Getting Started

1 Installation 3
2 Tutorial 5
3 Command-Line Tool 7
4 PyLink 11
5 Protocols 57
6 Unlocking 61
7 Bindings 63
8 Extras 93
9 Troubleshooting 107
10 Serial Wire Debug 109
11 About 117

Python Module Index 119
PyLink is a Python package that enables you to control your J-Link from Python. This library was developed at Square to enable us to leverage our J-Link as a part of our test infrastructure, which was written in Python.

Getting started is as simple as:

```python
>>> import pylink
>>> jlink = pylink.JLink()
>>> jlink.open(serial_no=123456789)
>>> jlink.product_name
J-Trace Cortex-M
```
CHAPTER 1

Installation

Warning: This package requires the J-Link Software and Development Pack provided by SEGGER. If you do not currently have the development pack installed, you must install it first before using this Python package.

Note: This library is known to support Python versions 2.4 - 2.7. Support for versions higher than 2.7 is not guaranteed.

1.1 Basic Installation

Installing PyLink with pip:

$ pip install pylink-square

Or use easy_install:

$ easy_install pylink-square

1.2 Building From Source

Clone the project into a local repository, then navigate to that directory and run:

$ python setup.py install

This will give you the tip of master (the development branch). While we strive for this to be stable at all times, some bugs may be introduced, so it is best to check out a release branch first before installing.

$ git checkout release-major.minor
$ python setup.py install
1.3 External Dependencies

In order to use this library, the J-Link Software and Development Pack provided by SEGGER is required. Once you have a copy of the development pack, you can start using PyLink. PyLink will automatically find the library if you installed it using one of the installers available from SEGGER’s site, but for best results, you should also do one of the following depending on your operating system:

1.3.1 On Mac

<table>
<thead>
<tr>
<th>Option A</th>
<th>Copy the library file to your libraries directory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp libjlinkarm.dylib /usr/local/lib/</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option B</th>
<th>Add SEGGER's J-Link directory to your dynamic libraries path.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ export DYLD_LIBRARY_PATH=/Applications/SEGGER/JLink:$DYLD_LIBRARY_PATH</td>
<td></td>
</tr>
</tbody>
</table>

1.3.2 On Windows

Windows searches for DLLs in the following order:

1. The current directory of execution.
2. The Windows system directory.
3. The Windows directory.

You can copy the JLinkARM.dll to any of the directories listed above. Alternatively, add the SEGGER J-Link directory to your %PATH%.

1.3.3 On Linux

<table>
<thead>
<tr>
<th>Option A</th>
<th>Copy the library to your libraries directory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ cp libjlinkarm.so /usr/local/lib/</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option B</th>
<th>Add SEGGER's J-Link library path to your libraries path.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ export LD_LIBRARY_PATH=/path/to/SEGGER/JLink:$LD_LIBRARY_PATH</td>
<td></td>
</tr>
</tbody>
</table>
In this tutorial, assume that the serial number of the J-Link emulator being connected to is 123456789, and that the target device is an Mkxxxxxxxxxx7.

### 2.1 Connecting to an Emulator

```python
>>> import pylink
>>> jlink = pylink.JLink()
>>> jlink.open(123456789)
>>> jlink.product_name
J-Trace Cortex-M
>>> jlink.oem
>>> jlink.opened()
True
>>> jlink.connected()
True
>>> jlink.target_connected()
False
```

### 2.2 Updating the Emulator

```python
>>> jlink.update_firmware()
1
```

### 2.3 Connecting to a Target CPU

```python
>>> jlink.connect('MKxxxxxxxxxx7')
>>> jlink.core_id()
50331903
>>> jlink.device_family()
3
>>> jlink.target_connected()
True
```
2.4 Flashing from a File

```python
codecell
>>> jlink.flash_file('/path/to/file', address)
1337
>>> jlink.memory_read8(0, 1337)
[ 0, 0, .... ]
```

2.5 Flashing from a List of Bytes

```python
codecell
>>> data = [1, 2, 3, 4]
>>> jlink.flash(data, 0)
4
>>> jlink.memory_read8(0, 4)
[1, 2, 3, 4]
```

2.6 Unlocking a Device

**Note:** Currently unlock is only supported for Kinetis on SWD.

```python
codecell
>>> pylink.unlock(jlink, 'Kinetis')
True
```
PyLink ships with a command-line interface that provides common functionality. After you’ve installed the package, the command should be readily available for use.

Python interface for SEGGER J-Link.

```
usage: pylink [-h] [--version] [-v] {emulator,info,firmware,flash,unlock,erase,license} ...
```

Options:

```
--version          show program’s version number and exit
-v, --verbose      increase output verbosity
```

Sub-commands:

```
emulator query for information about emulators or support
```

Query for information about emulators or support.

```
usage: pylink emulator [-h] (-l [(usb,ip)] | -s SUPPORTED | -t)
```

Options:

```
-l, --list          list all the connected emulators
-s, --supported    query whether a device is supported
-t, --test          perform a self-test
```

```
info get information about the J-Link
```

Get information about the J-Link.

```
usage: pylink info [-h] [-p] [-j] [-s SERIAL_NO | -i IP_ADDR]
```

Options:

```
-p, --product       print the production information
-j, --jtag          print the JTAG pin status
-s, --serial        specify the J-Link serial number
-i, --ip_addr       J-Link IP address
```
firmware  modify the J-Link firmware
Modify the J-Link firmware.

usage: pylink firmware [-h] (-d | -u) [-s SERIAL_NO | -i IP_ADDR]

Options:
- d, --downgrade   downgrade the J-Link firmware
- u, --upgrade    upgrade the J-Link firmware
- s, --serial     specify the J-Link serial number
- i, --ip_addr    J-Link IP address

flash  flash a device connected to the J-Link
Flashes firmware from a file to a device connected to a J-Link.

usage: pylink flash [-h] [-a ADDR] -t {jtag,swd} -d DEVICE
   [-s SERIAL_NO | -i IP_ADDR]
   file

Positional arguments:
  file    file to flash onto device

Options:
- a, --addr     start address to flash from
- t, --tif      target interface (JTAG | SWD)
                  Possible choices: jtag, swd
- d, --device   specify the target device name
- s, --serial   specify the J-Link serial number
- i, --ip_addr  J-Link IP address

unlock unlock a connected device
Unlocks a device connected to a J-Link. Note that this will erase the device.

usage: pylink unlock [-h] -t {jtag,swd} -d DEVICE [-s SERIAL_NO | -i IP_ADDR]
   {kinetis}

Positional arguments:
  name    name of MCU to unlock
            Possible choices: kinetis

Options:
- t, --tif     target interface (JTAG | SWD)
              Possible choices: jtag, swd
- d, --device  specify the target device name
- s, --serial  specify the J-Link serial number
- i, --ip_addr J-Link IP address
erase erases the device connected to the J-Link

Erases the target device.

usage: pylink erase [-h] -t {jtag,swd} -d DEVICE [-s SERIAL_NO | -i IP_ADDR]

Options:

- -tif target interface (JTAG | SWD)
  Possible choices: jtag, swd
- -device specify the target device name
- -serial specify the J-Link serial number
- -ip_addr J-Link IP address

license manage the licenses of your J-Link

Manage the licenses of the J-Link.

usage: pylink license [-h] (-l | -a ADD | -e) [-s SERIAL_NO | -i IP_ADDR]

Options:

- -list list the licenses of the J-Link
- -add add a custom license to the J-Link
- -erase erase the custom licenses on the J-Link
- -serial specify the J-Link serial number
- -ip_addr J-Link IP address

Copyright 2017 Square, Inc.
The PyLink package provides a Pythonic interface for interacting with the J-Link C SDK. This interface is provided through the JLink class, which provides several of the functions provided by the native SDK. Some methods require a specific interface, a target being connected, or an emulator being connected, and will raise errors as appropriate if these conditions are not met.

In lieu of return codes, this library uses the object-oriented paradigm of raising an exception. All exceptions are inherited from the JLinkException base class.

### 4.1 Exceptions

This submodule defines the different exceptions that can be generated by the JLink methods.

- **exception** pylink.errors.JLinkDataException (code)
  
  Bases: pylink.enums.JLinkDataErrors, pylink.errors.JLinkException
  
  J-Link data event exception.

- **exception** pylink.errors.JLinkEraseException (code)
  
  Bases: pylink.enums.JLinkEraseErrors, pylink.errors.JLinkException
  
  J-Link erase exception.

- **exception** pylink.errors.JLinkException (code)
  
  Bases: pylink.enums.JLinkGlobalErrors, exceptions.Exception
  
  Generic J-Link exception.

- **exception** pylink.errors.JLinkFlashException (code)
  
  Bases: pylink.enums.JLinkFlashErrors, pylink.errors.JLinkException
  
  J-Link flash exception.

- **exception** pylink.errors.JLinkRTTException (code)
  
  Bases: pylink.enums.JLinkRTTErrors, pylink.errors.JLinkException
  
  J-Link RTT exception.

- **exception** pylink.errors.JLinkReadException (code)
  
  Bases: pylink.enums.JLinkReadErrors, pylink.errors.JLinkException
  
  J-Link read exception.

- **exception** pylink.errors.JLinkWriteException (code)
  
  Bases: pylink.enums.JLinkWriteErrors, pylink.errors.JLinkException
J-Link write exception.

## 4.2 Library

This submodule defines a `Library`. This is not needed unless explicitly specifying a different version of the J-Link dynamic library.

```python
class pylink.library.Library (dllpath=None)
Bases: object

Wrapper to provide easy access to loading the J-Link SDK DLL.

This class provides a convenience for finding and loading the J-Link DLL across multiple platforms, and accounting for the inconsistencies between Windows and nix-based platforms.

```_standard_calls_``
list of names of the methods for the API calls that must be converted to standard calling convention on the Windows platform.

```JLINK_SDK_NAME``
name of the J-Link DLL on nix-based platforms.

```WINDOWS_JLINK_SDK_NAME``
name of the J-Link DLL on Windows platforms.

```JLINK_SDK_NAME`` = 'libjlinkarm'

```WINDOWS_32_JLINK_SDK_NAME`` = 'JLinkARM'

```WINDOWS_64_JLINK_SDK_NAME`` = 'JLink_x64'

dll()
Returns the DLL for the underlying shared library.

**Parameters** `self` (*Library*) – the `Library` instance

**Returns** A `ctypes` DLL instance if one was loaded, otherwise `None`.

```classmethod find_library_darwin()``
Loaded the SEGGER DLL from the installed applications.

This method accounts for the all the different ways in which the DLL may be installed depending on the version of the DLL. Always uses the first directory found.

SEGGER’s DLL is installed in one of three ways dependent on which version of the SEGGER tools are installed:

<table>
<thead>
<tr>
<th>Versions</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;5.0.0</td>
<td>/Applications/SEGGER/JLink\ NUMBER</td>
</tr>
<tr>
<td>&lt;6.0.0</td>
<td>/Applications/SEGGER/JLink/libjlinkarm.major.minor.dylib</td>
</tr>
<tr>
<td>&gt;= 6.0.0</td>
<td>/Applications/SEGGER/JLink/libjlinkarm</td>
</tr>
</tbody>
</table>

**Parameters** `cls` (*Library*) – the `Library` class

**Returns** The path to the J-Link library files in the order they are found.

```classmethod find_library_linux()``
Loaded the SEGGER DLL from the root directory.

On Linux, the SEGGER tools are installed under the `/opt/SEGGER` directory with versioned directories having the suffix `_VERSION`.
**Parameters**  
`cls (Library)` – the Library class

**Returns**  
The paths to the J-Link library files in the order that they are found.

**classmethod find_library_windows()**  
Loads the SEGGER DLL from the windows installation directory.

*On Windows, these are found either under:*
- `C:\Program Files\SEGGER\JLink`  
- `C:\Program Files (x86)\SEGGER\JLink`.

**Parameters**  
`cls (Library)` – the Library class

**Returns**  
The paths to the J-Link library files in the order that they are found.

**classmethod get_appropriate_windows_sdk_name()**  
Returns the appropriate JLink SDK library name on Windows depending on 32bit or 64bit Python variant.

SEGGER delivers two variants of their dynamic library on Windows:
- `JLinkARM.dll` for 32-bit platform  
- `JLink_x64.dll` for 64-bit platform

**Parameters**  
`cls (Library)` – the Library class

**Returns**  
The name of the library depending on the platform this module is run on.

**load** *(path=None)*  
Loads the specified DLL, if any, otherwise re-loads the current DLL.

If `path` is specified, loads the DLL at the given `path`, otherwise re-loads the DLL currently specified by this library.

---

**Note:**  
This creates a temporary DLL file to use for the instance. This is necessary to work around a limitation of the J-Link DLL in which multiple J-Links cannot be accessed from the same process.

**Parameters**
- `self (Library)` – the Library instance  
- `path (path)` – path to the DLL to load

**Returns**  
True if library was loaded successfully.

**Raises**  
`OSError` – if there is no J-LINK SDK DLL present at the path.

**See also:**  
J-Link Multi-session.

**load_default()**  
Loads the default J-Link SDK DLL.

The default J-Link SDK is determined by first checking if `ctypes` can find the DLL, then by searching the platform-specific paths.

**Parameters**  
`self (Library)` – the Library instance

**Returns**  
True if the DLL was loaded, otherwise False.
unload()
Unloads the library’s DLL if it has been loaded.
This additionally cleans up the temporary DLL file that was created when the library was loaded.

Parameters self (Library) – the Library instance
Returns True if the DLL was unloaded, otherwise False.

4.3 JLock

This submodule defines a JLock. This acts as a lockfile-like interface for interacting with a particular emulator in order to prevent multiple threads or processes from creating instances of JLink to interact with the same emulator.

class pylink.jlock.JLock (serial_no)
Bases: object
Lockfile for accessing a particular J-Link.
The J-Link SDK does not prevent accessing the same J-Link multiple times from the same process or multiple processes. As a result, a user can have the same J-Link being accessed by multiple processes. This class provides an interface to a lock-file like structure for the physical J-Links to ensure that any instance of a JLink with an open emulator connection will be the only one accessing that emulator.
This class uses a PID-style lockfile to allow acquiring of the lockfile in the instances where the lockfile exists, but the process which created it is no longer running.
To share the same emulator connection between multiple threads, processes, or functions, a single instance of a JLink should be created and passed between the threads and processes.

name
the name of the lockfile.

path
full path to the lockfile.

fd
file description of the lockfile.

acquired
boolean indicating if the lockfile lock has been acquired.

IPADDR_NAME_FMT = '.pylink-ip-{}.lck'
SERIAL_NAME_FMT = '.pylink-usb-{}.lck'

acquire()
Attempts to acquire a lock for the J-Link lockfile.
If the lockfile exists but does not correspond to an active process, the lockfile is first removed, before an attempt is made to acquire it.

Parameters self (Jlock) – the JLock instance
Returns True if the lock was acquired, otherwise False.
Raises OSError – on file errors.

release()
Cleans up the lockfile if it was acquired.

Parameters self (JLock) – the JLock instance
Returns False if the lock was not released or the lock is not acquired, otherwise True.

4.4 JLink

This submodule provides the definition for the JLink class, which is the interface to the J-Link.

