary(logfile.log_V401_1_1) eop
Tx:@1 BINARY(logfile.log_V401_1_1) EOP
Rx:@1 #05 #03 #06 #FE #00 #01 #02 #03 #04 #05 ... /* Total 70 bytes in "logfile.log_V401_1_2"
    */ EOP
```

The `binary` command does not add an end of packet terminator, so that multiple files, including multiple binary files, can be sent as a single packet.

As in the example in the last section, the packet is output to a binary log file, this time with the sequence number 2 at the end. Comparing the two files shows them to have identical contents.

Any file can be transmitted with this binary command. In this example, the file sent is a jar file, V401.jar:

```
binary(V401.jar) eop
Tx:@1 BINARY(V401.jar) EOP
Rx:@1 #50 #4B #03 #04 #0A #00 #00 #00 #00 #00
Rx:@1 ... /* Total 31140 bytes in "logfile.log_V401_1_3" */ EOP
```

Comparing the received logfile.log_V401_1_3 file with the original jar file again shows that the two files have identical content.

Several `binary` commands can be used in a single file together with input from other files and literal values such as may be appropriate for headers and EOP.

Running programs from within SpWIO

It may be useful to run programs from within SpWIO, for example to 'Do' a set of commands in a file, or to run a test program or to interpret a protocol. The example given here is a simple batch file to give a required delay, without having to count the number of equals signs. It is used in this example as a test of the timeout delay.

The batch file produces as output the = characters that are included in place of the program "call". (N.B. each = is 4 transmit bit-times — 40ns at 100Mb/s).

It calculates how many ='s are needed and then, in a crude loop, forms a suitable string (in e) before outputting it (with 'echo'). [and] are included to keep multiple ='s together, and to ensure echo has a non-null string to output (else it gives an unhelpful status message).

delay_ns.bat is

```
@SET/a n = %tx_speed% * %1 / 4000
@set e=
:x
@IF %n% LSS 1 GOTO done
@SET e=%e%=
@SET/a n = %n% - 1
@GOTO x
:done
@ECHO [%e%]
```

The delay program is run here on a DSI-RG408, with a short cable between ports 3 and 4.

When the links are running at high speed, there is a tendency for Null characters to be inserted, including in a delay as in this example, so the link speed should be set to 100Mbits/s for this test.

```
(/s 100)
Tx:@4 /*/s100Mb/s*/

delay_ns.bat(800)
Tx:@4 [ ================ ==== ]
```

(Unfortunately, the .bat extension is needed) This delay of 800ns is within the 850ns nominal Timeout, and so the link does not disconnect.

```
delay_ns.bat(880)
Tx:@4 [ ================ ====== ]
Rx:@3 /*TIMEOUT*/
Rx:@4 /*TIMEOUT*/
```

The delay of 880ns is beyond the nominal 850ns Timeout, and so the link disconnects.

Running protocol plug-ins, with examples from the RMAP plug-in

These examples are based on the RMAP plug-in which is described in the section "Remote Memory Access Protocol, RMAP, on page 20 of this manual. To run the plug-in with the SpWIO program, use the `/p <protocol-name>` parameter in the command line. For example to run the RMAP plug-in:

```
G:\>java -jar spwio.jar /p RMAP /u 192.168.0.251
//-4Links.SpWIO (v14:20061216) run by 4links on Fri Jan 16 10:50:24 GMT 2009
//-/p RMAP /u 192.168.0.251
//-Added Tx protocol "RMAP" (v10:20081216) from "ProtocolTx_RMAP.class"
//-Added Rx protocol "RMAP" (v10:20081216) from "ProtocolRx_RMAP.class"
//-Attached 192.168.0.251 is SRR-RG40x/8-EW-ER-TT-SO, 3-ports, link mode is normal at
    10.0Mb/s
//-Log file is "SpWIO_20090116_105025.log"
//-Input from "(console)"
```

To generate a transmit packet from a protocol plug-in you should name the protocol and add bracketed parameters – the plug-in will run, interpret your parameters and send a suitable packet.

```
<protocol-name>( <parameters> )
```

To find out the parameters that the plug-in uses, type the protocol name with no parameter. For example with RMAP, type:

```
    @1 RMAP()
Tx:@1 RMAP ( {W(rite) <value value ...> | R(ead) <number-of-bytes>} @ <address> [A(cknowledge)]
    [E(xtended-address) <n>] [F(ixed-address)] [K(ey) <n>] [D(estination) <address...>]
    [S(ource-path) <address_bytes>] [T(ransaction-identifier) <n>] [V(erify)] )
```

Generate an RMAP packet to read 10 bytes of data at address 0:

```
@1 RMAP(r 10 @ 0)
```

This will generate a fuller description of the packet being sent:

```
Tx:@1 RMAP (Transaction ID #0001, Key #00) Read 10 bytes from #00:00000000... Source path 254
```

Received packets will be presented to the loaded protocol plug-ins so that, if they are recognised, the content can be interpreted and displayed in a suitable format.

The return packet from the above RMAP read request might appear as:

