API Guide

Our products come into two different groups, the first being a passive recorder (MSR/SLR) used for intercepting and recording SpaceWire traffic. The second (DSI) is used for sending and receiving data from applications written to emulate or communicate with SpaceWire devices .

Passive Recorder (MSR/SLR)

This device is used for recording SpaceWire traffic and events, we provide various programs for recording data acquired by these devices. It is not anticipated that users will have to write their own tools for these devices. However, if you wish to write your own capture application then we provide the tools for you to do so.

We now provide comprehensive tools for recording data from our Passive Recorders so if you feel that you are having to write your own capture software please contact us as we may be able to save you some considerable time.

Active Devices (DSI)

This device is used for sending and receiving data to and from SpaceWire devices. Applications can send and receive traffic from the DSI., this allows users to model their SpaceWire networks and write verification/validation test suites.

The API guide is aimed at users who are using this kind of device.

Application Overview

4Links SpaceWire devices are connected to a host computer over a **TCPIP** protocol, 4 Links provide an API in order such that application developers need not be concerned with the low-level control of the device and its settings.

An application goes through the following phases

- Connection, connecting to the device
- Physical Link Attributes, Configuring the physical links we are using
- Virtual Link Attributes, Configuring the link options
- Traffic, running traffic through and or obtaining traffic from the device
- Termination, closing the connection to the device.

For each language we supply bindings for, we provide Doxygen information with functions broken down into the appropriate group.

The language binding documentation is broken down into the above groups

Connection

This is simply creating a connection to the device and is documented in the **Connection** module in each language.

Physical Link Attributes

Once connected we need to configure the physical links and is documented in the **Physical Link Attributes** module for each language.

Firstly, we use the set speed function to set the speed for links we don't want to run at the default 10Mbits.

Then, for each link you want to use :-

- set it as the active link
- set the mode of the device
    - if you want to use a 10Mbit link set mode LINK_mode_fixed
    - Otherwise LINK_mode_normal
    - There are other legacy modes available

Once setup, we need to check that the link is connected, for each link

- set the active link
- Call appropriate link connected verb to check if the link is connected, you may need to retry this as the links stabilize, we would suggest waiting a second between retries.

Virtual Link Attributes

Once we have physically set up the links, we need to set up its virtual attributes. These virtual attributes are used to generally control events occurring on the link.

Devices can be purchased with additional options which extend the functionality of the device. These being

- TimeTags (**TT**), the timetag option is used to provide nano second resolution for events occurring on the network
- Error Reporting **(ER),** to detect and report errors occurring on the SpaceWire network
- Error Waveforms (**EW)** to provide detailed network waveform data on SpaceWire errors

Timetags (Option TT)

The time-tag option sends a time-tag on the protocol stream when the following events are observed on a SpaceWire link. The time-tag is sent immediately before the observed event.

- first_byte, send a timetag when the first byte of a packet is observed.
- intermediate_bytes, send a timetag when middle bytes in a packet are observed.
- EOP, send a timetag on end of a packet
- EEP,send a timetag on end of error packet
- time_code, send a timetag on receipt of a SpaceWire timecod
- parity_error send a timetag on a parity error
- ESC_ESC, send a timetag on an illegal "ESC ESC"
- ESC_EOP, send a timetag on an illegal "ESC EOP"
- ESC_EEP, send a timetag on an illegal "ESC EEP"
- Timeout, send a timetag on a link timeout
- FCT, send a timetag when an FCT token is observed.
- NULL, send a timetag when a NULL token is observed.

Note that requesting timetags for events such as FCT and NULL may potentially indiuce a large amount of traffic to your client.

Error Reporting (Option ER)

The error-reporting option enables or disables the reporting of events ocurring in the SpaceWire stream. The default is that the only events reported are the receipt of Nchars (data bytes, EOPs and EEPs). Further events typically relate to errors on a link.

- time_code, report a time code being received.
- EOP
- EEP
- parity_error,report a parity error.
- ESC_ESC,rReport an illegal "ESC ESC" error.
- ESC_EOP, report an illegal "ESC EOP" error.
- ESC_EEP, report an error on illegal "ESC EEP
- Timeout , report an error on link timeout
- FCT, report an FCT being received
- NULL, report a NULL token being received

Error Waveforms

Error Waveforms gather signal information on a link when a triggering event occurs on a link (which may be the same link). Thus, there are two set of options associated with waveforms

- EW_enable_reporting sets the triggers for a link to be in a triggered state
- EW_source sets the set of links on of which has to be triggered for a waveform to be gathered on the link

The triggers which may be set by EW_enable_reporting are :-

- First null
- First fct
- Running error
- Starting error
- Middle character
- time_code, time code being received.
- EOP
- EEP
- parity_error, report a parity error.
- ESC_ESC, illegal "ESC ESC" error.
- ESC_EOP, illegal "ESC EOP" error.
- ESC_EEP, illegal "ESC EEP
- Timeout , report an error on link timeout
- FCT, report an FCT being received
- NULL, report a NULL token being received

The triggers s:ources which may be set on a link by EW_source are :-

- Barrier
- Port 1
- Port 2
- Port 3
- Port 4
- Port 5
- Port 6
- Port 7
- Port 8
- SMA ports 1&2
- SMA ports 3&4
- SMA ports 5&6
- SMA ports 7&8
- Local Clock

Traffic

Once the physical and virtual attributes have been setup, we can start the core application of sending and receiving traffic.

Sending Traffic

When we want to send data on a SpaceWire link we call the relevant send function (having set the active link if it may have changed). The send function takes the following information

- the buffer we want to send
- the size of the buffer
- SpaceWire packet attributes
    - EOP, the packet is terminated with a SpaceWire End of Packet (EOP)
    - EEP, the packet is terminated with a SpaceWire Error End of Packet
    - PART_EOP_EEP, the packet is not terminated it is expected that a later call to the send function is made with an EOP or EEP terminator
    - EXTN

EEP is not normally be used to terminate a packet but is available here to assist with testing where an erroneous packet may usefully be generated.

PART_EOP_EEP is **NOT** part of the SpaceWire standard, it just allows for the buffering of data within the application and device.

Note that the send function queues data in an internal buffer and that immediate transmission of complete data block is not guaranteed. If you need to force the data to go onto the wire then you can use the **flush** function.

Both the send and flush functions are blocking calls.

Reading Packet data

Reading packet data is very similar, the application calls the appropriate read function, passing

- The buffer to read data into
- The size of the buffer
- Flags indicating how out of band data is handled

The read function returns the actual amount of data read, along ide this the type of packet being read is available, this being: -

- EOP, the packet is terminated with a SpaceWire End of Packet (EOP)
- EEP, the packet is terminated with a SpaceWire Error End of Packet
- PART_EOP_EEP, the packet is not terminated as there is more data to follow.

Reading a packet is currently a **blocking** operation and if you want your application to do anything else at the same time then you must make use of threads.

It is possible to set up a receive timeout whereby if a packet data is not received within the given number of milliseconds the function returns with an error.

Out of Band Data

The data stream from the device contains a mixture of SpaceWire Data and data about events occurring on those links, all of which arrive at the read call.

The C++ and Python languages automatically handle the reception of this data and call virtual functions which the application can action as appropriate.

Java : in order to maintain compatibility with previously released versions of the API, the application must explicitly enable the handling of out of bound data by calling **enable_callbacks**

Waveform Data

Waveform data must be explicitly requested from the device, however if you are writing data in a separate thread you must wrap all of your send calls in a lock. The Java API provides synchronized calls for you to ensure the transmit stream is not corrupted.