```python
class pylink.jlink.JLink(lib=None, log=None, detailed_log=None, error=None, warn=None, unsecure_hook=None, serial_no=None, ip_addr=None, open_tunnel=False)
```

Bases: object

Python interface for the SEGGER J-Link.

This is a wrapper around the J-Link C SDK to provide a Python interface to it. The shared library is loaded and used to call the SDK methods.

- **ADAPTIVE_JTAG_SPEED** = 65535
- **AUTO_JTAG_SPEED** = 0
- **INVALID_JTAG_SPEED** = 65534
- **MAX_BUF_SIZE** = 336
- **MAX_JTAG_SPEED** = 50000
- **MAX_NUM_CPU_REGISTERS** = 256
- **MAX_NUM_MOES** = 8
- **MIN_JTAG_SPEED** = 5

**add_license(** *args, **kwargs**)**

Adds the given contents as a new custom license to the J-Link.

**Parameters**

- **self**(JLink) – the JLink instance
- **contents** – the string contents of the new custom license

**Returns** True if license was added, False if license already existed.

**Raises** JLinkException – if the write fails.

Note: J-Link V9 and J-Link ULTRA/PRO V4 have 336 Bytes of memory for licenses, while older versions of 80 bytes.

**breakpoint_clear(** *args, **kwargs**)**

Removes a single breakpoint.

**Parameters**

- **self**(JLink) – the JLink instance
- **handle**(int) – the handle of the breakpoint to be removed

**Returns** True if the breakpoint was cleared, otherwise False if the breakpoint was not valid.

**breakpoint_clear_all(** *args, **kwargs**)**

Removes all breakpoints that have been set.

**Parameters** **self**(JLink) – the JLink instance
Returns True if they were cleared, otherwise False.

**breakpoint_find(***args, **kwargs**)
Returns the handle of a breakpoint at the given address, if any.

**Parameters**

- **self** (*JLink*) – the JLink instance
- **addr** (*int*) – the address to search for the breakpoint

**Returns** A non-zero integer if a breakpoint was found at the given address, otherwise zero.

**Note:** Either handle or index can be specified. If the index is not provided, the handle must be set, and vice-versa. If both index and handle are provided, the index overrides the provided handle.

**Parameters**

- **self** (*JLink*) – the JLink instance
- **handle** (*int*) – option handle of a valid breakpoint
- **index** (*int*) – optional index of the breakpoint.

**Returns** An instance of **JLinkBreakpointInfo** specifying information about the breakpoint.

**Raises**

- **JLinkException** – on error.
- **ValueError** – if both the handle and index are invalid.

**breakpoint_set(***args, **kwargs**)
Sets a breakpoint at the specified address.

If thumb is True, the breakpoint is set in THUMB-mode, while if arm is True, the breakpoint is set in ARM-mode, otherwise a normal breakpoint is set.

**Parameters**

- **self** (*JLink*) – the JLink instance
- **addr** (*int*) – the address where the breakpoint will be set
- **thumb** (*bool*) – boolean indicating to set the breakpoint in THUMB mode
- **arm** (*bool*) – boolean indicating to set the breakpoint in ARM mode

**Returns** An integer specifying the breakpoint handle. This handle should be retained for future breakpoint operations.

**Raises**

- **TypeError** – if the given address is not an integer.
- **JLinkException** – if the breakpoint could not be set.

**capabilities**
Returns a bitwise combination of the emulator’s capabilities.

**Parameters** **self** (*JLink*) – the JLink instance
Returns Bitfield of emulator capabilities.

clear_error()
Clears the DLL internal error state.

Parameters

- self (JLink) – the JLink instance

Returns The error state before the clear.

close()
Closes the open J-Link.

Parameters

- self (JLink) – the JLink instance

Returns None

Raises JLinkException – if there is no connected JLink.

code_memory_read(*args, **kwargs)
Reads bytes from code memory.

Note: This is similar to calling memory_read or memory_read8, except that this uses a cache and reads ahead. This should be used in instances where you want to read a small amount of bytes at a time, and expect to always read ahead.

Parameters

- self (JLink) – the JLink instance
- addr (int) – starting address from which to read
- num_bytes (int) – number of bytes to read

Returns A list of bytes read from the target.

Raises JLinkException – if memory could not be read.

comm_supported(*args, **kwargs)
Returns true if the connected emulator supports comm_* functions.

Parameters

- self (JLink) – the JLink instance

Returns True if the emulator supports comm_* functions, otherwise False.

compatible_firmware_version
Returns the DLL’s compatible J-Link firmware version.

Parameters

- self (JLink) – the JLink instance

Returns The firmware version of the J-Link that the DLL is compatible with.

 Raises JLinkException – on error.

compile_date
Returns a string specifying the date and time at which the DLL was translated.

Parameters

- self (JLink) – the JLink instance

Returns Datetime string.

connect(*args, **kwargs)
Connects the J-Link to its target.

Parameters
• `self (JLink)` – the JLink instance
• `chip_name (str)` – target chip name
• `speed (int)` – connection speed, one of {5-12000, 'auto', 'adaptive'}
• `verbose (bool)` – boolean indicating if connection should be verbose in logging

**Returns** None

**Raises**
- `JLinkException` – if connection fails to establish.
- `TypeError` – if given speed is invalid

**connected()**
Returns whether a J-Link is connected.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns** True if the J-Link is open and connected, otherwise False.

**connected_emulators (host=1)**
Returns a list of all the connected emulators.

**Parameters**
- `self (JLink)` – the JLink instance
- `host (int)` – host type to search (default: `JLinkHost.USB`)

**Returns** List of `JLinkConnectInfo` specifying the connected emulators.

**Raises** `JLinkException` – if fails to enumerate devices.

**connection_required (func)**
Decorator to specify that a target connection is required in order for the given method to be used.

**Parameters**
- `func (function)` – function being decorated

**Returns** The wrapper function.

**core_cpu (**args, **kwargs)**
Returns the identifier of the core CPU.

**Note:** This is distinct from the value returned from `core_id()` which is the ARM specific identifier.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns** The identifier of the CPU core.

**core_id (**args, **kwargs)**
Returns the identifier of the target ARM core.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns** Integer identifier of ARM core.

**core_name (**args, **kwargs)**
Returns the name of the target ARM core.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns** The target core’s name.
**coresight_configuration_required** *(func)*

Decorator to specify that a coresight configuration or target connection is required in order for the given method to be used.

**Parameters**

*func*(function) – function being decorated

**Returns**

The wrapper function.

**coresight_configure** *(*args, **kwargs)*

Prepares target and J-Link for CoreSight function usage.

**Parameters**

- **self** *(JLink)* – the JLink instance
- **ir_pre** *(int)* – sum of instruction register length of all JTAG devices in the JTAG chain, close to TDO than the actual one, that J-Link shall communicate with
- **dr_pre** *(int)* – number of JTAG devices in the JTAG chain, closer to TDO than the actual one, that J-Link shall communicate with
- **ir_post** *(int)* – sum of instruction register length of all JTAG devices in the JTAG chain, following the actual one, that J-Link shall communicate with
- **dr_post** *(int)* – Number of JTAG devices in the JTAG chain, following the actual one, J-Link shall communicate with
- **ir_len** *(int)* – instruction register length of the actual device that J-Link shall communicate with
- **perform_tif_init** *(bool)* – if False, then do not output switching sequence on completion

**Returns**

None

---

**Note:** This must be called before calling **coresight_read()** or **coresight_write()**.

---

**coresight_read** *(*args, **kwargs)*

Reads an Ap/DP register on a CoreSight DAP.

Wait responses and special handling are both handled by this method.

**Parameters**

- **self** *(JLink)* – the JLink instance
- **reg** *(int)* – index of DP/AP register to read
- **ap** *(bool)* – True if reading from an Access Port register, otherwise False for Debug Port

**Returns**

Data read from register.

**Raises**

JLinkException – on hardware error

---

**coresight_write** *(*args, **kwargs)*

Writes an Ap/DP register on a CoreSight DAP.
Note: coresight_configure() must be called prior to calling this method.

Parameters

- **self**(JLink) – the JLink instance
- **reg**(int) – index of DP/AP register to write
- **data**(int) – data to write
- **ap**(bool) – True if writing to an Access Port register, otherwise False for Debug Port

Returns

Number of repetitions needed until write request accepted.

Raises

JLinkException – on hardware error

```python
cp15_present(*args, **kwargs)
```

Returns whether target has CP15 co-processor.

Returns

True if the target has CP15 co-processor, otherwise False.

```python
cp15_register_read(*args, **kwargs)
```

Reads value from specified coprocessor register.

Parameters

- **cr_n**(int) – CRn value
- **op_1**(int) – Op1 value
- **cr_m**(int) – CRm value
- **op_2**(int) – Op2 value

Returns

An integer containing the value of coprocessor register

Raises

JLinkException – on error

```python
cp15_register_write(*args, **kwargs)
```

Writes value to specified coprocessor register.

Parameters

- **cr_n**(int) – CRn value
- **op_1**(int) – Op1 value
- **cr_m**(int) – CRm value
- **op_2**(int) – Op2 value
- **value**(int) – value to write

Returns

An integer containing the result of the command

Raises

JLinkException – on error

```python
cpu_capability(*args, **kwargs)
```

Checks whether the J-Link has support for a CPU capability.

This method checks if the emulator has built-in intelligence to handle the given CPU capability for the target CPU it is connected to.

Parameters

- **self**(JLink) – the JLink instance
• **capability** (*int*) – the capability to check for

Returns True if the J-Link has built-in intelligence to support the given capability for the CPU it is connected to, otherwise False.

cpu_halt_reasons(*args, **kwargs*)
Retrieves the reasons that the CPU was halted.

Parameters
  **self** (*JLink*) – the JLink instance

Returns A list of JLinkMOEInfo instances specifying the reasons for which the CPU was halted. This list may be empty in the case that the CPU is not halted.

Raises JLinkException – on hardware error.

cpu_speed(*args, **kwargs*)
Retrieves the CPU speed of the target.

If the target does not support CPU frequency detection, this function will return 0.

Parameters
  • **self** (*JLink*) – the JLink instance
  • **silent** (*bool*) – True if the CPU detection should not report errors to the error handler on failure.

Returns The measured CPU frequency on success, otherwise 0 if the core does not support CPU frequency detection.

 Raises JLinkException – on hardware error.

custom_licenses
Returns a string of the installed licenses the J-Link has.

Parameters
  **self** (*JLink*) – the JLink instance

Returns String of the contents of the custom licenses the J-Link has.

detailed_log_handler
Returns the detailed log handler function.

Parameters
  **self** (*JLink*) – the JLink instance

Returns None if the detailed log handler was not set, otherwise a ctypes.CFUNCTYPE.

device_family(*args, **kwargs*)
Returns the device family of the target CPU.

Parameters
  **self** (*JLink*) – the JLink instance

Returns Integer identifier of the device family.

disable_dialog_boxes(*args, **kwargs*)
Disables showing dialog boxes on certain methods.

**Warning:** This has the effect of also silencing dialog boxes that appear when updating firmware / to confirm updating firmware.

Dialog boxes will be shown for a brief period of time (approximately five seconds), before being automatically hidden, and the default option chosen.

Parameters
  **self** (*JLink*) – the JLink instance

Returns None
disable_reset_inits_registers(*args, **kwargs)
Disables CPU register initialization on resets.
When .reset() is called, the CPU registers will be read and not initialized.

Parameters
self (JLink) – the JLink instance

Returns
True if was previously enabled, otherwise False.

disable_reset_pulls_reset(*args, **kwargs)
Disables RESET pin toggling on the JTAG bus on resets.
When .reset() is called, it will not toggle the RESET pin on the JTAG bus.

Parameters
self (JLink) – the JLink instance

Returns
None

disable_reset_pulls_trst(*args, **kwargs)
Disables TRST pin toggling on the JTAG bus on resets.
When .reset() is called, it will not toggle the TRST pin on the JTAG bus.

Parameters
self (JLink) – the JLink instance

Returns
None

disable_soft_breakpoints(*args, **kwargs)
Disables software breakpoints.

Note: After this function is called, software_breakpoint_set() cannot be used without first calling enable_soft_breakpoints().

Parameters
self (JLink) – the JLink instance

Returns
None

disassemble_instruction(instruction)
Disassembles and returns the assembly instruction string.

Parameters

• self (JLink) – the JLink instance.

• instruction (int) – the instruction address.

Returns
A string corresponding to the assembly instruction string at the given instruction address.

Raises

• JLinkException – on error.

• TypeError – if instruction is not a number.

enable_dialog_boxes(*args, **kwargs)
Enables showing dialog boxes on certain methods.

Note: This can be used for batch or automated test running.

Parameters
self (JLink) – the JLink instance
Returns None

**enable_reset_inits_registers** (*args, **kwargs*)
Enables CPU register initialization on resets.
When `.reset()` is called, it will initialize the CPU registers.

Parameters **self** (*JLink*) – the JLink instance

Returns True if was previously enabled, otherwise False.

**enable_reset_pulls_reset** (*args, **kwargs*)
Enables RESET pin toggling on the JTAG bus on resets.
When `.reset()` is called, it will also toggle the RESET pin on the JTAG bus.

Parameters **self** (*JLink*) – the JLink instance

Returns None

**enable_reset_pulls_trst** (*args, **kwargs*)
Enables TRST pin toggling on the JTAG bus on resets.
When `.reset()` is called, it will also toggle the TRST pin on the JTAG bus.

Parameters **self** (*JLink*) – the JLink instance

Returns None

**enable_soft_breakpoints** (*args, **kwargs*)
Enables software breakpoints.

Parameters **self** (*JLink*) – the JLink instance

Returns None

**erase** (*args, **kwargs*)
Erases the flash contents of the device.
This erases the flash memory of the target device. If this method fails, the device may be left in an inoperable state.

Parameters **self** (*JLink*) – the JLink instance

Returns Number of bytes erased.

**erase_licenses** (*args, **kwargs*)
Erases the custom licenses from the connected J-Link.

Note: This method will erase all licenses stored on the J-Link.

Parameters **self** (*JLink*) – the JLink instance

Returns True on success, otherwise False.

**error**
DLL internal error state.
Parameters `self` (JLink) – the JLink instance

Returns The DLL internal error state. This is set if any error occurs in underlying DLL, otherwise it is None.

`error_handler`

Returns the error handler function.

Parameters `self` (JLink) – the JLink instance

Returns None if the error handler was not set, otherwise a ctypes.CFUNCTYPE.

`etm_register_read`(*args, **kwargs)

Reads a value from an ETM register.

Parameters

- `self` (JLink) – the JLink instance.
- `register_index` (int) – the register to read.

Returns The value read from the ETM register.

`etm_register_write`(*args, **kwargs)

Writes a value to an ETM register.

Parameters

- `self` (JLink) – the JLink instance.
- `register_index` (int) – the register to write to.
- `value` (int) – the value to write to the register.
- `delay` (bool) – boolean specifying if the write should be buffered.

Returns None

`etm_supported`(*args, **kwargs)

Returns if the CPU core supports ETM.

Parameters `self` (JLink) – the JLink instance.

Returns True if the CPU has the ETM unit, otherwise False.

`exec_command` (`cmd`)

Executes the given command.

This method executes a command by calling the DLL’s exec method. Direct API methods should be prioritized over calling this method.

Parameters

- `self` (JLink) – the JLink instance
- `cmd` (str) – the command to run

Returns The return code of running the command.

Raises JLinkException – if the command is invalid or fails.

See also:

For a full list of the supported commands, please see the SEGGER J-Link documentation, UM08001.

`extended_capabilities`

 Gets the capabilities of the connected emulator as a list.

Parameters `self` (JLink) – the JLink instance
Returns  List of 32 integers which define the extended capabilities based on their value and index within the list.

```python
def extended_capability(*args, **kwargs)
```

Checks if the emulator has the given extended capability.

**Parameters**
- `self (JLink)` – the JLink instance
- `capability (int)` – capability being queried

**Returns**  True if the emulator has the given extended capability, otherwise False.

```python
features
```

Returns a list of the J-Link embedded features.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns**  `[ 'RDI', 'FlashBP', 'FlashDL', 'JFlash', 'GDB' ]`

**Return type**  A list of strings, each a feature. Example

```python
firmware_newer(*args, **kwargs)
```

Returns whether the J-Link’s firmware version is newer than the one that the DLL is compatible with.

**Note:** This is not the same as calling `not jlink.firmware_outdated()`.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns**  True if the J-Link’s firmware is newer than the one supported by the DLL, otherwise False.

```python
firmware_outdated(*args, **kwargs)
```

Returns whether the J-Link’s firmware version is older than the one that the DLL is compatible with.

**Note:** This is not the same as calling `not jlink.firmware_newer()`.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns**  True if the J-Link’s firmware is older than the one supported by the DLL, otherwise False.

```python
firmware_version
```

Returns a firmware identification string of the connected J-Link.

**It consists of the following:**
- Product Name (e.g. J-Link)
- The string: compiled
- Compile data and time.
- Optional additional information.

**Parameters**
- `self (JLink)` – the JLink instance

**Returns**  Firmware identification string.
flash(*args, **kwargs)
    Flashes the target device.

    The given on_progress callback will be called as on_progress(action, progress_string, percentage)
    periodically as the data is written to flash. The action is one of Compare, Erase, Verify, Flash.

    Parameters
    • self (JLink) – the JLink instance
    • data (list) – list of bytes to write to flash
    • addr (int) – start address on flash which to write the data
    • on_progress (function) – callback to be triggered on flash progress
    • power_on (boolean) – whether to power the target before flashing
    • flags (int) – reserved, do not use

    Returns  Number of bytes flashed. This number may not necessarily be equal to len(data),
    but that does not indicate an error.

    Raises  JLinkException – on hardware errors.

flash_file(*args, **kwargs)
    Flashes the target device.

    The given on_progress callback will be called as on_progress(action, progress_string, percentage)
    periodically as the data is written to flash. The action is one of Compare, Erase, Verify, Flash.

    Parameters
    • self (JLink) – the JLink instance
    • path (str) – absolute path to the source file to flash
    • addr (int) – start address on flash which to write the data
    • on_progress (function) – callback to be triggered on flash progress
    • power_on (boolean) – whether to power the target before flashing

    Returns  Integer value greater than or equal to zero. Has no significance.

    Raises  JLinkException – on hardware errors.

flash_write(*args, **kwargs)
    Writes data to the flash region of a device.

    The given number of bits, if provided, must be either 8, 16, or 32.

    Parameters
    • self (JLink) – the JLink instance
    • addr (int) – starting flash address to write to
    • data (list) – list of data units to write
    • nbits (int) – number of bits to use for each unit

    Returns  Number of bytes written to flash.
• `self` (`JLink`) – the JLink instance
• `addr` (`int`) – starting flash address to write to
• `data` (`list`) – list of halfwords to write

Returns Number of bytes written to flash.

`flash_write32` (*args, **kwargs)
Writes words to the flash region of a device.

Parameters
• `self` (`JLink`) – the JLink instance
• `addr` (`int`) – starting flash address to write to
• `data` (`list`) – list of words to write

Returns Number of bytes written to flash.

`flash_write8` (*args, **kwargs)
Writes bytes to the flash region of a device.

Parameters
• `self` (`JLink`) – the JLink instance
• `addr` (`int`) – starting flash address to write to
• `data` (`list`) – list of bytes to write

Returns Number of bytes written to flash.

`get_device_index` (`chip_name`)  
Finds index of device with chip name

Parameters
• `self` (`JLink`) – the JLink instance
• `chip_name` (`str`) – target chip name

Returns Index of the device with the matching chip name.

Raises JLinkException – if chip is unsupported.

`gpio_get` (*args, **kwargs)
Returns a list of states for the given pins.

Defaults to the first four pins if an argument is not given.

Parameters
• `self` (`JLink`) – the JLink instance
• `pins` (`list`) – indices of the GPIO pins whose states are requested

Returns A list of states.

Raises JLinkException – on error.

`gpio_properties` (*args, **kwargs)
Returns the properties of the user-controllable GPIOs.

Provided the device supports user-controllable GPIOs, they will be returned by this method.

Parameters `self` (`JLink`) – the JLink instance
**gpio_set** (*args, **kwargs)
Sets the state for one or more user-controllable GPIOs.

For each of the given pins, sets the corresponding state based on the index.

**Parameters**
- `self` (JLink) – the JLink instance
- `pins` (list) – list of GPIO indices
- `states` (list) – list of states to set

**Returns** A list of updated states.

**Raises**
- JLinkException – on error.
- ValueError – if `len(pins) != len(states)`

**halt** (*args, **kwargs)
Halts the CPU Core.

**Parameters**
- `self` (JLink) – the JLink instance

**Returns** True if halted, False otherwise.

**halted** (*args, **kwargs)
Returns whether the CPU core was halted.

**Parameters**
- `self` (JLink) – the JLink instance

**Returns** True if the CPU core is halted, otherwise False.

**Raises** JLinkException – on device errors.

**hardware_breakpoint_set** (*args, **kwargs)
Sets a hardware breakpoint at the specified address.

If thumb is True, the breakpoint is set in THUMB-mode, while if arm is True, the breakpoint is set in ARM-mode, otherwise a normal breakpoint is set.

**Parameters**
- `self` (JLink) – the JLink instance
- `addr` (int) – the address where the breakpoint will be set
- `thumb` (bool) – boolean indicating to set the breakpoint in THUMB mode
- `arm` (bool) – boolean indicating to set the breakpoint in ARM mode

**Returns** An integer specifying the breakpoint handle. This handle should be retained for future breakpoint operations.

**Raises**
- TypeError – if the given address is not an integer.
- JLinkException – if the breakpoint could not be set.
**hardware_info**

Returns a list of 32 integer values corresponding to the bitfields specifying the power consumption of the target.

The values returned by this function only have significance if the J-Link is powering the target.

**The words, indexed, have the following significance:**

0. If 1, target is powered via J-Link.
1. Overcurrent bitfield: 0: No overcurrent. 1: Overcurrent happened. 2ms @ 3000mA 2: Overcurrent happened. 10ms @ 1000mA 3: Overcurrent happened. 40ms @ 400mA
2. Power consumption of target (mA).
3. Peak of target power consumption (mA).
4. Peak of target power consumption during J-Link operation (mA).

**Parameters**

- `self (JLink)` – the JLink instance
- `mask (int)` – bit mask to decide which hardware information words are returned (defaults to all the words).

**Returns** List of bitfields specifying different states based on their index within the list and their value.

**Raises** JLinkException – on hardware error.

**hardware_status**

Retrieves and returns the hardware status.

**Parameters**

- `self (JLink)` – the JLink instance

**Returns** A JLinkHardwareStatus describing the J-Link hardware.

**hardware_version**

Returns the hardware version of the connected J-Link as a major.minor string.

**Parameters**

- `self (JLink)` – the JLink instance

**Returns** Hardware version string.

**ice_register_read(**args**, **kwargs)**

Reads a value from an ARM ICE register.

**Parameters**

- `self (JLink)` – the JLink instance
- `register_index (int)` – the register to read

**Returns** The value read from the register.

**ice_register_write(**args**, **kwargs)**

Writes a value to an ARM ICE register.

**Parameters**

- `self (JLink)` – the JLink instance
- `register_index (int)` – the ICE register to write to
- `value (int)` – the value to write to the ICE register
• **delay**(bool) – boolean specifying if the write should be delayed

  **Returns** None

**index**
Retrieves and returns the index number of the actual selected J-Link.

  **Parameters** self (JLink) – the JLink instance

  **Returns** Index of the currently connected J-Link.

**interface_required**(interface)
Decorator to specify that a particular interface type is required for the given method to be used.

  **Parameters** interface (int) – attribute of JLinkInterfaces

  **Returns** A decorator function.

**invalidate_firmware**(*args, **kwargs)
Invalidates the emulator’s firmware.

  This method is useful for downgrading the firmware on an emulator. By calling this method, the current emulator’s firmware is invalidated, which will make the emulator download the firmware of the J-Link SDK DLL that this instance was created with.

  **Parameters** self (JLink) – the JLink instance

  **Returns** None

  **Raises** JLinkException – on hardware error.

**ir_len**(*args, **kwargs)
Counts and returns the total length of instruction registers of all the devices in the JTAG scan chain.

  **Parameters** self (JLink) – the JLink instance

  **Returns** Total instruction register length.

**jtag_configure**(*args, **kwargs)
Configures the JTAG scan chain to determine which CPU to address.

  Must be called if the J-Link is connected to a JTAG scan chain with multiple devices.

  **Parameters**

  • self (JLink) – the JLink instance

  • instr_regs (int) – length of instruction registers of all devices closer to TD1 then the addressed CPU

  • data_bits (int) – total number of data bits closer to TD1 than the addressed CPU

  **Returns** None

  **Raises** ValueError – if instr_regs or data_bits are not natural numbers

**jtag_create_clock**(*args, **kwargs)
Creates a JTAG clock on TCK.

  **Parameters**

  • self (JLink) – the JLink instance

  **Returns** either 0 or 1.

---

**Note:** This function only needs to be called once.
**Return type** The state of the TDO pin