```
Rx:@1 RMAP Read reply: To #FE, From #FE, Transaction ID #0001, Status = OK: #00 #00 #00 #00
    #00 #00 #00 #00 #00 #00 (Header CRC OK) (Data CRC OK)
```

A simple RMAP write, without Acknowledgement, just generates the fuller description of the packet:

```
@1 RMAP(w 1 2 3 4 @ 1)
Tx:@1 RMAP (Transaction ID #0002, Key #00) Write {#01 #02 #03 #04} to #00:00000001... Source
    path 254
```

Having written to the RMAP memory, the memory can be read again, with the response showing that the data had been written correctly:

```
@1 RMAP(r 10 @ 0)
Tx:@1 RMAP (Transaction ID #0003, Key #00) Read 10 bytes from #00:00000000... Source path 254
Rx:@1 RMAP Read reply: To #FE, From #FE, Transaction ID #0003, Status = OK: #00 #01 #02 #03
    #04 #00 #00 #00 #00 #00 (Header CRC OK) (Data CRC OK)
```

A more elaborate RMAP write, with Acknowledge and a Key, generates both the fuller description of the write packet and the acknowledge response

```
@1 RMAP(w 5 6 7 8 @ 5 A K #AA )
Tx:@1 RMAP (Transaction ID #0004, Key #AA) Write {#05 #06 #07 #08} to #00:00000005...
    Acknowledge Source path 254
Rx:@1 RMAP Read reply: To #FE, From #FE, Transaction ID #0004, Status = OK (Header CRC OK)
```

And another RMAP memory read again shows that the data had been written correctly:

```
@1 RMAP(r 10 @ 0)
Tx:@1 RMAP (Transaction ID #0005, Key #00) Read 10 bytes from #00:00000000... Source path 254
Rx:@1 RMAP Read reply: To #FE, From #FE, Transaction ID #0005, Status = OK: #00 #01 #02 #03
    #04 #05 #06 #07 #08 #00 (Header CRC OK) (Data CRC OK)
```

Issues

These known issues with the DSI products should be noted when using SpWIO. They are all concerning synchronized outputs and do not affect use of SpWIO with the ESL products.

High-speed synchronized outputs

As of the date of this manual, additional Null characters tend to be inserted within the data stream when high link speeds are selected, as can be seen by this example of a synchronized set of outputs at 400Mbits/s. This is a firmware issue rather than a program issue, but it may be useful to point out the effect as seen when running SpWIO:

