Python API
User Guide

Overview
We provide a Python API for those developers who wish to program in python.

Release
The python api is available from software release 34.0

Platforms
This binding is available for Windows 64 bit platforms, CentOS 6 and CentOS 7 for python 2.7.

> **CentOS 6&7** the binding is present in the python-libs RPM with the examples present in the python-examples RPM. I
>
> **CentOS 6** requires the python-argparse RPM to be installed in order to run some examples.
>
> **Windows** it is installed by the installation batch script

Python3 support is also available on Windows 64 platforms.

Environment
If you run the appropriate script to set up the environment the python module is added to the **PYTHONPATH** variable.

Examples
Python examples can be found in ~/etherlinks/python/examples where ~ is the chosen installation directory for the 4 Links software.

Simple Python Smoke Test

The following test returns the serial number for a device with IP address 1.2.3.4

```
import sys

# pull in the the 4links python library
from EtherSpaceLink import *


eslcon = connection(“1.2.3.4”)
print ("Serial       : %s" % eslcon.serial())
```

API
Guide

The python API is installed in the ~/python/libs directory and the PYTHONPATH is module is set up accordingly via the environment set up scripts. The Python API makes use of the EtherSpaceLink DLL/shared object and as such if you want to copy the python api into a different location you will also need to ensure that the DLL/shared object is still accessible.

The python API is very close in functionality to that of the C API and as such each function has an analogous C API which contains the full documentation.

API Callback Methods

These functions are called when something happens on the link, if you are not interested in these events then don't override them in your class.

link_selected(self, link_)
A different link has been selected, i.e. any new data arriving has arrived on this link

link_timeout(self, link_)
A link has suffered a timeout.

link_status(self, link_, rxspeed_, connected, runstatus_)
Called when rx information about a link has been requested

link_tx_speed(self, link_, txspeed_)
Called When tx information about a link has been requested

parity_error(self, link_)
Called when a parity error has occurred.

error_event(self, link_)
Called when an error event has occurred on the link

perror1(self, link_)
Called when an error has occurred on the link

perror2(self, link_)
Called when an error has occurred on the link

timetag_tt(self, time_)
A timetag record containing the aboslute time

timetag(self, time_)
A timetag record containing the time since the last time tag

err(self, time, state, error_bits)
Error has occurred on the currently active link

waveform_start(self, hdr_, data_, sz)
Waveform start

waveform_data(self, hdr_, data_, sz)
Waveform data

waveform_end(self, hdr_, data_, sz)
Waveform end

unknown_ram_data(self, data_, length_, complete_, data_buffer_position_)
Called when data has been received and the API does not know how to interpret

unknown_special_data(self, data_, length_, complete_, data_buffer_position_)
Called when unknown special data has been received and the API does not know how to interpret

unknown_extn_data(self, data_, length_, complete_, data_buffer_position_)
Called when unknown special data has been received and the API does not know how to interpret

special_data(self, data_, length_, complete_, data_buffer_position_)
Called when special data is received, override this and no further processing occurs

extn_data(self, data_, length_, complete_, data_buffer_position_)
Called when extension data is received, override this and no further processing occurs

Class Methods

version()
Returns string containing the version of the underlying API

__init__(address_, file_=False)
Class constructor, it can be given a host to connect to or a filename (and file_ set to true)

abort()
Forces any thread on a read call off the read, once called no more data can be sent over the connection

flush()
Flushes the transmit buffer

read(data_)
Reads into the bytearray returning the number of bytes read

read_info():
Returns the packet type from the last read, **EOP** or **EEP** or **PART_EOP_EEP**

write(data_, flags_)
Writes the bytearray data_ to the currently active link.
flags indicates the packet type