```python
jtag_flush(*args, **kwargs)
```
Flushes the internal JTAG buffer.

**Note:** The buffer is automatically flushed when a response from the target is expected, or the buffer is full. This can be used after a `memory_write()` in order to flush the buffer.

**Parameters**

- `self (JLink)` – the JLink instance

**Returns** None

```python
jtag_send(*args, **kwargs)
```
Sends data via JTAG.

Sends data via JTAG on the rising clock edge, TCK. At each clock edge, on bit is transferred in from TDI and out to TDO. The clock uses the TMS to step through the standard JTAG state machine.

**Parameters**

- `self (JLink)` – the JLink instance
- `tms (int)` – used to determine the state transitions for the Test Access Port (TAP) controller from its current state
- `tdi (int)` – input data to be transferred in from TDI to TDO
- `num_bits (int)` – a number in the range [1, 32] inclusively specifying the number of meaningful bits in the tms and tdi parameters for the purpose of extracting state and data information

**Returns** None

**Raises** `ValueError` if `num_bits < 1` or `num_bits > 32`.

**See also:**

- JTAG Technical Overview.

```python
licenses
```
Returns a string of the built-in licenses the J-Link has.

**Parameters**

- `self (JLink)` – the JLink instance

**Returns** String of the contents of the built-in licenses the J-Link has.

```python
log_handler
```
Returns the log handler function.

**Parameters**

- `self (JLink)` – the JLink instance

**Returns** None if the log handler was not set, otherwise a `ctypes.CFUNCTYPE`.

```python
memory_read(*args, **kwargs)
```
Reads memory from a target system or specific memory zone.

The optional `zone` specifies a memory zone to access to read from, e.g., IDATA, DDATA, or CODE.

The given number of bits, if provided, must be either 8, 16, or 32. If not provided, always reads `num_units` bytes.

**Parameters**
• **self** (*JLink*) – the JLink instance
• **addr** (*int*) – start address to read from
• **num_units** (*int*) – number of units to read
• **zone** (*str*) – optional memory zone name to access
• **nbits** (*int*) – number of bits to use for each unit

**Returns** List of units read from the target system.

**Raises**
• **JLinkException** – if memory could not be read.
• **ValueError** – if `nbits` is not `None`, and not in `8`, `16`, or `32`.

**memory_read16** (*args, **kwargs*)
Reads memory from the target system in units of 16-bits.

**Parameters**
• **self** (*JLink*) – the JLink instance
• **addr** (*int*) – start address to read from
• **num_halfwords** (*int*) – number of half words to read
• **zone** (*str*) – memory zone to read from

**Returns** List of halfwords read from the target system.

**Raises** **JLinkException** – if memory could not be read

**memory_read32** (*args, **kwargs*)
Reads memory from the target system in units of 32-bits.

**Parameters**
• **self** (*JLink*) – the JLink instance
• **addr** (*int*) – start address to read from
• **num_words** (*int*) – number of words to read
• **zone** (*str*) – memory zone to read from

**Returns** List of words read from the target system.

**Raises** **JLinkException** – if memory could not be read

**memory_read64** (*args, **kwargs*)
Reads memory from the target system in units of 64-bits.

**Parameters**
• **self** (*JLink*) – the JLink instance
• **addr** (*int*) – start address to read from
• **num_long_words** (*int*) – number of long words to read

**Returns** List of long words read from the target system.

**Raises** **JLinkException** – if memory could not be read

**memory_read8** (*args, **kwargs*)
Reads memory from the target system in units of bytes.
Parameters

- `self(JLink)` – the JLink instance
- `addr(int)` – start address to read from
- `num_bytes(int)` – number of bytes to read
- `zone(str)` – memory zone to read from

Returns List of bytes read from the target system.

Raises `JLinkException` – if memory could not be read.

```python
memory_write(*args, **kwargs)
```

Writes memory to a target system or specific memory zone.

The optional `zone` specifies a memory zone to access to write to, e.g. IDATA, DDATA, or CODE.

The given number of bits, if provided, must be either 8, 16, or 32.

Parameters

- `self(JLink)` – the JLink instance
- `addr(int)` – start address to write to
- `data(list)` – list of data units to write
- `zone(str)` – optional memory zone name to access
- `nbits(int)` – number of bits to use for each unit

Returns Number of units written.

Raises

- `JLinkException` – on write hardware failure.
- `ValueError` – if `nbits` is not None and not in 8, 16 or 32.

```python
memory_write16(*args, **kwargs)
```

Writes half-words to memory of a target system.

Parameters

- `self(JLink)` – the JLink instance
- `addr(int)` – start address to write to
- `data(list)` – list of half-words to write
- `zone(str)` – optional memory zone to access

Returns Number of half-words written to target.

Raises `JLinkException` – on memory access error.

```python
memory_write32(*args, **kwargs)
```

Writes words to memory of a target system.

Parameters

- `self(JLink)` – the JLink instance
- `addr(int)` – start address to write to
- `data(list)` – list of words to write
- `zone(str)` – optional memory zone to access
Returns Number of words written to target.

Raises JLinkException – on memory access error.

memory_write64 (*args, **kwargs)

Writes long words to memory of a target system.

Note: This is little-endian.

Parameters

• self (JLink) – the JLink instance
• addr (int) – start address to write to
• data (list) – list of long words to write
• zone (str) – optional memory zone to access

Returns Number of long words written to target.

Raises JLinkException – on memory access error.

memory_write8 (*args, **kwargs)

Writes bytes to memory of a target system.

Parameters

• self (JLink) – the JLink instance
• addr (int) – start address to write to
• data (list) – list of bytes to write
• zone (str) – optional memory zone to access

Returns Number of bytes written to target.

Raises JLinkException – on memory access error.

memory_zones (*args, **kwargs)

Gets all memory zones supported by the current target.

Some targets support multiple memory zones. This function provides the ability to get a list of all the memory zones to facilitate using the memory zone routing functions.

Parameters self (JLink) – the JLink instance

Returns A list of all the memory zones as JLinkMemoryZone structures.

Raises JLinkException – on hardware errors.

minimum_required (version)

Decorator to specify the minimum SDK version required.

Parameters version (str) – valid version string

Returns A decorator function.

num_active_breakpoints (*args, **kwargs)

Returns the number of currently active breakpoints.

Parameters self (JLink) – the JLink instance

Returns The number of breakpoints that are currently set.
num_active_watchpoints(*args, **kwargs)
Returns the number of currently active watchpoints.

Parameters self (JLink) – the JLink instance

Returns The number of watchpoints that are currently set.

num_available_breakpoints(*args, **kwargs)
Returns the number of available breakpoints of the specified type.
If arm is set, gets the number of available ARM breakpoint units. If thumb is set, gets the number of available THUMB breakpoint units. If ram is set, gets the number of available software RAM breakpoint units. If flash is set, gets the number of available software flash breakpoint units. If hw is set, gets the number of available hardware breakpoint units.
If a combination of the flags is given, then num_available_breakpoints() returns the number of breakpoints specified by the given flags. If no flags are specified, then the count of available breakpoint units is returned.

Parameters
• self (JLink) – the JLink instance
• arm (bool) – Boolean indicating to get number of ARM breakpoints.
• thumb (bool) – Boolean indicating to get number of THUMB breakpoints.
• ram (bool) – Boolean indicating to get number of SW RAM breakpoints.
• flash (bool) – Boolean indicating to get number of Flash breakpoints.
• hw (bool) – Boolean indicating to get number of Hardware breakpoints.

Returns The number of available breakpoint units of the specified type.

num_available_watchpoints(*args, **kwargs)
Returns the number of available watchpoints.

Parameters self (JLink) – the JLink instance

Returns The number of watchpoints that are available to be set.

num_connected_emulators()
Returns the number of emulators which are connected via USB to the host.

Parameters self (JLink) – the JLink instance

Returns The number of connected emulators.

num_memory_zones(*args, **kwargs)
Returns the number of memory zones supported by the target.

Parameters self (JLink) – the JLink instance

Returns An integer count of the number of memory zones supported by the target.

Raises JLinkException – on error.

num_supported_devices()
Returns the number of devices that are supported by the opened J-Link DLL.

Parameters self (JLink) – the JLink instance

Returns Number of devices the J-Link DLL supports.

oem
Retrieves and returns the OEM string of the connected J-Link.
Parameters `self` (`JLink`) – the JLink instance

Returns The string of the OEM. If this is an original SEGGER product, then `None` is returned instead.

Raises `JLinkException` – on hardware error.

`open` (`serial_no=None, ip_addr=None`)  
Connects to the J-Link emulator (defaults to USB).

If `serial_no` and `ip_addr` are both given, this function will connect to the J-Link over TCP/IP.

Parameters

- `self` (`JLink`) – the JLink instance
- `serial_no` (`int`) – serial number of the J-Link
- `ip_addr` (`str`) – IP address and port of the J-Link (e.g. 192.168.1.1:80)

Returns `None`

Raises

- `JLinkException` – if fails to open (i.e. if device is unplugged)
- `TypeError` – if `serial_no` is present, but not `int` coercible.
- `AttributeError` – if `serial_no` and `ip_addr` are both `None`.

`open_required` (`func`)  
Decorator to specify that the J-Link DLL must be opened, and a J-Link connection must be established.

Parameters `func` (`function`) – function being decorated

Returns The wrapper function.

`open_tunnel` (`serial_no, port=19020`)  
Connects to the J-Link emulator (over SEGGER tunnel).

Parameters

- `self` (`JLink`) – the JLink instance
- `serial_no` (`int`) – serial number of the J-Link
- `port` (`int`) – optional port number (default to 19020).

Returns `None`

`opened` ()  
Returns whether the DLL is open.

Parameters `self` (`JLink`) – the JLink instance

Returns `True` if the J-Link is open, otherwise `False`.

`power_off` (*args, **kwargs)  
Turns off the power supply over pin 19 of the JTAG connector.

If given the optional `default` parameter, deactivates the power supply by default.

Parameters

- `self` (`JLink`) – the JLink instance
- `default` (`bool`) – boolean indicating if to set power off by default

Returns The current JLink instance
Raises `JLinkException` – if J-Link does not support powering the target.

```python
power_on(*args, **kwargs)
```

Turns on the power supply over pin 19 of the JTAG connector.

If given the optional `default` parameter, activates the power supply by default.

**Parameters**

- `self (JLink)` – the JLink instance
- `default (bool)` – boolean indicating if to set power by default

**Returns** None

Raises `JLinkException` – if J-Link does not support powering the target.

```python
product_name
```

Returns the product name of the connected J-Link.

**Parameters** `self (JLink)` – the JLink instance

**Returns** Product name.

```python
register_list(*args, **kwargs)
```

Returns a list of the indices for the CPU registers.

The returned indices can be used to read the register content or grab the register name.

**Parameters** `self (JLink)` – the JLink instance

**Returns** List of registers.

```python
register_name(*args, **kwargs)
```

Retrives and returns the name of an ARM CPU register.

**Parameters**

- `self (JLink)` – the JLink instance
- `register_index (int)` – index of the register whose name to retrieve

**Returns** Name of the register.

```python
register_read(*args, **kwargs)
```

Reads the value from the given register.

**Parameters**

- `self (JLink)` – the JLink instance
- `register_index (int/str)` – the register to read

**Returns** The value stored in the given register.

```python
register_read_multiple(*args, **kwargs)
```

Retrieves the values from the registers specified.

**Parameters**

- `self (JLink)` – the JLink instance
- `register_indices (list)` – list of registers to read

**Returns** A list of values corresponding one-to-one for each of the given register indices. The returned list of values are the values in order of which the indices were specified.

Raises `JLinkException` – if a given register is invalid or an error occurs.
**register_write**(*args, **kwargs*)
Writes into an ARM register.

**Note:** The data is not immediately written, but is cached before being transferred to the CPU on CPU start.

**Parameters**

- **self** (*JLink*) – the JLink instance
- **reg_index** (*int/str*) – the ARM register to write to
- **value** (*int*) – the value to write to the register

**Returns** The value written to the ARM register.

**Raises** *JLinkException* – on write error.

**register_write_multiple**(*args, **kwargs*)
Writes to multiple CPU registers.

Writes the values to the given registers in order. There must be a one-to-one correspondence between the values and the registers specified.

**Parameters**

- **self** (*JLink*) – the JLink instance
- **register_indices** (*list*) – list of registers to write to
- **values** (*list*) – list of values to write to the registers

**Returns** None

**Raises**
- *ValueError* – if len(register_indices) != len(values)
- *JLinkException* – if a register could not be written to or on error

**reset**(*args, **kwargs*)
Resets the target.

This method resets the target, and by default toggles the RESET and TRST pins.

**Parameters**

- **self** (*JLink*) – the JLink instance
- **ms** (*int*) – Amount of milliseconds to delay after reset (default: 0)
- **halt** (*bool*) – if the CPU should halt after reset (default: True)

**Returns** Number of bytes read.

**reset_tap**(*args, **kwargs*)
Resets the TAP controller via TRST.

**Note:** This must be called at least once after power up if the TAP controller is to be used.

**Parameters** **self** (*JLink*) – the JLink instance
Returns None

restart(*args, **kwargs)
Restart the CPU core and simulates/emulates instructions.

Note: This is a no-op if the CPU isn’t halted.

Parameters
• self (JLink) – the JLink instance
• num_instructions (int) – number of instructions to simulate, defaults to zero
• skip_breakpoints (bool) – skip current breakpoint (default: False)

Returns True if device was restarted, otherwise False.

Raises ValueError – if instruction count is not a natural number.

rtt_control(*args, **kwargs)
Issues an RTT Control command.

All RTT control is done through a single API call which expects specifically laid-out configuration structures.

Parameters
• self (JLink) – the JLink instance
• command (int) – the command to issue (see enums.JLinkRTTCommand)
• config (ctypes type) – the configuration to pass by reference.

Returns An integer containing the result of the command.

Raises JLinkRTTException – on error.

rtt_get_buf_descriptor(*args, **kwargs)
After starting RTT, get the descriptor for an RTT control block.

Parameters
• self (JLink) – the JLink instance
• buffer_index (int) – the index of the buffer to get.
• up (bool) – True if buffer is an UP buffer, otherwise False.

Returns JLinkRTTerminalBufDesc describing the buffer.

Raises JLinkRTTException – if the RTT control block has not yet been found.

rtt_get_num_down_buffers(*args, **kwargs)
After starting RTT, get the current number of down buffers.

Parameters self (JLink) – the JLink instance

Returns The number of configured down buffers on the target.

Raises JLinkRTTException – if the underlying JLINK_RTTERMINAL_Control call fails.

rtt_get_num_up_buffers(*args, **kwargs)
After starting RTT, get the current number of up buffers.

Parameters self (JLink) – the JLink instance
Returns The number of configured up buffers on the target.

Raises JLinkRTTException – if the underlying JLINK_RTTERMINAL_Control call fails.

```
rtt_get_status(*args, **kwargs)
```

After starting RTT, get the status.

Parameters

- `self (JLink)` – the JLink instance

Returns The status of RTT.

Raises JLinkRTTException – on error.