```
@1 [| 1 1 1 eop]
Tx:@1 [ | #01 #01 #01 EOP ]
@2 [| 2 2 2 eop]
Tx:@2 [ | #02 #02 #02 EOP ]
@3 [| 3 3 3 eop]
Tx:@3 [ | #03 #03 #03 EOP ]
@4 [| 4 4 4 eop]
Tx:@4 [ | #04 #04 #04 EOP ]
// Barrier lifted at 610.355 563 407 0s
Rx:@4 /*610.355 563 405 3s (Barrier-1.7ns)*/ #03 #03 #03 /*610.355 563 541 3s (Barrier+134.3ns)*/ EOP
Rx:@1 /*610.355 563 406 7s (Barrier-0.3ns)*/ #02 #02 #02 /*610.355 563 541 3s (Barrier+134.3ns)*/ EOP
Rx:@2 /*610.355 563 406 7s (Barrier-0.3ns)*/ #01 #01 #01 /*610.355 563 541 3s (Barrier+134.3ns)*/ EOP
Rx:@3 /*610.355 563 405 3s (Barrier-1.7ns)*/ #04 #04 #04 /*610.355 563 561 3s (Barrier+154.3ns)*/ EOP
```

At 400Mbits/s a data character should have a duration of 25ns, so the three data characters should take 75ns. Additional Null characters result in the first three of these packets having durations of approximately 135ns, and the fourth packet having a duration of 155ns. This is a known issue and will be corrected. The problem is less evidenced at lower speeds, but nevertheless it can occasionally occur at link speeds above 100Mbits/s. It appears to occur much more at 200Mbits/s than at 199Mbits/s.

Low-speed synchronized outputs

Synchronization is achieved by holding D and S without transitions for a period. At low transmit speeds this period may exceed the SpaceWire timeout period. Using synchronized outputs at or above 100Mbits/s will avoid such timeouts.

Resigning too soon from a barrier synchronization

Due to `RESIGN` being actioned early in a pipeline, closely spaced `BARRIER` and `RESIGN` commands may result in resignation before synchronization. The minimum spacing is TBD - a few bytes of data is enough to prevent unexpected behaviour.

SpWIO capabilities available in 4Links products

In the table, a tick indicates that the capability is present in the basic product without additional options, and an option code indicates that the option is required for the capability. SpWIO can also be used with other 4Links products such as the SpaceWire-PCI and SpaceWire-cPCI boards, and other products to be announced.

Capability		ESL-RFxxx	ESL-RGxxx	DSI- / SPG-RGxxx	SRR
/i	Input file	✓	✓	✓	✓
/d	Delay on CR	✓	✓	✓	✓
/l	Record log file	✓	✓	✓	✓
/a	Record long packets to binary files	✓	✓	✓	✓
/q	Quiet Mode	✓	✓	✓	✓
/t	Filter transmit packets from input/replay file	✓	✓	✓	✓
/v	View tokens	-ER	-ER	-ER	
/x	Ignore tokens			-EI	
/f	Flow control			-EI	
/ew	Event/Waveform cause	-EW	-EW	-EW	
/es	Event/Waveform source	-EW	-EW	-EW	
/m	Link mode	✓	✓	✓	✓
/s	Link speed	✓	✓	✓	✓
/w	Time Tags (resolution)	-TT (100ns)	-TT (< 10ns)	-TT (< 2ns)	
`@<unit>`		✓	✓	✓	✓
`@<port>`			✓	✓	✓
`<number>s`	16-bit, little-endian	✓	✓	✓	✓
`<number>S`	16-bit, big-endian	✓	✓	✓	✓
`<number>w`	32-bit, little-endian	✓	✓	✓	✓
`<number>W`	32-bit, big-endian	✓	✓	✓	✓
`'abcd'`	Text string	✓	✓	✓	✓
`//`	Comment until end of line	✓	✓	✓	✓
`/* … */`	Comment delimiters within a line or over multiple lines	✓	✓	✓	✓
`EOP`	End Of Packet	✓	✓	✓	✓
`EEP`	Error End of Packet	✓	✓	✓	✓
`EVENT`	Insert event trigger			-EW	
`ESC`	Escape			-EI	
`FCT`	Flow-Control Token			-EI	
`~`	Invert Parity			-EI	
`=`	Delay D and S transitions			-EI, -SO	
`[ ]`	Multiple packet Store and Forward buffer			1kByte	
`(parameters)`	Change parameters within program	✓	✓	✓	✓
`binary(filename)`	Transmit a binary file	✓	✓	✓	✓
`<program>(parameters)`	Run a program	✓	✓	✓	✓
`+`	Join Sync group			-SO	
`-`	Resign from Sync group			-SO	
`|`	Synchronization Barrier			-SO	

4Links®

Quick Reference

Command Line: `java -jar SpWIO.jar <parameters>`

<parameters>	default	description
/m [dnl] [sf]	n	Set link mode [disabled, normal, legacy] [slow, fixed];
/s <number>	10Mb/s	Link transmit speed on all ports;
/q [y][n][t][f]	n	Quiet mode;
/l <file>	no log	Keep a record of activity;
/a <nn>	Show all	abbreviate received packet in console view and, if a log file has been specified, send long packet to binary file
/i <file>	Console	take command input from file;
/t <label>	"	transmit only sequences labelled <label>;
/d <dd>	0	Delay of dd ms at end of each line of input file;
/u [<name>=]<IP-address>		specify a unit, and give it a name.
/p <name>		Add plug-in for protocol <name>.
/v [(delta) event-list]	xtabc	Tokens to be viewed on console (and log file)
/w [(delta) event-list]	never	When to add a timetag
/x [(delta) event-list]		Exception events – these events do not cause link failure
/ew [(delta) event-list]		Event trigger
/es [1-8, 0, A-D]		Event source

Options

TT Timetags	EW Events & Waveforms	ER Event Reporting	SO Synchronized Outputs	EI Error injection

ESL	DSI/SPG	Event
	a	ESC-ESC
	b	ESC-EOP
	b	ESC-EEP
	d	NCHAR
e	e	EOP
	f	FCT
e	g	EEP
	i	First null

ESL	DSI/SPG	Event
m	m	Mid-packet
	n	NULL
	p	Parity error
s	s	Start of packet
	t	Time code
	v	Excess data
	w	Excess FCT
	x	Timeout

Transmit

@<unit>	Select unit for subsequent data
@<port>	Select port for subsequent data
<number>	
23	Decimal byte
023	Octal byte (= 19 decimal)
#23	Hexadecimal byte (= 35 decimal)
0x23	Hexadecimal byte (= 35 decimal)
<number>s	(lower case s) Short word (16-bits, 2bytes) Little-endian
<number>S	(upper case S) Short word (16-bits, 2bytes) Big-endian
<number>w	(lower case w) Word (32-bits, 4bytes) Little-endian
<number>W	(upper case W) Word (32-bits, 4bytes) Big-endian
'abcd'	String of bytes, leftmost first; \' represents '
//	Comment until end of line
/* … */	Comment delimiters within a line or over multiple lines
EOP	End-of-packet
EEP	Error-end-of-packet
[	Store data temporarily
]	Forward stored data
(parameters)	Change SpWIO parameter(s) from within the program
binary(filename)	Transmit a binary file (without EOP)
<program>(parameters)	Run an external program or batch file with parameters
<protocol>(parameters)	Use the protocol plug-in to generate a transmit packet
<CR>	Return will flush the transmit buffer
ESC	Escape
FCT	Flow control token
~	Invert parity bit
=	Delay D and S transitions
+	Join a barrier synchronization
-	Resign from a barrier synchronization
|	Barrier synchronization, all synchronized ports will transmit at the same time
EVENT	Insert an event trigger