EOP	End Of Packet
EEP	Error End of Packet
PART_EOP_EEP	Part packet

active_port(link_)
Sets the current active link

mode(mode_)
Sets the mode of the currently active link. Mode_ may be one of

- LINK_mode_disabled
- LINK_mode_normal
- LINK_mode_legacy
- LINK_mode_master
- LINK_mode_long_timeout
- LINK_mode_fixed_speed
- LINK_mode_slow_speed

rx_speed()
Returns the current rx_speed

manufacturer()
Returns the manufacturer string of the device

options()
Returns a string describing the firmware options installed on the device

product()
Returns a string describing the product

mac()
Returns the 6 byte MAC address of the device

serial()
Returns a string containing the serial number of the device

request_link_status()
Requests the current link status

request_tx_speed()
Requests the current link speed

link_connected()
Returns if the link is connected (1) and 0 if not

nolinks()
Returns the number of links the device supports.

record(file_, recordwrites_, writeerror_=False)
Sets the recording file indicating if we want to recordwrites, and an optional parameter indicating if write errors should be returned back to the write call.

record_flush()
Flush the current recording file.

record_size()
Returns the current number of bytes in the current recording file

record_file()
Returns the current record file name

log(file_)
Sets the current log file

raw()
Returns the underlying EtherSpaceLink pointer

eintr(return_)
Sets whether EINTR returns an error on the read call

set_rx_timeout(timeout_)
Sets the rx timeout

get_rx_timeout(timeout_)
Gets the rx timeout

percent_read()
When reading from a file returns the % of the file read

/TT_module(module_)
Returns the slot for the given module id

slot(slot_)
Returns the module for the given slot

socket()
Returns the underlying network socket

enable_timecode_rx(enable_)
Set whether timecodes are sent to the client, enable_ True if wanted, False if not

TT_reporting(when_)
Sets timetag reporting

ER_reporting(when_)
Sets Error reporting

ER_bits(buffer_)
Returns the error bits from the buffer

EW_reporting(when_)
Sets the when an Error Waveform is generated

EW_source(source_)
Sets the Error Waveform sources

EW_request(link_)
Requests waveform for the given link

EW_clear(link_)
Clears the error waveform the given link

request_now()
Returns the current time on the device

dump(prefix_)
Dumps to the given file with the given prefix

dump_limit(limit_)
Sets the maximum number of dump records to be generated

EI_ignore_events(what_)
Ignore events from the EI module, the EI module will disconnect a link in the event of an error, this function allows the link to ignore errors and continue running in the event of a particular error

EI_flow_control(initial_fct_, fct_)
The ECSS-E-ST-50-12C SpaceWire standard requires at least one flow-control token to be sent to starta link, and the maximum flow-control credit to be 56 N-Chars (as indicated by 7 flow-control tokens). Flow-control tokens are normally issued as space becomes available in the receive buffer. Data tokens (actually N-Chars - data, end-of-packet and error-end-of-packet) may be received up to the number for which credit has been issued. EI_flow_control() allows the link to be set outside the limits defined by the ECSS SpaceWire standard

write_buffer_empty()
Returns if the write buffer is empty for the currently active link

set_max_packet_data(sz_)
Control the data receive compressor - discard data from packet

sma_56_pulse_width(width_)
It is possible to connect an SMA connector. This sets the pulse width of the device

set_timecode_rx(enable_=True)
Set where timecode rx is enabled

set_timecode_tx(bits_, first_, interval_, report_)
Set where timecode tx is enabled and the time code parameters

request_timecode_rx_status()
Requese time code rx status

request_timecode_tx_status()
Reques time code tx status

io_stats(rxtotal_, calls, iocalls)
Returns rx io statistics

timeout_errors(errors_=True)
Sets whether a timeout generates an error

non_blocking(blocking_=True
Marks the socket as non blocking

Exceptions

All of the above functions will throw an error exception when an error occurs.

The error exception has the following methods

get() returns the esl error which had occurred (see EtherSpaceLink_Constants.h)
text() returns textual description of error
oserr() returns OS error which last occurred
name() returns the name of the device the exception occurred on