```
rtt_read(*args, **kwargs)
```

Reads data from the RTT buffer.

This method will read at most num_bytes bytes from the specified RTT buffer. The data is automatically removed from the RTT buffer. If there are not num_bytes bytes waiting in the RTT buffer, the entire contents of the RTT buffer will be read.

Parameters

- `self (JLink)` – the JLink instance
- `buffer_index (int)` – the index of the RTT buffer to read from
- `num_bytes (int)` – the maximum number of bytes to read

Returns A list of bytes read from RTT.

Raises JLinkRTTException – if the underlying JLINK_RTTERMINAL_Read call fails.

```
rtt_start(*args, **kwargs)
```

Starts RTT processing, including background read of target data.

Parameters

- `self (JLink)` – the JLink instance
- `block_address (int)` – optional configuration address for the RTT block

Returns None

Raises JLinkRTTException – if the underlying JLINK_RTTERMINAL_Control call fails.

```
rtt_stop(*args, **kwargs)
```

Stops RTT on the J-Link and host side.

Parameters

- `self (JLink)` – the JLink instance

Returns None

Raises JLinkRTTException – if the underlying JLINK_RTTERMINAL_Control call fails.

```
rtt_write(*args, **kwargs)
```

Writes data to the RTT buffer.

This method will write at most len(data) bytes to the specified RTT buffer.

Parameters

- `self (JLink)` – the JLink instance
- `buffer_index (int)` – the index of the RTT buffer to write to
- `data (list)` – the list of bytes to write to the RTT buffer

Returns The number of bytes successfully written to the RTT buffer.
PyLink Documentation, Release 0.11.1

Raises JLinkRTEException – if the underlying JLINK_RTTERMINAL_Write call fails.

scan_chain_len(*args, **kwargs)
  Retrieves and returns the number of bits in the scan chain.

Parameters
  * self (JLink) – the JLink instance
  * scan_chain (int) – scan chain to be measured

Returns Number of bits in the specified scan chain.

Raises JLinkException – on error.

scan_len(*args, **kwargs)
  Retrieves and returns the length of the scan chain select register.

Parameters self (JLink) – the JLink instance

Returns Length of the scan chain select register.

serial_number
  Returns the serial number of the connected J-Link.

Parameters self (JLink) – the JLink instance

Returns Serial number as an integer.

set_big_endian(*args, **kwargs)
  Sets the target hardware to big endian.

Parameters self (JLink) – the JLink instance

Returns True if target was little endian before call, otherwise False.

set_etb_trace(*args, **kwargs)
  Sets the trace source to ETB.

Parameters self (JLink) – the JLink instance.

Returns None

set_etm_trace(*args, **kwargs)
  Sets the trace source to ETM.

Parameters self (JLink) – the JLink instance.

Returns None

set_little_endian(*args, **kwargs)
  Sets the target hardware to little endian.

Parameters self (JLink) – the JLink instance

Returns True if target was big endian before call, otherwise False.

set_log_file(*args, **kwargs)
  Sets the log file output path. see https://wiki.segger.com/Enable_J-Link_log_file

Parameters
  * self (JLink) – the JLink instance
  * file_path (str) – the file path where the log file will be stored

Returns None

Raises JLinkException – if the path specified is invalid.
**set_max_speed**(*args, **kwargs*)
Sets JTAG communication speed to the maximum supported speed.

Parameters
self (JLink) – the JLink instance

Returns
None

**set_reset_pin_high**(*args, **kwargs*)
Sets the reset pin high.

Parameters
self (JLink) – the JLink instance

Returns
None

**set_reset_pin_low**(*args, **kwargs*)
Sets the reset pin low.

Parameters
self (JLink) – the JLink instance

Returns
None

**set_reset_strategy**(*args, **kwargs*)
Sets the reset strategy for the target.

The reset strategy defines what happens when the target is reset.

Parameters
• self (JLink) – the JLink instance
• strategy (int) – the reset strategy to use

Returns
The previous reset strategy.

**set_speed**(*args, **kwargs*)
Sets the speed of the JTAG communication with the ARM core.

If no arguments are present, automatically detects speed.

If a speed is provided, the speed must be no larger than JLink.MAX_JTAG_SPEED and no smaller than JLink.MIN_JTAG_SPEED. The given speed can also not be JLink.INVALID_JTAG_SPEED.

Parameters
• self (JLink) – the JLink instance
• speed (int) – the speed in kHz to set the communication at
• auto (bool) – automatically detect correct speed
• adaptive (bool) – select adaptive clocking as JTAG speed

Returns
None

Raises
• TypeError – if given speed is not a natural number.
• ValueError – if given speed is too high, too low, or invalid.

**set_tck_pin_high**(*args, **kwargs*)
Sets the TCK pin to the high value (1).

Parameters
self (JLink) – the JLink instance

Returns
None

Raises
JLinkException – if the emulator does not support this feature.
**set_tck_pin_low**(*args, **kwargs*)

Sets the TCK pin to the low value (0).

**Parameters**

- **self** (JLink) – the JLink instance

**Returns**

None

**Raises**

JLinkException – if the emulator does not support this feature.

**set_tdi_pin_high**(*args, **kwargs*)

Sets the test data input to logical 1.

**Parameters**

- **self** (JLink) – the JLink instance

**Returns**

None

**set_tdi_pin_low**(*args, **kwargs*)

Clears the test data input.

TDI is set to logical 0 (Ground).

**Parameters**

- **self** (JLink) – the JLink instance

**Returns**

None

**set_tif**(*args, **kwargs*)

Selects the specified target interface.

Note that a restart must be triggered for this to take effect.

**Parameters**

- **self** (JLink) – the JLink instance
- **interface** (int) – integer identifier of the interface

**Returns**

True if target was updated, otherwise False.

**Raises**

JLinkException – if the given interface is invalid or unsupported.

**set_tms_pin_high**(*args, **kwargs*)

Sets the test mode select to logical 1.

**Parameters**

- **self** (JLink) – the JLink instance

**Returns**

None

**set_tms_pin_low**(*args, **kwargs*)

Clears the test mode select.

TMS is set to logical 0 (Ground).

**Parameters**

- **self** (JLink) – the JLink instance

**Returns**

None

**set_trace_source**(*args, **kwargs*)

Sets the source to be used for tracing.

The source must be one of the ones provided by enums.JLinkTraceSource.

**Parameters**

- **self** (JLink) – the JLink instance.
- **source** (int) – the source to use.

**Returns**

None
set_trst_pin_high(*args, **kwargs)
Sets the TRST pin to high (1).
Deasserts the TRST pin.

Parameters
self (JLink) – the JLink instance

Returns None

set_trst_pin_low(*args, **kwargs)
Sets the TRST pin to low (0).
This asserts the TRST pin.

Parameters
self (JLink) – the JLink instance

Returns None

set_vector_catch(*args, **kwargs)
Sets vector catch bits of the processor.
The CPU will jump to a vector if the given vector catch is active, and will enter a debug state. This has the effect of halting the CPU as well, meaning the CPU must be explicitly restarted.

Parameters
self (JLink) – the JLink instance

Returns None

Raises
JLinkException – on error.

software_breakpoint_set(*args, **kwargs)
Sets a software breakpoint at the specified address.
If thumb is True, the breakpoint is set in THUMB-mode, while if arm is True, the breakpoint is set in ARM-mode, otherwise a normal breakpoint is set.
If flash is True, the breakpoint is set in flash, otherwise if ram is True, the breakpoint is set in RAM. If both are True or both are False, then the best option is chosen for setting the breakpoint in software.

Parameters
• self (JLink) – the JLink instance
• addr (int) – the address where the breakpoint will be set
• thumb (bool) – boolean indicating to set the breakpoint in THUMB mode
• arm (bool) – boolean indicating to set the breakpoint in ARM mode
• flash (bool) – boolean indicating to set the breakpoint in flash
• ram (bool) – boolean indicating to set the breakpoint in RAM

Returns An integer specifying the breakpoint handle. This handle should be retained for future breakpoint operations.

Raises
• TypeError – if the given address is not an integer.
• JLinkException – if the breakpoint could not be set.

speed
Returns the current JTAG connection speed.

Parameters
self (JLink) – the JLink instance

Returns JTAG connection speed.
speed_info
Retrieves information about supported target interface speeds.

Parameters self (JLink) – the JLink instance

Returns The JLinkSpeedInfo instance describing the supported target interface speeds.

step(*args, **kwargs)
Executes a single step.
Steps even if there is a breakpoint.

Parameters
• self (JLink) – the JLink instance
• thumb (bool) – boolean indicating if to step in thumb mode

Returns None
Raises JLinkException – on error

strace_clear(*args, **kwargs)
Clears the trace event specified by the given handle.

Parameters
• self (JLink) – the JLink instance.
• handle (int) – handle of the trace event.

Returns None
Raises JLinkException – on error.

strace_clear_all(*args, **kwargs)
Clears all STRACE events.

Parameters self (JLink) – the JLink instance.

Returns None
Raises JLinkException – on error.

strace_code_fetch_event(*args, **kwargs)
Sets an event to trigger trace logic when an instruction is fetched.

Parameters
• self (JLink) – the JLink instance.
• operation (int) – one of the operations in JLinkStraceOperation.
• address (int) – the address of the instruction that is fetched.
• address_range (int) – optional range of address to trigger event on.

Returns An integer specifying the trace event handle. This handle should be retained in order to clear the event at a later time.

Raises JLinkException – on error.

strace_configure(*args, **kwargs)
Configures the trace port width for tracing.

Note that configuration cannot occur while STRACE is running.

Parameters
• **self** (*JLink*) – the JLink instance
• **port_width** (*int*) – the trace port width to use.

Returns None

Raises
• ValueError – if **port_width** is not 1, 2, or 4.
• JLinkException – on error.

**strace_data_access_event** (*args, **kwargs*)
Sets an event to trigger trace logic when data access is made.

Data access corresponds to either a read or write.

Parameters
• **self** (*JLink*) – the JLink instance.
• **operation** (*int*) – one of the operations in **JLinkStraceOperation**.
• **address** (*int*) – the address of the load/store data.
• **data** (*int*) – the data to be compared the event data to.
• **data_mask** (*int*) – optional bitmask specifying bits to ignore in comparison.
• **access_width** (*int*) – optional access width for the data.
• **address_range** (*int*) – optional range of address to trigger event on.

Returns An integer specifying the trace event handle. This handle should be retained in order to clear the event at a later time.

Raises JLinkException – on error.

**strace_data_load_event** (*args, **kwargs*)
Sets an event to trigger trace logic when data read access is made.

Parameters
• **self** (*JLink*) – the JLink instance.
• **operation** (*int*) – one of the operations in **JLinkStraceOperation**.
• **address** (*int*) – the address of the load data.
• **address_range** (*int*) – optional range of address to trigger event on.

Returns An integer specifying the trace event handle. This handle should be retained in order to clear the event at a later time.

Raises JLinkException – on error.

**strace_data_store_event** (*args, **kwargs*)
Sets an event to trigger trace logic when data write access is made.

Parameters
• **self** (*JLink*) – the JLink instance.
• **operation** (*int*) – one of the operations in **JLinkStraceOperation**.
• **address** (*int*) – the address of the store data.
• **address_range** (*int*) – optional range of address to trigger event on.
Returns An integer specifying the trace event handle. This handle should be retained in order to clear the event at a later time.

Raises JLinkException – on error.

strace_read (*args, **kwargs)
Reads and returns a number of instructions captured by STRACE.

The number of instructions must be a non-negative value of at most 0x10000 (65536).

Parameters
- self (JLink) – the JLink instance.
- num_instructions (int) – number of instructions to fetch.

Returns A list of instruction addresses in order from most recently executed to oldest executed instructions. Note that the number of instructions returned can be less than the number of instructions requested in the case that there are not num_instructions in the trace buffer.

Raises
- JLinkException – on error.
- ValueError – if num_instructions < 0 or num_instructions > 0x10000.

strace_set_buffer_size (*args, **kwargs)
Sets the STRACE buffer size.

Parameters
- self (JLink) – the JLink instance.

Returns None

Raises JLinkException – on error.

strace_start (*args, **kwargs)
Starts the capturing of STRACE data.

Parameters
- self (JLink) – the JLink instance.

Returns None

 Raises JLinkException – on error.

strace_stop (*args, **kwargs)
Stops the sampling of STRACE data.

Any capturing of STRACE data is automatically stopped when the CPU is halted.

Parameters
- self (JLink) – the JLink instance.

Returns None

Raises JLinkException – on error.

supported_device (index=0)
Gets the device at the given index.

Parameters
- self (JLink) – the JLink instance
- index (int) – the index of the device whose information to get

Returns A JLinkDeviceInfo describing the requested device.

Raises ValueError – if index is less than 0 or >= supported device count.
supported_tifs (*args, **kwargs)
    Returns a bitmask of the supported target interfaces.

    Parameters self (JLink) – the JLink instance
    Returns  Bitfield specifying which target interfaces are supported.

swd_read16 (*args, **kwargs)
    Gets a unit of 16 bits from the input buffer.

    Parameters
        • self (JLink) – the JLink instance
        • offset (int) – the offset (in bits) from which to start reading
    Returns  The integer read from the input buffer.

swd_read32 (*args, **kwargs)
    Gets a unit of 32 bits from the input buffer.

    Parameters
        • self (JLink) – the JLink instance
        • offset (int) – the offset (in bits) from which to start reading
    Returns  The integer read from the input buffer.

swd_read8 (*args, **kwargs)
    Gets a unit of 8 bits from the input buffer.

    Parameters
        • self (JLink) – the JLink instance
        • offset (int) – the offset (in bits) from which to start reading
    Returns  The integer read from the input buffer.

swd_sync (*args, **kwargs)
    Causes a flush to write all data remaining in output buffers to SWD device.

    Parameters
        • self (JLink) – the JLink instance
        • pad (bool) – True if should pad the data to full byte size
    Returns  None

swd_write (*args, **kwargs)
    Writes bytes over SWD (Serial Wire Debug).

    Parameters
        • self (JLink) – the JLink instance
        • output (int) – the output buffer offset to write to
        • value (int) – the value to write to the output buffer
        • nbits (int) – the number of bits needed to represent the output and value
    Returns  The bit position of the response in the input buffer.

swd_write16 (*args, **kwargs)
    Writes two bytes over SWD (Serial Wire Debug).
Parameters

- `self (JLink)` – the JLink instance
- `output (int)` – the output buffer offset to write to
- `value (int)` – the value to write to the output buffer

Returns The bit position of the response in the input buffer.

**swd_write32** (*args, **kwargs)
Writes four bytes over SWD (Serial Wire Debug).

Parameters

- `self (JLink)` – the JLink instance
- `output (int)` – the output buffer offset to write to
- `value (int)` – the value to write to the output buffer

Returns The bit position of the response in the input buffer.

**swd_write8** (*args, **kwargs)
Writes one byte over SWD (Serial Wire Debug).

Parameters

- `self (JLink)` – the JLink instance
- `output (int)` – the output buffer offset to write to
- `value (int)` – the value to write to the output buffer

Returns The bit position of the response in the input buffer.

**swo_disable** (*args, **kwargs)
Disables ITM & Stimulus ports.

Parameters

- `self (JLink)` – the JLink instance
- `port_mask (int)` – mask specifying which ports to disable

Returns None

Raises JLinkException – on error

**swo_enable** (*args, **kwargs)
Enables SWO output on the target device.

Configures the output protocol, the SWO output speed, and enables any ITM & stimulus ports.

This is equivalent to calling `.swo_start()`.

Note: If SWO is already enabled, it will first stop SWO before enabling it again.

Parameters

- `self (JLink)` – the JLink instance
- `cpu_speed (int)` – the target CPU frequency in Hz
- `swo_speed (int)` – the frequency in Hz used by the target to communicate
- `port_mask (int)` – port mask specifying which stimulus ports to enable
Returns None

Raises JLinkException – on error

swo_enabled()
Returns whether or not SWO is enabled.

Parameters self (JLink) – the JLink instance

Returns True if SWO is enabled, otherwise False.

swo_flush(*args, **kwargs)
Flushes data from the SWO buffer.

After this method is called, the flushed part of the SWO buffer is empty.

If num_bytes is not present, flushes all data currently in the SWO buffer.

Parameters

• self (JLink) – the JLink instance
• num_bytes (int) – the number of bytes to flush

Returns None

Raises JLinkException – on error

swo_num_bytes(*args, **kwargs)
Retrives the number of bytes in the SWO buffer.

Parameters self (JLink) – the JLink instance

Returns Number of bytes in the SWO buffer.

Raises JLinkException – on error

swo_read(*args, **kwargs)
Reads data from the SWO buffer.

The data read is not automatically removed from the SWO buffer after reading unless remove is True. Otherwise the callee must explicitly remove the data by calling .swo_flush().

Parameters

• self (JLink) – the JLink instance
• offset (int) – offset of first byte to be retrieved
• num_bytes (int) – number of bytes to read
• remove (bool) – if data should be removed from buffer after read

Returns A list of bytes read from the SWO buffer.

swo_read_stimulus(*args, **kwargs)
Reads the printable data via SWO.

This method reads SWO for one stimulus port, which is all printable data.

Note: Stimulus port 0 is used for printf debugging.

Parameters

• self (JLink) – the JLink instance
• **port** (*int*) – the stimulus port to read from, 0 - 31
  
• **num_bytes** (*int*) – number of bytes to read
  
Returns A list of bytes read via SWO.
  
Raises `ValueError` – if `port < 0` or `port > 31`

`swo_set_emu_buffer_size(*args, **kwargs)`
  
Sets the size of the buffer used by the J-Link to collect SWO data.
  
**Parameters**
  
• **self** (*JLink*) – the JLink instance
  
• **buf_size** (*int*) – the new size of the emulator buffer
  
**Returns** None
  
**Raises** `JLinkException` – on error

`swo_set_host_buffer_size(*args, **kwargs)`
  
Sets the size of the buffer used by the host to collect SWO data.
  
**Parameters**
  
• **self** (*JLink*) – the JLink instance
  
• **buf_size** (*int*) – the new size of the host buffer
  
**Returns** None
  
**Raises** `JLinkException` – on error

`swo_speed_info(*args, **kwargs)`
  
Retrieves information about the supported SWO speeds.
  
**Parameters** **self** (*JLink*) – the JLink instance
  
**Returns** A `JLinkSWOSpeedInfo` instance describing the target’s supported SWO speeds.
  
**Raises** `JLinkException` – on error

`swo_start(*args, **kwargs)`
  
Starts collecting SWO data.
  
**Parameters**
  
• **self** (*JLink*) – the JLink instance
  
• **swo_speed** (*int*) – the frequency in Hz used by the target to communicate
  
**Returns** None
  
**Raises** `JLinkException` – on error

`swo_stop(*args, **kwargs)`
  
Stops collecting SWO data.
  
**Parameters** **self** (*JLink*) – the JLink instance
  
**Returns** None

---

**Note:** If SWO is already enabled, it will first stop SWO before enabling it again.
PyLink Documentation, Release 0.11.1

Raises JLinkException – on error

swo_supported_speeds (*args, **kwargs)
Retrieves a list of SWO speeds supported by both the target and the connected J-Link.
The supported speeds are returned in order from highest to lowest.

Parameters
• self (JLink) – the JLink instance
• cpu_speed (int) – the target’s CPU speed in Hz
• num_speeds (int) – the number of compatible speeds to return

Returns A list of compatible SWO speeds in Hz in order from highest to lowest.

sync_firmware (*args, **kwargs)
Syncs the emulator’s firmware version and the DLL’s firmware.
This method is useful for ensuring that the firmware running on the J-Link matches the firmware supported
by the DLL.

Parameters self (JLink) – the JLink instance

Returns None

target_connected()
Returns whether a target is connected to the J-Link.

Parameters self (JLink) – the JLink instance

Returns True if a target is connected, otherwise False.

test()
Performs a self test.

Parameters self (JLink) – the JLink instance

Returns True if test passed, otherwise False.

tif
Returns the current target interface of the J-Link.

Parameters self (JLink) – the JLink instance

Returns Integer specifying the current target interface.

trace_buffer_capacity (*args, **kwargs)
Retrieves the trace buffer’s current capacity.

Parameters self (JLink) – the JLink instance

Returns The current capacity of the trace buffer. This is not necessarily the maximum possible
size the buffer could be configured with.

trace_flush (*args, **kwargs)
Flushes the trace buffer.
After this method is called, the trace buffer is empty. This method is best called when the device is reset.

Parameters self (JLink) – the JLink instance.

Returns None

trace_format (*args, **kwargs)
Retrieves the current format the trace buffer is using.
Parameters `self` (JLink) – the JLink instance.

Returns The current format the trace buffer is using. This is one of the attributes of JLinkTraceFormat.

`trace_max_buffer_capacity` (*args, **kwargs)
Retrieves the maximum size the trace buffer can be configured with.

Parameters `self` (JLink) – the JLink instance.

Returns The maximum configurable capacity for the trace buffer.

`trace_min_buffer_capacity` (*args, **kwargs)
Retrieves the minimum capacity the trace buffer can be configured with.

Parameters `self` (JLink) – the JLink instance.

Returns The minimum configurable capacity for the trace buffer.

`trace_read` (*args, **kwargs)
Reads data from the trace buffer and returns it.

Parameters
  • `self` (JLink) – the JLink instance.
  • `offset` (int) – the offset from which to start reading from the trace buffer.
  • `num_items` (int) – number of items to read from the trace buffer.

Returns A list of JLinkTraceData instances corresponding to the items read from the trace buffer. Note that this list may have size less than `num_items` in the event that there are not `num_items` items in the trace buffer.

Raises JLinkException – on error.

`trace_region` (*args, **kwargs)
Retrieves the properties of a trace region.

Parameters
  • `self` (JLink) – the JLink instance.
  • `region_index` (int) – the trace region index.

Returns An instance of JLinkTraceRegion describing the specified region.

`trace_region_count` (*args, **kwargs)
Retrieves a count of the number of available trace regions.

Parameters `self` (JLink) – the JLink instance.

Returns Count of the number of available trace regions.

`trace_sample_count` (*args, **kwargs)
Retrieves the number of samples in the trace buffer.

Parameters `self` (JLink) – the JLink instance.

Returns Number of samples in the trace buffer.

`trace_set_buffer_capacity` (*args, **kwargs)
Sets the capacity for the trace buffer.

Parameters
  • `self` (JLink) – the JLink instance.
• `size (int)` – the new capacity for the trace buffer.

Returns None

`trace_set_format (*args, **kwargs)`
Sets the format for the trace buffer to use.

Parameters
• `self (JLink)` – the JLink instance.
• `fmt (int)` – format for the trace buffer; this is one of the attributes of JLinkTraceFormat.

Returns None

`trace_start (*args, **kwargs)`
Starts collecting trace data.

Parameters `self (JLink)` – the JLink instance.

Returns None

`trace_stop (*args, **kwargs)`
Stops collecting trace data.

Parameters `self (JLink)` – the JLink instance.

Returns None

`unlock (*args, **kwargs)`
Unlocks the device connected to the J-Link.

Unlocking a device allows for access to read/writing memory, as well as flash programming.

**Note:** Unlock is not supported on all devices.

Supported Devices: Kinetis

Returns True.

Raises JLinkException – if the device fails to unlock.

`update_firmware (*args, **kwargs)`
Performs a firmware update.

If there is a newer version of firmware available for the J-Link device, then updates the firmware.

Parameters `self (JLink)` – the JLink instance

Returns Checksum of the new firmware on update, 0 if the firmware was not changed.

`version`
Returns the device’s version.

The device’s version is returned as a string of the format: M.mr where M is major number, m is minor number, and r is revision character.

Parameters `self (JLink)` – the JLink instance

Returns Device version string.

`warning_handler`
Returns the warning handler function.
Parameters `self` (JLink) – the JLink instance

Returns None if the warning handler was not set, otherwise `ctypes.CFUNCTYPE`.

`watchpoint_clear(*args, **kwargs)`

Cleans the watchpoint with the specified handle.

Parameters

- `self` (JLink) – the JLink instance
- `handle` (int) – the handle of the watchpoint

Returns True if watchpoint was removed, otherwise False.

`watchpoint_clear_all(*args, **kwargs)`

Removes all watchpoints that have been set.

Parameters `self` (JLink) – the JLink instance

Returns True if they were cleared, otherwise False.

`watchpoint_info(*args, **kwargs)`

Returns information about the specified watchpoint.

Note: Either handle or index can be specified. If the index is not provided, the handle must be set, and vice-versa. If both index and handle are provided, the index overrides the provided handle.

Parameters

- `self` (JLink) – the JLink instance
- `handle` (int) – optional handle of a valid watchpoint.
- `index` (int) – optional index of a watchpoint.

Returns An instance of JLinkWatchpointInfo specifying information about the watchpoint if the watchpoint was found, otherwise None.

Raises

- JLinkException – on error.
- ValueError – if both handle and index are invalid.

`watchpoint_set(*args, **kwargs)`

Sets a watchpoint at the given address.

This method allows for a watchpoint to be set on a given address or range of addresses. The watchpoint can then be triggered if the data at the given address matches the specified data or range of data as determined by data_mask, on specific access size events, reads, writes, or privileged accesses.

Both addr_mask and data_mask are used to specify ranges. Bits set to 1 are masked out and not taken into consideration when comparison against an address or data value. E.g., an addr_mask with a value of 0x1 and addr with value 0xdeadbeef means that the watchpoint will be set on addresses 0xdeadbeef and 0xdeadbeee. If the data was 0x11223340 and the given data_mask has a value of 0x0000000F, then the watchpoint would trigger for data matching 0x11223340 - 0x1122334F.
Note: If both read and write are specified, then the watchpoint will trigger on both read and write events to the given address.

Parameters
- **self** (*JLink*) – the JLink instance
- **addr_mask** (*int*) – optional mask to use for determining which address the watchpoint should be set on
- **data** (*int*) – optional data to set the watchpoint on in order to have the watchpoint triggered when the value at the specified address matches the given data
- **data_mask** (*int*) – optional mask to use for determining the range of data on which the watchpoint should be triggered
- **access_size** (*int*) – if specified, this must be one of {8, 16, 32} and determines the access size for which the watchpoint should trigger
- **read** (*bool*) – if True, triggers the watchpoint on read events
- **write** (*bool*) – if True, triggers the watchpoint on write events
- **privileged** (*bool*) – if True, triggers the watchpoint on privileged accesses

Returns  The handle of the created watchpoint.

Raises
- **ValueError** – if an invalid access size is given.
- **JLinkException** – if the watchpoint fails to be set.
CHAPTER 5

Protocols

The J-Link has multiple ways of communicating with a target: Serial Wire Debug (SWD), Serial Wire Output (SWO), Memory, Coresight, Registers, etc. For some of these communication methods, there is a specific protocol that defines how the communication takes place.

This module provides definitions to facilitate communicating over the different protocols. All the methods use a JLink instance, but take care of the housekeeping work involved with each protocol.

5.1 Serial Wire Debug (SWD)

This subsection defines the classes and methods needed to use the SWD protocol.

class pylink.protocols.swd.ReadRequest (address, ap)
    Bases: pylink.protocols.swd.Request
    Definition for a SWD (Serial Wire Debug) Read Request.

    send (jlink)
    Starts the SWD transaction.

    Steps for a Read Transaction:
    1. First phase in which the request is sent.
    2. Second phase in which an ACK is received. This phase consists of three bits. An OK response has the value 1.
    3. Once the ACK is received, the data phase can begin. Consists of 32 data bits followed by 1 parity bit calculated based on all 32 data bits.
    4. After the data phase, the interface must be clocked for at least eight cycles to clock the transaction through the SW-DP; this is done by reading an additional eight bits (eight clocks).

    Parameters
    • self (ReadRequest) – the ReadRequest instance
    • jlink (JLink) – the JLink instance to use for write/read

    Returns
    An Response instance.

class pylink.protocols.swd.Request (address, ap, data=None)
    Bases: _ctypes.Union
    Definition of a SWD (Serial Wire Debug) Request.
An SWD Request is composed of 8 bits.

**start**
the start bit is always one

**ap_dp**
indicates whether the transaction is DP (0) or AP (1).

**read_write**
indicates if the transaction is a read-access (1) or a write-access (0).

**address**

**parity**
the parity bit, the bit is used by the target to verify the integrity of the request. Should be 1 if bits 1–4 contain an odd number of 1’s, otherwise 0.

**stop**
the stop bit, should always be zero.

**park**
the park bit, should always be one.

**value**
the overall value of the request.

**addr2**
Structure/Union member

**addr3**
Structure/Union member

**ap_dp**
Structure/Union member

**bit**
Structure/Union member

**parity**
Structure/Union member

**park**
Structure/Union member

**read_write**
Structure/Union member

**send**(jlink)
Starts the SWD transaction.
Sends the request and receives an ACK for the request.

**Parameters**

- **self**(Request) – the Request instance
- **jlink**(JLink) – the JLink instance to use for write/read

**Returns**
The bit position of the ACK response.

**start**
Structure/Union member

**stop**
Structure/Union member
value
   Structure/Union member

class pylink.protocols.swd.RequestBits
   Bases: _ctypes.Structure
   SWD request bits.
   addr2
      Structure/Union member
   addr3
      Structure/Union member
   ap_dp
      Structure/Union member
   parity
      Structure/Union member
   park
      Structure/Union member
   read_write
      Structure/Union member
   start
      Structure/Union member
   stop
      Structure/Union member

class pylink.protocols.swd.Response
   (status, data=None)
   Bases: object
   Response class to hold the response from the send of a SWD request.
   STATUS_ACK = 1
   STATUS_FAULT = 4
   STATUS_INVALID = -1
   STATUS_WAIT = 2
   ack()
      Returns whether the response was ACK’d.
      Parameters self (Response) – the Response instance
      Returns True if response was ACK’d, otherwise False.
   fault()
      Returns whether the response exited with fault.
      Parameters self (Response) – the Response instance
      Returns True if response exited with a fault, otherwise False.
   invalid()
      Returns whether the response exited with a bad result.
      This occurs when the parity is invalid.
      Parameters self (Response) – the Response instance
      Returns True if the parity checked failed, otherwise False.
wait ()
Returns whether the response was a wait.

Parameters
- self (Response) – the Response instance

Returns True if response exited with wait, otherwise False.

class pylink.protocols.swd.WriteRequest (address, ap, data)
Bases: pylink.protocols.swd.Request
Definition for a SWD (Serial Wire Debug) Write Request.

send (jlink)
Starts the SWD transaction.

Steps for a Write Transaction:
1. First phase in which the request is sent.
2. Second phase in which an ACK is received. This phase consists of three bits. An OK response has the value 1.
3. Everytime the SWD IO may change directions, a turnaround phase is inserted. For reads, this happens after the data phase, while for writes this happens after between the acknowledge and data phase, so we have to do the turnaround before writing data. This phase consists of two bits.
4. Write the data and parity bits.

Parameters
- self (WriteRequest) – the WriteRequest instance
- jlink (JLink) – the JLink instance to use for write/read

Returns An Response instance.
Unlocking

Sometimes a user error may result in a device becoming locked. When a device is locked, it’s memory cannot be written to, nor can it’s memory be read from. This is a security feature in many MCUs.

This module provides functions for unlocking a locked device.

**Note:** Unlocking a device results in a mass-erase. Do not unlock a device if you do not want it be erased.

pylink.unlockers.unlock(*jlink, name*)

Unlocks a J-Link’s target device.

**Parameters**

- *jlink (JLink)* – the connected J-Link device
- *name (str)* – the MCU name (e.g. Kinetis)

**Returns** True if the device was unlocked, otherwise False.

**Raises** NotImplementedError – if no unlock method exists for the MCU.

pylink.unlockers.unlock_kinetis.unlock_kinetis(*args, **kwargs*)

Unlock for Freescale Kinetis K40 or K60 device.

**Parameters** *jlink (JLink)* – an instance of a J-Link that is connected to a target.

**Returns** True if the device was successfully unlocked, otherwise False.

**Raises** ValueError – if the J-Link is not connected to a target.
The native J-Link SDK is a C library. PyLink makes use of ctypes to interface with the library, and as such implements native Python structure bindings, and constants for values returned by the C SDK.

### 7.1 Structures

```python
class pylink.struts.JLinkBreakpointInfo
    Bases: _ctypes.Structure

    Class representing information about a breakpoint.

    SizeOfStruct
        the size of the structure (this should not be modified).

    Handle
        breakpoint handle.

    Addr
        address of where the breakpoint has been set.

    Type
        type flags which were specified when the breakpoint was created.

    ImpFlags
        describes the current state of the breakpoint.

    UseCnt
        describes how often the breakpoint is set at the same address.

    Addr
        Structure/Union member

    Handle
        Structure/Union member

    ImpFlags
        Structure/Union member

    SizeOfStruct
        Structure/Union member

    Type
        Structure/Union member
```
**UseCnt**  
Structure/Union member

**hardware_breakpoint()**  
Returns whether this is a hardware breakpoint.

Parameters `self` (`JLinkBreakpointInfo`) – the `JLinkBreakpointInfo` instance

Returns `True` if the breakpoint is a hardware breakpoint, otherwise `False`.

**pending()**  
Returns if this breakpoint is pending.

Parameters `self` (`JLinkBreakpointInfo`) – the `JLinkBreakpointInfo` instance

Returns `True` if the breakpoint is still pending, otherwise `False`.

**software_breakpoint()**  
Returns whether this is a software breakpoint.

Parameters `self` (`JLinkBreakpointInfo`) – the `JLinkBreakpointInfo` instance

Returns `True` if the breakpoint is a software breakpoint, otherwise `False`.

class `pylink.structs.JLinkConnectInfo`  
Bases: `_ctypes.Structure`  
J-Link connection info structure.

**SerialNumber**  
J-Link serial number.

**Connection**  
type of connection (e.g. `enums.JLinkHost.USB`)

**USBAddr**  
USB address if connected via USB.

**aIPAddr**  
IP address if connected via IP.

**Time**  
Time period (ms) after which UDP discover answer was received.

**Time_us**  
Time period (uS) after which UDP discover answer was received.

**HWVersion**  
Hardware version of J-Link, if connected via IP.

**abMACAddr**  
MAC Address, if connected via IP.

**acProduct**  
Product name, if connected via IP.

**acNickname**  
Nickname, if connected via IP.

**acFWString**  
Firmware string, if connected via IP.

**IsDHCPAssignedIP**  
Is IP address reception via DHCP.
IsDHCPAssignedIPIsValid
True if connected via IP.

NumIPConnections
Number of IP connections currently established.

NumIPConnectionsIsValid
True if connected via IP.

aPadding
Bytes reserved for future use.

Connection
Structure/Union member

HWVersion
Structure/Union member

IsDHCPAssignedIP
Structure/Union member

IsDHCPAssignedIPIsValid
Structure/Union member

NumIPConnections
Structure/Union member

NumIPConnectionsIsValid
Structure/Union member

SerialNumber
Structure/Union member

Time
Structure/Union member

Time_us
Structure/Union member

USBAddr
Structure/Union member

aIPAddr
Structure/Union member

aPadding
Structure/Union member

abMACAddr
Structure/Union member

acFWString
Structure/Union member

acNickname
Structure/Union member

acProduct
Structure/Union member

class pylink.structs.JLinkDataEvent
Bases: _ctypes.Structure
Class representing a data event.
A data may halt the CPU, trigger SWO output, or trigger trace output.

**SizeOfStruct**
the size of the structure (this should not be modified).

**Type**
the type of the data event (this should not be modified).

**Addr**
the address on which the watchpoint was set.

**AddrMask**
the address mask used for comparison.

**Data**
the data on which the watchpoint has been set.

**DataMask**
the data mask used for comparison.

**Access**
the control data on which the event has been set.

**AccessMask**
the control mask used for comparison.

```python
class pylink.structs.JLinkDeviceInfo(*args, **kwargs)
    Bases: _ctypes.Structure
    J-Link device information.
    This structure is used to represent a device that is supported by the J-Link.
    **SizeOfStruct**
    Size of the struct (DO NOT CHANGE).
    **sName**
    name of the device.
    **CoreId**
    core identifier of the device.
```
FlashAddr
base address of the internal flash of the device.

RAMAddr
base address of the internal RAM of the device.

EndianMode
the endian mode of the device (0 -> only little endian, 1 -> only big endian, 2 -> both).

FlashSize
total flash size in bytes.

RAMSize
total RAM size in bytes.

sManu
device manufacturer.

aFlashArea
a list of JLinkFlashArea instances.

aRamArea
a list of JLinkRAMArea instances.

Core
CPU core.

Core
Structure/Union member

CoreId
Structure/Union member

EndianMode
Structure/Union member

FlashAddr
Structure/Union member

FlashSize
Structure/Union member

RAMAddr
Structure/Union member

RAMSize
Structure/Union member

sizeofStruct
Structure/Union member

aFlashArea
Structure/Union member

aRamArea
Structure/Union member

manufacturer
Returns the name of the manufacturer of the device.

Parameters self (JLinkDeviceInfo) – the JLinkDeviceInfo instance

Returns Manufacturer name.
name
   Returns the name of the device.

   **Parameters**  
   `self` (*JLinkDeviceInfo*) – the `JLinkDeviceInfo` instance

   **Returns**  
   Device name.

`sManu`
   Structure/Union member

`sName`
   Structure/Union member

class pylink.structs.JLinkFlashArea
   Bases: `_ctypes.Structure`
   Definition for a region of Flash.

   **Addr**
   address where the flash area starts.

   **Size**
   size of the flash area.

class pylink.structs.JLinkGPIODescriptor
   Bases: `_ctypes.Structure`
   Definition for the structure that details the name and capabilities of a user-controllable GPIO.

   **acName**
   name of the GPIO.

   **Caps**
   bitfield of capabilities.

   **acName**
   Structure/Union member

   **Caps**
   Structure/Union member

class pylink.structs.JLinkHardwareStatus
   Bases: `_ctypes.Structure`
   Definition for the hardware status information for a J-Link.

   **VTarget**
   target supply voltage.

   **tck**
   measured state of TCK pin.

   **tdi**
   measured state of TDI pin.

   **tdo**
   measured state of TDO pin.
tms
measured state of TMS pin.

tres
measured state of TRES pin.

trst
measured state of TRST pin.

VTarget
Structure/Union member

tck
Structure/Union member
	di
Structure/Union member

tdi
Structure/Union member

tms
Structure/Union member

tres
Structure/Union member

trst
Structure/Union member

voltage
Returns the target supply voltage.
This is an alias for .VTarget.

Parameters
self (JLinkHardwareStatus) – the JLinkHardwareStatus instance

Returns
Target supply voltage as an integer.

class pylink.structs.JLinkMOEInfo
Bases: _ctypes.Structure
Structure representing the Method of Debug Entry (MOE).
The method of debug entry is a reason for which a CPU has stopped. At any given time, there may be multiple
methods of debug entry.

HaltReason
reason why the CPU stopped.

Index
if cause of CPU stop was a code/data breakpoint, this identifies the index of the code/data breakpoint unit
which causes the CPU to stop, otherwise it is -1.

HaltReason
Structure/Union member

Index
Structure/Union member

code_breakpoint ()
Returns whether this a code breakpoint.

Parameters
self (JLinkMOEInfo) – the JLinkMOEInfo instance
Returns True if this is a code breakpoint, otherwise False.

data_breakpoint()
Returns whether this a data breakpoint.

Parameters self (JLinkMOEInfo) – the JLinkMOEInfo instance

Returns True if this is a data breakpoint, otherwise False.

dbgrq()
Returns whether this a DBGRQ.

Parameters self (JLinkMOEInfo) – the JLinkMOEInfo instance

Returns True if this is a DBGRQ, otherwise False.

vectorCatch()
Returns whether this a vector catch.

Parameters self (JLinkMOEInfo) – the JLinkMOEInfo instance

Returns True if this is a vector catch, otherwise False.

class pylink.structs.JLinkMemoryZone
Bases: _ctypes.Structure

 Represents a CPU memory zone.

sName initials of the memory zone.

sDesc name of the memory zone.

VirtAddr start address of the virtual address space of the memory zone.

abDummy reserved for future use.

VirtAddr Structure/Union member

abDummy Structure/Union member

name Alias for the memory zone name.

Parameters self (JLinkMemoryZone) – the JLinkMemoryZone instance

Returns The memory zone name.

sDesc Structure/Union member

sName Structure/Union member

class pylink.structs.JLinkRAMArea
Bases: pylink.structs.JLinkFlashArea

Definition for a region of RAM.

Addr address where the flash area starts.
Size
size of the flash area.

```python
class pylink.structs.JLinkRTTerminalBufDesc
    Bases: _ctypes.Structure
    Structure describing a RTT buffer.

BufferIndex
    index of the buffer to request information about.

Direction
    direction of the upper (0 for up, 1 for Down).

acName
    Name of the buffer.

SizeOfBuffer
    size of the buffer in bytes.

Flags
    flags set on the buffer.

BufferIndex
    Structure/Union member

Direction
    Structure/Union member

Flags
    Structure/Union member

SizeOfBuffer
    Structure/Union member

acName
    Structure/Union member
```

down
Returns a boolean indicating if the buffer is an ‘DOWN’ buffer.

    Parameters self (JLinkRTTerminalBufDesc) – the terminal buffer descriptor.
    Returns True if the buffer is an ‘DOWN’ buffer, otherwise False.

name
Returns the name of the buffer.

    Parameters self (JLinkRTTerminalBufDesc) – the terminal buffer descriptor.
    Returns String name of the buffer.

up
Returns a boolean indicating if the buffer is an ‘UP’ buffer.

    Parameters self (JLinkRTTerminalBufDesc) – the terminal buffer descriptor.
    Returns True if the buffer is an ‘UP’ buffer, otherwise False.
```

```python
class pylink.structs.JLinkRTTerminalStart
    Bases: _ctypes.Structure
    Structure used to configure an RTT instance.

ConfigBlockAddress
    Address of the RTT block.
```

7.1. Structures
ConfigBlockAddress
  Structure/Union member

Reserved
  Structure/Union member

class pylink.structs.JLinkRTTerminalStatus
  Bases: _ctypes.Structure

Structure describing the status of the RTT terminal.

NumBytesTransferred
  number of bytes sent to the client application.

NumBytesRead
  number of bytes read from the target.

HostOverflowCount
  number of overflows on the host.

IsRunning
  if RTT is running.

NumUpBuffers
  number of ‘UP’ buffers.

NumDownBuffers
  number of ‘DOWN’ buffers.

HostOverflowCount
  Structure/Union member

IsRunning
  Structure/Union member

NumBytesRead
  Structure/Union member

NumBytesTransferred
  Structure/Union member

NumDownBuffers
  Structure/Union member

NumUpBuffers
  Structure/Union member

Reserved
  Structure/Union member

class pylink.structs.JLinkSWOSpeedInfo
  Bases: _ctypes.Structure

Structure representing information about target’s supported SWO speeds.

To calculate the supported SWO speeds, the base frequency is taken and divide by a number in the range of \([ MinDiv,MaxDiv ]\).

SizeofStruct
  size of the structure.

Interface
  interface type for the speed information.
**BaseFreq**
base frequency (Hz) used to calculate supported SWO speeds.

**MinDiv**
minimum divider allowed to divide the base frequency.

**MaxDiv**
maximum divider allowed to divide the base frequency.

**MinPrescale**
minimum prescaler allowed to adjust the base frequency.

**MaxPrescale**
maximum prescaler allowed to adjust the base frequency.

---

**Note:** You should *never* change `.SizeofStruct` or `.Interface`.

---

**BaseFreq**
Structure/Union member

**Interface**
Structure/Union member

**MaxDiv**
Structure/Union member

**MaxPrescale**
Structure/Union member

**MinDiv**
Structure/Union member

**MinPrescale**
Structure/Union member

**SizeofStruct**
Structure/Union member

---

**class** pylink.structs.JLinkSWOStartInfo

**Bases:** _ctypes.Structure

Represents configuration information for collecting Serial Wire Output (SWO) information.

**SizeofStruct**
size of the structure.

**Interface**
the interface type used for SWO.

**Speed**
the frequency used for SWO communication in Hz.

---

**Note:** You should *never* change `.SizeofStruct` or `.Interface`.

---

**Interface**
Structure/Union member

**SizeofStruct**
Structure/Union member
**Speed**
Structure/Union member

```python
class pylink.structs.JLinkSpeedInfo
    Bases: _ctypes.Structure

    Represents information about an emulator’s supported speeds.
    The emulator can support all target interface speeds calculated by dividing the base frequency by atleast
    MinDiv.

    **SizeOfStruct**
    the size of this structure.

    **BaseFreq**
    Base frequency (in HZ) used to calculate supported speeds.

    **MinDiv**
    minimum divider allowed to divide the base frequency.

    **SupportAdaptive**
    1 if emulator supports adaptive clocking, otherwise 0.
```

**BaseFreq**
Structure/Union member

**MinDiv**
Structure/Union member

**SizeOfStruct**
Structure/Union member

**SupportAdaptive**
Structure/Union member

```python
class pylink.structs.JLinkStraceEventInfo
    Bases: _ctypes.Structure

    Class representing the STRACE event information.

    **SizeOfStruct**
    size of the structure.

    **Type**
    type of event.

    **Op**
    the STRACE operation to perform.

    **AccessSize**
    access width for trace events.

    **Reserved0**
    reserved.

    **Addr**
    specifies the load/store address for data.

    **Data**
    the data to be compared for the operation for data access events.

    **DataMask**
    bitmask for bits of data to omit in comparision for data access events.
```
AddrRangeSize
    address range for range events.

AccessSize
    Structure/Union member

Addr
    Structure/Union member

AddrRangeSize
    Structure/Union member

Data
    Structure/Union member

DataMask
    Structure/Union member

Op
    Structure/Union member

Reserved0
    Structure/Union member

SizeOfStruct
    Structure/Union member

Type
    Structure/Union member

class pylink.structs.JLinkTraceData
    Bases: _ctypes.Structure

    Structure representing trace data returned by the trace buffer.

PipeStat
    type of trace data.

Sync
    sync point in buffer.

Packet
    trace data packet.

Packet
    Structure/Union member

PipeStat
    Structure/Union member

Sync
    Structure/Union member

branch()
    Returns whether the data corresponds to a branch execution.

    Parameters self (JLinkTraceData) – the JLinkTraceData instance.

    Returns True if this is trace data for a branch execution.

data_branch()
    Returns whether the data corresponds to a branch with data.

    Parameters self (JLinkTraceData) – the JLinkTraceData instance.
data_instruction()  
Returns whether the data corresponds to an data instruction.

Parameters self (JLinkTraceData) – the JLinkTraceData instance.

Returns True if this is trace data for a data instruction.

instruction()  
Returns whether the data corresponds to an executed instruction.

Parameters self (JLinkTraceData) – the JLinkTraceData instance.

Returns True if this is trace data for an executed instruction.

non_instruction()  
Returns whether the data corresponds to an un-executed instruction.

Parameters self (JLinkTraceData) – the JLinkTraceData instance.

Returns True if this is trace data for an un-executed instruction.

trace_disabled()  
Returns whether the data corresponds to trace being disabled.

Parameters self (JLinkTraceData) – the JLinkTraceData instance.

Returns True if this is trace data for the trace disabled event.

trigger()  
Returns whether the data corresponds to a trigger event.

Parameters self (JLinkTraceData) – the JLinkTraceData instance.

Returns True if this is trace data for a trigger event.

wait()  
Returns whether the data corresponds to a wait.

Parameters self (JLinkTraceData) – the JLinkTraceData instance.

Returns True if this is trace data for a wait.

class pylink.structs.JLinkTraceRegion  
Bases: _ctypes.Structure  
Structure describing a trace region.

SizeOfStruct  
size of the structure.

RegionIndex  
index of the region.

NumSamples  
number of samples in the region.

Off  
offset in the trace buffer.

RegionCnt  
number of trace regions.

Dummy  
unused.
**Timestamp**
- timestamp of last event written to buffer.

**Dummy**
- Structure/Union member

**NumSamples**
- Structure/Union member

**Off**
- Structure/Union member

**RegionCnt**
- Structure/Union member

**RegionIndex**
- Structure/Union member

**SizeOfStruct**
- Structure/Union member

**Timestamp**
- Structure/Union member

class pylink.structs.JLinkWatchpointInfo
    Bases: _ctypes.Structure
    Class representing information about a watchpoint.

**SizeOfStruct**
- the size of the structure (this should not be modified).

**Handle**
- the watchpoint handle.

**Addr**
- the address the watchpoint was set at.

**AddrMask**
- the address mask used for comparison.

**Data**
- the data on which the watchpoint was set.

**DataMask**
- the data mask used for comparison.

**Ctrl**
- the control data on which the breakpoint was set.

**CtrlMask**
- the control mask used for comparison.

**WPUnit**
- the index of the watchpoint unit.

**Addr**
- Structure/Union member

**AddrMask**
- Structure/Union member

**Ctrl**
- Structure/Union member
7.2 Enumerations

class pylink.enums.JLinkAccessFlags
    Bases: object
    J-Link access types for data events.
    These access types allow specifying the different types of access events that should be monitored.

    READ
        specifies to monitor read accesses.

    WRITE
        specifies to monitor write accesses.

    PRIVILEGED
        specifies to monitor privileged accesses.

    SIZE_8BIT
        specifies to monitor an 8-bit access width.

    SIZE_16BIT
        specifies to monitor an 16-bit access width.

    SIZE_32BIT
        specifies to monitor an 32-bit access width.

    PRIV = 16
    READ = 0
    SIZE_16BIT = 2
    SIZE_32BIT = 4
    SIZE_8BIT = 0
    WRITE = 1

class pylink.enums.JLinkAccessMaskFlags
    Bases: object
    J-Link access mask flags.
SIZE specifies to not care about the access size of the event.

DIR specifies to not care about the access direction of the event.

PRIV specifies to not care about the access privilege of the event.

DIR = 1
PRIV = 16
SIZE = 6

class pylink.enums.JLinkBreakpoint
   Bases: object

   J-Link breakpoint types.

   SW_RAM
   Software breakpoint located in RAM.

   SW_FLASH
   Software breakpoint located in flash.

   SW
   Software breakpoint located in RAM or flash.

   HW
   Hardware breakpoint.

   ANY
   Allows specifying any time of breakpoint.

   ARM
   Breakpoint in ARM mode (only available on ARM 7/9 cores).

   THUMB
   Breakpoint in THUMB mode (only available on ARM 7/9 cores).

   ANY = 4294967280
   ARM = 1
   HW = 4294967040
   SW = 240
   SW_FLASH = 32
   SW_RAM = 16
   THUMB = 2

class pylink.enums.JLinkBreakpointImplementation
   Bases: object

   J-Link breakpoint implementation types.

   HARD
   Hardware breakpoint using a breakpoint unit.

   SOFT
   Software breakpoint using a breakpoint instruction.
Breakpoint has not been set yet.

Breakpoint set in flash.

Breakpoint set in flash.

Flash = 16
Hard = 1
Pending = 4
Soft = 2

```python
class pylink.enums.JLinkCPUCapabilities
    Bases: object
    Target CPU Cabilities.
    DCC = 16384
    GO = 32
    HALT = 128
    HSS = 32768
    IS_HALTED = 256
    READ_MEMORY = 2
    READ_REGISTERS = 8
    RESET = 512
    RUN_STOP = 1024
    STEP = 64
    TERMINAL = 2048
    WRITE_MEMORY = 4
    WRITE_REGISTERS = 16
```

```python
class pylink.enums.JLinkCore
    Bases: object
    Enumeration for the different CPU core identifiers.
    These are the possible cores for targets the J-Link is connected to. Note that these are bitfields.
    ANY = 4294967295
    ARM11 = 201326591
    ARM1136 = 188153855
    ARM1136J = 188089087
    ARM1136JF = 188090111
    ARM1136JF_S = 188090367
    ARM1136J_S = 188089343
    ARM1156 = 190251007
    ARM1176 = 192348159
```
ARM1176J = 192283391
ARM1176JF = 192284415
ARM1176JF_S = 192284671
ARM1176J_S = 192283647
ARM7 = 134217727
ARM7TDMI = 117440767
ARM7TDMI_R3 = 117440575
ARM7TDMI_R4 = 117440591
ARM7TDMI_S = 117441023
ARM7TDMI_S_R3 = 117440831
ARM7TDMI_S_R4 = 117440847
ARM9 = 167772159
ARM920T = 153092351
ARM922T = 153223423
ARM926EJ_S = 153485823
ARM946E_S = 155582975
ARM966E_S = 157680127
ARM968E_S = 157811199
ARM9TDMI_S = 150995455
CIP51 = 302055423
COLDFIRE = 50331647
CORTEX_A12 = 134873343
CORTEX_A15 = 134938879
CORTEX_A17 = 135004415
CORTEX_A5 = 251658495
CORTEX_A7 = 134742271
CORTEX_A8 = 134217983
CORTEX_A9 = 134807807
CORTEX_M0 = 100663551
CORTEX_M1 = 16777471
CORTEX_M3 = 50331903
CORTEX_M3_R1P0 = 50331664
CORTEX_M3_R1P1 = 50331665
CORTEX_M3_R2P0 = 50331680
CORTEX_M4 = 234881279
CORTEX_M7 = 234946815

7.2. Enumerations
CORTEX_M_V8BASEL = 100729087
CORTEX_M_V8MAINL = 235012351
CORTEX_R4 = 201326847
CORTEX_R5 = 201392383
EFM8_UNSPEC = 318767103
MIPS = 301989887
MIPS_M4K = 285278207
MIPS_MICROAPTIV = 285343743
NONE = 0
POWER_PC = 285212671
POWER_PC_N1 = 285147391
POWER_PC_N2 = 285147647
RX = 234881023
RX110 = 220332031
RX111 = 220266495
RX113 = 220397567
RX210 = 219217919
RX21A = 219283455
RX220 = 219348991
RX230 = 219414527
RX231 = 219480063
RX23T = 219545599
RX610 = 218169343
RX621 = 218562559
RX62G = 218628095
RX62N = 218234879
RX62T = 218300415
RX630 = 218431487
RX631 = 218693631
RX63N = 218365951
RX63T = 218497023
RX64M = 221315071
RX71M = 221380607
SIM = 83886079
XSACLE = 100663295
class pylink.enums.JLinkDataErrors
    Bases: pylink.enums.JLinkGlobalErrors
    Enumeration for the error codes generated when setting a data event.

    ERROR_INVALID_ACCESS_MASK = 2147483776
    ERROR_INVALID_ADDR_MASK = 2147483680
    ERROR_INVALID_DATA_MASK = 2147483712
    ERROR_NO_MORE_ADDR_COMP = 2147483650
    ERROR_NO_MORE_DATA_COMP = 2147483652
    ERROR_NO_MORE_EVENTS = 2147483649
    ERROR_UNKNOWN = 2147483648

    @classmethod
to_string(error_code)
        Returns the string message for the given error code.

        Parameters
        • cls (JLinkDataErrors) – the JLinkDataErrors class
        • error_code (int) – error code to convert

        Returns An error string corresponding to the error code.
        Raises ValueError – if the error code is invalid.

class pylink.enums.JLinkDeviceFamily
    Bases: object
    Enumeration for the difference device families.

    These are the possible device families for targets that the J-Link is connected to.

    ANY = 255
    ARM10 = 10
    ARM11 = 11
    ARM7 = 7
    ARM9 = 9
    AUTO = 0
    COLDFIRE = 2
    CORTEX_A5 = 15
    CORTEX_A8 = 8
    CORTEX_A9 = 8
    CORTEX_M0 = 6
    CORTEX_M1 = 1
    CORTEX_M3 = 3
    CORTEX_M4 = 14
    CORTEX_R4 = 12
    EFM8 = 18
MIPS = 17
POWERPC = 16
RX = 13
SIMULATOR = 4
XSCALE = 5

class pylink.enums.JLinkEraseErrors
    Bases: pylink.enums.JLinkGlobalErrors
    Enumeration for the error codes generated during an erase operation.
    ILLEGAL_COMMAND = -5

classmethod to_string(error_code)
    Returns the string message for the given error_code.
    Parameters
    • cls (JLinkEraseErrors) – the JLinkEraseErrors class
    • error_code (int) – error code to convert
    Returns An error string corresponding to the error code.
    Raises ValueError – if the error code is invalid.

class pylink.enums.JLinkEventTypes
    Bases: object
    J-Link data event types.
    BREAKPOINT
        breakpoint data event.
    BREAKPOINT = 1

class pylink.enums.JLinkFlags
    Bases: object
    Enumeration for the different flags that are passed to the J-Link C SDK API methods.
    DLG_BUTTON_CANCEL = 8
    DLG_BUTTON_NO = 2
    DLG_BUTTON_OK = 4
    DLG_BUTTON_YES = 1
    GO_OVERSTEP_BP = 1
    HW_PIN_STATUS_HIGH = 1
    HW_PIN_STATUS_LOW = 0
    HW_PIN_STATUS_UNKNOWN = 255

class pylink.enums.JLinkFlashErrors
    Bases: pylink.enums.JLinkGlobalErrors
    Enumeration for the error codes generated during a flash operation.
    COMPARE_ERROR = -2
    PROGRAM_ERASE_ERROR = -3
VERIFICATION_ERROR = -4

classmethod to_string(error_code)

Returns the string message for the given error_code.

Parameters

- cls (JLinkFlashErrors) – the JLinkFlashErrors class
- error_code (int) – error code to convert

Returns An error string corresponding to the error code.

Raises ValueError – if the error code is invalid.

class pylink.enums.JLinkFunctions

Bases: object

Collection of function prototype and type builders for the J-Link SDK API calls.

FLASH_PROGRESS_PROTOTYPE
  alias of CFunctionType

LOG_PROTOTYPE
  alias of CFunctionType

UNSECURE_HOOK_PROTOTYPE
  alias of CFunctionType

class pylink.enums.JLinkGlobalErrors

Bases: object

Enumeration for the error codes which any J-Link SDK DLL API-function can have as a return value.

CPU_IN_LOW_POWER_MODE = -274
DEVICE_FEATURE_NOT_SUPPORTED = -271
DLL_NOT_OPEN = -258
EMU_COMM_ERROR = -257
EMU_FEATURE_UNSUPPORTED = -262
EMU_NO_CONNECTION = -256
EMU_NO_MEMORY = -263
FLASH_PROG_COMPARE_FAILED = -265
FLASH_PROG_PROGRAM_FAILED = -266
FLASH_PROG_VERIFY_FAILED = -267
INVALID_HANDLE = -260
NO_CPU_FOUND = -261
NO_TARGET_DEVICE_SELECTED = -273
OPEN_FILE_FAILED = -268
TIF_STATUS_ERROR = -264
UNKNOWN_FILE_FORMAT = -269
UNSPECIFIED_ERROR = -1
VCC_FAILURE = -259
WRITE_TARGET_MEMORY_FAILED = -270
WRONG_USER_CONFIG = -272

classmethod to_string(error_code)
      Returns the string message for the given error_code.

Parameters
      • cls (JlinkGlobalErrors) – the JLinkGlobalErrors class
      • error_code (int) – error code to convert

Returns
      An error string corresponding to the error code.

Raises
      ValueError – if the error code is invalid.

class pylink.enums.JLinkHaltReasons
      Bases: object
      Halt reasons for the CPU.

    DBGRQ
      CPU has been halted because DBGRQ signal asserted.

    CODE_BREAKPOINT
      CPU has been halted because of code breakpoint match.

    DATA_BREAKPOINT
      CPU has been halted because of data breakpoint match.

    VECTOR_CATCH
      CPU has been halted because of vector catch.

    CODE_BREAKPOINT = 1
    DATA_BREAKPOINT = 2
    DBGRQ = 0
    VECTOR_CATCH = 3

class pylink.enums.JLinkHost
      Bases: object
      Enumeration for the different JLink hosts: currently only IP and USB.

    IP = 2
    USB = 1
    USB_OR_IP = 3

class pylink.enums.JLinkInterfaces
      Bases: object
      Target interfaces for the J-Link.

    C2 = 6
    FINE = 3
    ICSP = 4
    JTAG = 0
    SPI = 5
    SWD = 1
class pylink.enums.JLinkROMTable
    Bases: object
    The J-Link ROM tables.
    AHBAP = 270
    APBAP = 269
    DBG = 268
    DWT = 261
    ETB = 267
    ETM = 257
    FPB = 262
    ITM = 260
    MTB = 258
    NONE = 256
    NVIC = 263
    PTM = 266
    SECURE = 271
    TF = 265
    TMC = 264
    TPIU = 259

class pylink.enums.JLinkRTTCommand
    Bases: object
    RTT commands.
    GETDESC = 2
    GETNUMBUF = 3
    GETSTAT = 4
    START = 0
    STOP = 1

class pylink.enums.JLinkRTTDirection
    Bases: object
    RTT Direction.
    DOWN = 1
    UP = 0

class pylink.enums.JLinkRTTErrors
    Bases: pylink.enums.JLinkGlobalErrors
    Enumeration for error codes from RTT.
    RTT_ERROR_CONTROL_BLOCK_NOT_FOUND = -2
    @classmethod
to_string(\n        error_code\n    )
    \n    Returns the string message for the given error code.
Parameters

- **cls (JLinkRTTErrors)** – the JLinkRTTErrors class
- **error_code (int)** – error code to convert

Returns  An error string corresponding to the error code.

Raises  ValueError – if the error code is invalid.

class pylink.enums.JLinkReadErrors
    Bases: pylink.enums.JLinkGlobalErrors

Enumeration for the error codes generated during a read.

**ZONE_NOT_FOUND_ERROR = -5**

classmethod to_string (error_code)

Returns the string message for the given error_code.

Parameters

- **cls (JLinkReadErrors)** – the JLinkReadErrors class
- **error_code (int)** – error code to convert

Returns  An error string corresponding to the error code.

Raises  ValueError – if the error code is invalid.

class pylink.enums.JLinkResetStrategyCortexM3
    Bases: object

Target reset strategies for the J-Link.

**NORMAL**

default reset strategy, does whatever is best to reset.

**CORE**

only the core is reset via the VECTRESET bit.

**RESETPIN**

pulls the reset pin low to reset the core and peripherals.

**CONNECT_UNDER_RESET**

J-Link connects to target while keeping reset active. This is recommended for STM32 devices.

**HALT_AFTER_BTL**

halt the core after the bootloader is executed.

**HALT_BEFORE_BTL**

halt the core before the bootloader is executed.

**KINETIS**

performs a normal reset, but also disables the watchdog.

**ADI_HALT_AFTER_KERNEL**

sets the SYSRESETREQ bit in the AIRCR in order to reset the device.

**CORE_AND_PERIPHERALS**

sets the SYSRESETREQ bit in the AIRCR, and the VC_CORERESET bit in the DEMCR to make sure that the CPU is halted immediately after reset.

**LPC1200**

reset for LPC1200 devices.
**S3FN60D**
reset for Samsung S3FN60D devices.

---

**Note:** Please see the J-Link SEGGER Documentation, UM8001, for full information about the different reset strategies.

---

```python
ADI_HALTED_AFTER_KERNEL = 7
CONNECT_UNDER_RESET = 3
CORE = 1
CORE_AND_PERIPHERALS = 8
HALT_AFTER_BTL = 4
HALT_BEFORE_BTL = 5
KINETIS = 6
LPC1200 = 9
NORMAL = 0
RESETPIN = 2
S3FN60D = 10

class pylink.enums.JLinkSWOCommands
    Bases: object
    Serial Wire Output (SWO) commands.
    FLUSH = 2
    GET_NUM_BYTES = 10
    GET_SPEED_INFO = 3
    SET_BUFFERSIZE_EMU = 21
    SET_BUFFERSIZE_HOST = 20
    START = 0
    STOP = 1

class pylink.enums.JLinkSWOInterfaces
    Bases: object
    Serial Wire Output (SWO) interfaces.
    MANCHESTER = 1
    UART = 0

class pylink.enums.JLinkStraceCommand
    Bases: object
    STRACE command.
    SET_BUFFER_SIZE = 3
    TRACE_EVENT_CLR = 1
    TRACE_EVENT_CLR_ALL = 2
```

---

7.2. Enumerations
class pylink.enums.JLinkStraceEvent
    Bases: object
    STRACE events.
    CODE_FETCH = 0
    DATA_ACCESS = 1
    DATA_LOAD = 2
    DATA_STORE = 3

class pylink.enums.JLinkStraceOperation
    Bases: object
    STRACE operation specifiers.
    TRACE_EXCLUDE_RANGE = 3
    TRACE_INCLUDE_RANGE = 2
    TRACE_START = 0
    TRACE_STOP = 1

class pylink.enums.JLinkTraceCommand
    Bases: object
    J-Link trace commands.
    FLUSH = 2
    GET_CONF_CAPACITY = 17
    GET_FORMAT = 33
    GET_MAX_CAPACITY = 20
    GET_MIN_CAPACITY = 19
    GET_NUM_REGIONS = 48
    GET_NUM_SAMPLES = 16
    GET_REGION_PROPS = 49
    GET_REGION_PROPS_EX = 50
    SET_CAPACITY = 18
    SET_FORMAT = 32
    START = 0
    STOP = 1

class pylink.enums.JLinkTraceFormat
    Bases: object
    J-Link trace formats.
    FORMAT_4BIT
      4-bit data.
    FORMAT_8BIT
      8-bit data.
FORMAT_16BIT
16-bit data.

FORMAT_MULTIPLEXED
multiplexing on ETM / buffer link.

FORMAT_DEMULTIPLEXED
de-multiplexing on ETM / buffer link.

FORMAT_DOUBLE_EDGE
clock data on both ETM / buffer link edges.

FORMAT_ETM7_9
ETM7/ETM9 protocol.

FORMAT_ETM10
ETM10 protocol.

FORMAT_1BIT
1-bit data.

FORMAT_2BIT
2-bit data.

FORMAT_16BIT = 4
FORMAT_1BIT = 256
FORMAT_2BIT = 512
FORMAT_4BIT = 1
FORMAT_8BIT = 2
FORMAT_DEMULTIPLEXED = 16
FORMAT_DOUBLE_EDGE = 32
FORMAT_ETM10 = 128
FORMAT_ETM7_9 = 64
FORMAT_MULTIPLEXED = 8

class pylink.enums.JLinkTraceSource
    Bases: object
    Sources for tracing.
    ETB = 0
    ETM = 1
    MTB = 2

class pylink.enums.JLinkVectorCatchCortexM3
    Bases: object
    Vector catch types for the ARM Cortex M3.
    CORE_RESET
        The CPU core reset.
    MEM_ERROR
        A memory management error occurred.
COPROCESSOR_ERROR
    Usage fault error accessing the Coprocessor.

CHECK_ERROR
    Usage fault error on enabled check.

STATE_ERROR
    Usage fault state error.

BUS_ERROR
    Normal bus error.

INT_ERROR
    Interrupt or exception service error.

HARD_ERROR
    Hard fault error.

BUS_ERROR = 256
CHECK_ERROR = 64
COPROCESSOR_ERROR = 32
CORE_RESET = 1
HARD_ERROR = 1024
INT_ERROR = 512
MEM_ERROR = 16
STATE_ERROR = 128

class pylink.enums.JLinkWriteErrors
    Bases: pylink.enums.JLinkGlobalErrors

Enumeration for the error codes generated during a write.

ZONE_NOT_FOUND_ERROR = -5

classmethod to_string(error_code)
    Returns the string message for the given error_code.

Parameters

    • cls(JLinkWriteErrors) – the JLinkWriteErrors class
    • error_code(int) – error code to convert

Returns An error string corresponding to the error code.

Raises ValueError – if the error code is invalid.
PyLink makes use of a number of different submodules as a part of its implementation. These submodules are *extras*, and the user should not need to use them explicitly.

### 8.1 Binpacker

This submodule provides functions for creating arrays of bytes from an integer.

```python
def pack(value, nbits=None):
    """Packs a given value into an array of 8-bit unsigned integers."
    """nbits""" is not present, calculates the minimal number of bits required to represent the given value. The result is little endian.

    Parameters
    • `value (int)` – the integer value to pack
    • `nbits (int)` – optional number of bits to use to represent the value

    Returns
    An array of `ctypes.c_uint8` representing the packed value.

    Raises
    • `ValueError` – if `value < 0 and nbits is None or nbits <= 0`.
    • `TypeError` – if `nbits or value` are not numbers.
```

```python
def pack_size(value):
    """Returns the number of bytes required to represent a given value."
    """
    Parameters
    • `value (int)` – the natural number whose size to get

    Returns
    The minimal number of bytes required to represent the given integer.

    Raises
    • `ValueError` – if `value < 0`.
    • `TypeError` – if `value is not a number`.
```

### 8.2 Decorators

This submodule provides different decorator functions.
pylink.decorators.async_decorator(func)
Asynchronous function decorator. Interprets the function as being asynchronous, so returns a function that will handle calling the Function asynchronously.

Parameters func(function) – function to be called asynchronously

Returns The wrapped function.

Raises AttributeError – if func is not callable

8.3 Registers

This submodule provides ctypes bindings for different registers.

class pylink.registers.AbortRegisterBits
Bases: _ctypes.Structure

This class holds the different bit mask for the Abort Register.

DAPABORT write 1 to trigger a DAP abort.

STKCMPCLR write 1 to clear the STICKYCMP sticky compare flag (only supported on SW-DP).

STKERRCLR write 1 to clear the STICKYERR sticky error flag (only supported on SW-DP).

WDERRCLR write 1 to clear the WDATAERR write data error flag (only supported on SW-DP).

ORUNERRCLR write 1 to clear the STICKYORUN overrun error flag (only supported on SW-DP).

DAPABORT Structure/Union member

ORUNERRCLR Structure/Union member

RESERVED Structure/Union member

STKCMPCLR Structure/Union member

STKERRCLR Structure/Union member

WDERRCLR Structure/Union member

class pylink.registers.AbortRegisterFlags
Bases: _ctypes.Union

Mask for the abort register bits.

value the value stored in the mask.

DAPABORT Structure/Union member
**class** pylink.registers.ControlStatusRegisterBits  
**Bases:** _ctypes.Structure

This class holds the different bit masks for the DP Control / Status Register bit assignments.

- **ORUNDETECT**  
  if set, enables overrun detection.

- **STICKYORUN**  
  if overrun is enabled, is set when overrun occurs.

- **TRNMODE**  
  transfer mode for access port operations.

- **STICKYCMP**  
  is set when a match occurs on a pushed compare or verify operation.

- **STICKYERR**  
  is set when an error is returned by an access port transaction.

- **READOK**  
  is set when the response to a previous access port or RDBUFF was OK.

- **WDATAERR**  
  set to 1 if a Write Data Error occurs.

- **MASKLANE**  
  bytes to be masked in pushed compare and verify operations.

- **TRNCNT**  
  transaction counter.

- **RESERVED**  
  reserved.

- **CDBGRSTREQ**  
  debug reset request.

- **CDBGRSTACK**  
  debug reset acknowledge.

- **CDBGPWRUPREQ**  
  debug power-up request.
CDBGPWRUPACK
debug power-up acknowledge.

CSYSPWRUPREQ
system power-up request

CSYSPWRUPACK
system power-up acknowledge.

See also:
See the ARM documentation on the significance of these masks here.

CDBGPWRUPACK
Structure/Union member

CDBGWUPREQ
Structure/Union member

CDBGXSTACK
Structure/Union member

CDBGXSTREQ
Structure/Union member

CSYSPWRUPACK
Structure/Union member

CSYSPWRUPREQ
Structure/Union member

MASKLANE
Structure/Union member

ORUNDETECT
Structure/Union member

READOK
Structure/Union member

RESERVED
Structure/Union member

STICKYCMP
Structure/Union member

STICKYERR
Structure/Union member

STICKYORUN
Structure/Union member

TRNCNT
Structure/Union member

TRNMODE
Structure/Union member

WDATAERR
Structure/Union member

class pylink.registers.ControlStatusRegisterFlags
Bases: _ctypes.Union
Mask for the control/status register bits.
value
   the value stored in the mask.

CDBGPWRUPACK
   Structure/Union member

CDBGPWRUPREQ
   Structure/Union member

CDBGRSSTACK
   Structure/Union member

CDBGRSSTREQ
   Structure/Union member

CSYSPWRUPACK
   Structure/Union member

CSYSPWRUPREQ
   Structure/Union member

MASKLANE
   Structure/Union member

ORUNDETECT
   Structure/Union member

READOK
   Structure/Union member

RESERVED
   Structure/Union member

STICKYCMP
   Structure/Union member

STICKYERR
   Structure/Union member

STICKYORUN
   Structure/Union member

TRNCNT
   Structure/Union member

TRNMODE
   Structure/Union member

WDATAERR
   Structure/Union member

bit
   Structure/Union member

value
   Structure/Union member

class pylink.registers.IDCodeRegisterBits
   Bases: _ctypes.Structure

   This class holds the different bit masks for the IDCode register.

   valid
       validity bit, should always be 0.
**manufacturer**
the JEDEC Manufacturer ID.

**part_no**
the part number defined by the manufacturer.

**version_code**
the version code.

**manufacturer**
Structure/Union member

**part_no**
Structure/Union member

**valid**
Structure/Union member

**version_code**
Structure/Union member

```python
class pylink.registers.IDCodeRegisterFlags
    Bases: _ctypes.Union

    Mask for the IDCode register bits.

    **value**
    the value stored in the mask.

    **bit**
    Structure/Union member

    **manufacturer**
    Structure/Union member

    **part_no**
    Structure/Union member

    **valid**
    Structure/Union member

    **value**
    Structure/Union member

    **version_code**
    Structure/Union member
```

```python
class pylink.registers.MDMAPControlRegisterBits
    Bases: _ctypes.Structure

    This class holds the different bit masks for the MDM-AP Control Register.

    **flash_mass_erase**
    set to cause a mass erase, this is cleared automatically when a mass erase finishes.

    **debug_disable**
    set to disable debug, clear to allow debug.

    **debug_request**
    set to force the core to halt.

    **sys_reset_request**
    set to force a system reset.
```
core_hold_reset
    set to suspend the core in reset at the end of reset sequencing.

VLLDBGREQ
    set to hold the system in reset after the next recovery from VLLSx (Very Low Leakage Stop).

VLLDBGACK
    set to release a system held in reset following a VLLSx (Very Low Leakage Stop) recovery.

VLLSTATACK
    set to acknowledge that the DAP LLS (Low Leakage Stop) and VLLS (Very Low Leakage Stop) status bits have read.

VLLDBGACK
    Structure/Union member

VLLDBGREQ
    Structure/Union member

VLLSTATACK
    Structure/Union member

core_hold_reset
    Structure/Union member

debug_disable
    Structure/Union member

debug_request
    Structure/Union member

flash_mass_erase
    Structure/Union member

sys_reset_request
    Structure/Union member

class pylink.registers.MDMAPControlRegisterFlags
    Bases: _ctypes.Union

    Mask for the MDM-AP control register bits.

    value
        the value stored in the mask.

VLLDBGACK
    Structure/Union member

VLLDBGREQ
    Structure/Union member

VLLSTATACK
    Structure/Union member

bit
    Structure/Union member

core_hold_reset
    Structure/Union member

debuge_disable
    Structure/Union member
debug_request
    Structure/Union member
flash_mass_erase
    Structure/Union member
sys_reset_request
    Structure/Union member
value
    Structure/Union member

class pylink.registers.MDMAPStatusRegisterBits
    Bases: _ctypes.Structure
Holds the bit masks for the MDM-AP Status Register.

flash_mass_erase_ack
    cleared after a system reset, indicates that a flash mass erase was acknowledged.

flash_ready
    indicates that flash has been initialized and can be configured.

system_security
    if set, system is secure and debugger cannot access the memory or system bus.

system_reset
    1 if system is in reset, otherwise 0.

mass_erase_enabled
    1 if MCU can be mass erased, otherwise 0.

low_power_enabled
    1 if low power stop mode is enabled, otherwise 0.

very_low_power_mode
    1 if device is in very low power mode.

LLSMODEEXIT
    indicates an exit from LLS mode has occurred.

VLLSxMODEEXIT
    indicates an exit from VLLSx mode has occurred.

core_halted; indicates core has entered debug halt mode.

core_deep_sleep
    indicates core has entered a low power mode.

core_sleeping
    indicates the core has entered a low power mode.

Note: if core_sleeping & !core_deep_sleep, then the core is in VLPW (very low power wait) mode, otherwise if core_sleeping & core_deep_sleep, then it is in VLPS (very low power stop) mode.

LLSMODEEXIT
    Structure/Union member

RESERVED_A
    Structure/Union member
RESERVED_B
Structure/Union member

RESERVED_C
Structure/Union member

VLLSxMODEEXIT
Structure/Union member

backdoor_access_enabled
Structure/Union member

core_deep_sleep
Structure/Union member

core_halted
Structure/Union member

core_sleeping
Structure/Union member

flash_mass_erase_ack
Structure/Union member

flash_ready
Structure/Union member

low_power_enabled
Structure/Union member

mass_erase_enabled
Structure/Union member

system_reset
Structure/Union member

system_security
Structure/Union member

very_low_power_mode
Structure/Union member

class pylink.registers.MDMAPStatusRegisterFlags
Bases: _ctypes.Union

Mask for the MDM-AP status register bits.

value
    the value stored in the mask.

LLSMODEEXIT
Structure/Union member

RESERVED_A
Structure/Union member

RESERVED_B
Structure/Union member

RESERVED_C
Structure/Union member

VLLSxMODEEXIT
Structure/Union member

8.3. Registers
backdoor_access_enabled
  Structure/Union member

bit
  Structure/Union member

core_deep_sleep
  Structure/Union member

core_halted
  Structure/Union member

core_sleeping
  Structure/Union member

flash_mass_erase_ack
  Structure/Union member

flash_ready
  Structure/Union member

low_power_enabled
  Structure/Union member

mass_erase_enabled
  Structure/Union member

system_reset
  Structure/Union member

system_security
  Structure/Union member

value
  Structure/Union member

very_low_power_mode
  Structure/Union member

class pylink.registers.SelectRegisterBits
  Bases: _ctypes.Structure

  This class holds the different bit masks for the AP Select Register.

CTRLSEL
  SW-DP debug port address bank select.

RESERVED_A
  reserved.

APBANKSEL
  selects the active four-word register window on the current access port.

RESERVED_B
  reserved.

APSEL
  selects the current access port.

APBANKSEL
  Structure/Union member

APSEL
  Structure/Union member
CTRLSEL
Structure/Union member
RESERVED_A
Structure/Union member
RESERVED_B
Structure/Union member

class pylink.registers.SelectRegisterFlags
    Bases: _ctypes.Union
    Mask for the select register bits.
    value
        the value stored in the mask.

APBANKSEL
Structure/Union member
APSEL
Structure/Union member
CTRLSEL
Structure/Union member
RESERVED_A
Structure/Union member
RESERVED_B
Structure/Union member
bit
    Structure/Union member
value
    Structure/Union member

8.4 Threads

This submodule provides custom threading.Thread types.

class pylink.threads.ThreadReturn (daemon=False, *args, **kwargs)
    Bases: threading.Thread
    Implementation of a thread with a return value.

See also:
    StackOverflow.

join (*args, **kwargs)
    Joins the thread.

    Parameters
    • self(ThreadReturn) – the ThreadReturn instance
    • args – optional list of arguments
    • kwargs – optional key-word arguments

    Returns  The return value of the exited thread.
run()

Runs the thread.

Parameters self (ThreadReturn) – the ThreadReturn instance

Returns None

### 8.5 Util

This submodule provides different utility functions.

**pylink.util.calculate_parity(n)**

Calculates and returns the parity of a number.

The parity of a number is 1 if the number has an odd number of ones in its binary representation, otherwise 0.

Parameters n (int) – the number whose parity to calculate

Returns 1 if the number has an odd number of ones, otherwise 0.

Raises ValueError – if n is less than 0.

**pylink.util.flash_progress_callback(action, progress_string, percentage)**

Callback that can be used with JLink.flash().

This callback generates a progress bar in the console to show the progress of each of the steps of the flash.

Parameters

- action (str) – the current action being invoked
- progress_string (str) – the current step in the progress
- percentage (int) – the percent to which the current step has been done

Returns None

Note: This function ignores the compare action.

**pylink.util.is_integer(val)**

Returns whether the given value is an integer.

Parameters val (object) – value to check

Returns True if the given value is an integer, otherwise False.

**pylink.util.is_natural(val)**

Returns whether the given value is a natural number.

Parameters val (object) – value to check

Returns True if the given value is a natural number, otherwise False.

**pylink.util.is_os_64bit()**

Returns whether the current running platform is 64bit.

Returns True if the platform is 64bit, otherwise False.

**pylink.util.noop(*args, **kwargs)**

No-op. Does nothing.
• **args** – list of arguments
• **kwargs** – keyword arguments dictionary

**Returns** None

`pylink.util.progress_bar(iteration, total, prefix=None, suffix=None, decs=1, length=100)`

Creates a console progress bar.

This should be called in a loop to create a progress bar.

See StackOverflow.

**Parameters**

• **iteration** (*int*) – current iteration
• **total** (*int*) – total iterations
• **prefix** (*str*) – prefix string
• **suffix** (*str*) – suffix string
• **decs** (*int*) – positive number of decimals in percent complete
• **length** (*int*) – character length of the bar

**Returns** None

**Note:** This function assumes that nothing else is printed to the console in the interim.

`pylink.util.unsecure_hook_dialog(title, msg, flags)`

No-op that ignores the dialog.

**Parameters**

• **title** (*str*) – title of the unsecure dialog
• **msg** (*str*) – text of the unsecure dialog
• **flags** (*int*) – flags specifying which values can be returned

**Returns** `enums.JLinkFlags.DLG_BUTTON_NO`
This page details common errors people run into while using PyLink. These errors do not mean the library is not working as intended, but rather a fault on the user end. If you cannot solve your issue by following any of the steps below, feel free to reach out.

9.1 Unspecified Error

If you ever see something similar to the following:

```
Traceback (most recent call last):
  File "pylink/decorators.py", line 38, in async_wrapper
    return func(*args, **kwargs)
  File "pylink/jlink.py", line 256, in open
    raise JLinkException(result)
__main__.JLinkException: Unspecified error.
```

Then congratulations, you’ve run into a catch-all error. This is a limitation imposed by native C SDK in which there is a catch-all error case. There are a couple possible solutions to this, and they are detailed below.

9.1.1 Unspecified Error during `open()`

If you see the unspecified error during `open()`, it means that one of the following is true:

- Your J-Link is not connected to your computer.
- Your J-Link is connected to your computer, but is currently held open by another application.

9.1.2 Unspecified Error during `connect()`

If you see the unspecified error during `connect()`, it means that any of the following is not true:

- The target device’s chip name you passed to `connect()` is not the chip name of the actual target.
- You’re trying to connect to the target over JTAG when it only supports SWD.
- You’re trying to connect to the target, but the target is not plugged in.
- You’re trying to connect to the target using a J-Link that does not have the target plugged in under its “Target” port.
- The connection speed is bad (try 'auto' instead).
9.1.3 Unspecified Error during `erase()`

If you see the unspecified error during `erase()`, it means that your device is not properly halted. If you’re using a Cortex-M device, try setting the reset strategy to `JLinkResetStrategyCortexM3.RESETPIN` to avoid your device’s application running when the system is booted; this is particularly useful if your application launches the watchdog or another service which would interpret the J-Link when erasing.

9.1.4 Unspecified Error during `flash()`

If you see the unspecified error during `flash()`, it means that either:

- Your device is not properly halt. While `flash()` attempts to halt the CPU, it cannot if the device is breakpoints or similar.
- The device is locked, in which case you have to unlock the device first.

9.1.5 Unspecified Error in Coresight

If you see an unspecified error while using a Coresight method, it means that you are trying to read from / write to an invalid register.
Serial Wire Output (SWO) alongside Serial Wire Debug (SWD) allows for the CPU to emit real-time trace data. In particular, when used with an Instrumentation Trace Macrocell (ITM), it can be used to form a Serial Wire Viewer (SWV). The ITM ports are provided by the ARM controller. The SWV typically implements a form of printf style debugging for embedded systems.

10.1 Getting Started

First, get your J-Link set up by instantiating an instance of a JLink and connecting to your target device. Once that is established, you want to call either `swo_start()`:

```python
speed = 9600
jlink.swo_start(swo_speed=speed)
```

or call `swo_enable()`:

```python
swo_speed = 9600
cpu_speed = 72000000 # 72 MHz
port_mask = 0x01
jlink.swo_enable(cpu_speed, swo_speed, port_mask)
```

Once enabled, you can begin reading data from the target.

10.2 Serial Wire Methods

```python
class pylink.jlink.JLink (lib=None, log=None, detailed_log=None, error=None, warn=None, unsecure_hook=None, serial_no=None, ip_addr=None, open_tunnel=False)

    Python interface for the SEGGER J-Link.

    This is a wrapper around the J-Link C SDK to provide a Python interface to it. The shared library is loaded and used to call the SDK methods.

    `swd_read16(*args, **kwargs)`

    Gets a unit of 16 bits from the input buffer.

    Parameters

    - **self (JLink)** – the JLink instance
    - **offset (int)** – the offset (in bits) from which to start reading
```
Returns The integer read from the input buffer.

```python
def swd_read32(*args, **kwargs)
    # Gets a unit of 32 bits from the input buffer.
```

Parameters

- `self (JLink)` – the JLink instance
- `offset (int)` – the offset (in bits) from which to start reading

Returns The integer read from the input buffer.

```python
def swd_read8(*args, **kwargs)
    # Gets a unit of 8 bits from the input buffer.
```

Parameters

- `self (JLink)` – the JLink instance
- `offset (int)` – the offset (in bits) from which to start reading

Returns The integer read from the input buffer.

```python
def swd_sync(*args, **kwargs)
    # Causes a flush to write all data remaining in output buffers to SWD device.
```

Parameters

- `self (JLink)` – the JLink instance
- `pad (bool)` – True if should pad the data to full byte size

Returns None

```python
def swd_write(*args, **kwargs)
    # Writes bytes over SWD (Serial Wire Debug).
```

Parameters

- `self (JLink)` – the JLink instance
- `output (int)` – the output buffer offset to write to
- `value (int)` – the value to write to the output buffer
- `nbits (int)` – the number of bits needed to represent the `output` and `value`

Returns The bit position of the response in the input buffer.

```python
def swd_write16(*args, **kwargs)
    # Writes two bytes over SWD (Serial Wire Debug).
```

Parameters

- `self (JLink)` – the JLink instance
- `output (int)` – the output buffer offset to write to
- `value (int)` – the value to write to the output buffer

Returns The bit position of the response in the input buffer.

```python
def swd_write32(*args, **kwargs)
    # Writes four bytes over SWD (Serial Wire Debug).
```

Parameters

- `self (JLink)` – the JLink instance
• **output** (*int*) – the output buffer offset to write to
• **value** (*int*) – the value to write to the output buffer

**Returns** The bit position of the response in the input buffer.

`swo_write8(*args, **kwargs)`
Writes one byte over SWD (Serial Wire Debug).

**Parameters**
• **self** (*JLink*) – the JLink instance
• **output** (*int*) – the output buffer offset to write to
• **value** (*int*) – the value to write to the output buffer

**Returns** The bit position of the response in the input buffer.

**Note:** If SWO is already enabled, it will first stop SWO before enabling it again.

**Parameters**
• **self** (*JLink*) – the JLink instance
• **cpu_speed** (*int*) – the target CPU frequency in Hz
• **swo_speed** (*int*) – the frequency in Hz used by the target to communicate
• **port_mask** (*int*) – port mask specifying which stimulus ports to enable

**Returns** None

**Raises** JLinkException – on error

`swo_flush(*args, **kwargs)`
Flushes data from the SWO buffer.

After this method is called, the flushed part of the SWO buffer is empty.

If **num_bytes** is not present, flushes all data currently in the SWO buffer.

**Parameters**
• **self** (*JLink*) – the JLink instance
• **num_bytes** (*int*) – the number of bytes to flush

**Returns** None

**Raises** JLinkException – on error

`swo_num_bytes(*args, **kwargs)`
Retrives the number of bytes in the SWO buffer.

**Parameters** **self** (*JLink*) – the JLink instance

**Returns** Number of bytes in the SWO buffer.

**Raises** JLinkException – on error
swo_read(*args, **kwargs)
Reads data from the SWO buffer.

The data read is not automatically removed from the SWO buffer after reading unless `remove` is True. Otherwise the callee must explicitly remove the data by calling `.swo_flush()`.

Parameters

• `self (JLink)` – the JLink instance
• `offset (int)` – offset of first byte to be retrieved
• `num_bytes (int)` – number of bytes to read
• `remove (bool)` – if data should be removed from buffer after read

Returns A list of bytes read from the SWO buffer.

swo_read_stimulus(*args, **kwargs)
Reads the printable data via SWO.

This method reads SWO for one stimulus port, which is all printable data.

Note: Stimulus port 0 is used for `printf` debugging.

Parameters

• `self (JLink)` – the JLink instance
• `port (int)` – the stimulus port to read from, 0 - 31
• `num_bytes (int)` – number of bytes to read

Returns A list of bytes read via SWO.

Raises ValueError – if `port < 0` or `port > 31`

swo_set_emu_buffer_size(*args, **kwargs)
Sets the size of the buffer used by the J-Link to collect SWO data.

Parameters

• `self (JLink)` – the JLink instance
• `buf_size (int)` – the new size of the emulator buffer

Returns None

Raises JLinkException – on error

swo_set_host_buffer_size(*args, **kwargs)
Sets the size of the buffer used by the host to collect SWO data.

Parameters

• `self (JLink)` – the JLink instance
• `buf_size (int)` – the new size of the host buffer

Returns None

Raises JLinkException – on error

swo_speed_info(*args, **kwargs)
Retrieves information about the supported SWO speeds.
Parameters `self`(JLink) – the JLink instance

Returns A `JLinkSWOSpeedInfo` instance describing the target’s supported SWO speeds.

Raises `JLinkException` – on error

`swo_start`(*args, **kwargs)
Starts collecting SWO data.

**Note:** If SWO is already enabled, it will first stop SWO before enabling it again.

Parameters

- `self`(JLink) – the JLink instance
- `swo_speed`(int) – the frequency in Hz used by the target to communicate

Returns None

Raises `JLinkException` – on error

`swo_stop`(*args, **kwargs)
Stops collecting SWO data.

Parameters `self`(JLink) – the JLink instance

Returns None

Raises `JLinkException` – on error

`swo_supported_speeds`(*args, **kwargs)
Retrives a list of SWO speeds supported by both the target and the connected J-Link.

The supported speeds are returned in order from highest to lowest.

Parameters

- `self`(JLink) – the JLink instance
- `cpu_speed`(int) – the target’s CPU speed in Hz
- `num_speeds`(int) – the number of compatible speeds to return

Returns A list of compatible SWO speeds in Hz in order from highest to lowest.

### 10.3 Examples

#### 10.3.1 Serial Wire Viewer

```python
# -*- coding: utf-8 -*-
# Copyright 2017 Square, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
```
# Example Serial Wire Viewer.

This module demonstrates implementing a Serial Wire Viewer using the PyLink library.

Usage: swv.py jlink_serial_number device

Author: Ford Peprah
Date: Friday, September 23rd, 2016
Copyright: 2016 Square, Inc.

```python
import pylink

try:
    import StringIO
except ImportError:
    import io as StringIO
import string
import sys
import time

# Serial Wire Viewer

def serial_wire_viewer(jlink_serial, device):
    # Implement a Serial Wire Viewer (SWV).

    # A Serial Wire Viewer (SWV) allows us implement real-time logging of output
    # from a connected device over Serial Wire Output (SWO).

    jlink_serial (str): the J-Link serial number
device (str): the target CPU

    Returns:
    Always returns `0``.

    Raises:
    JLinkException: on error

    # Use Serial Wire Debug as the target interface. Need this in order to use
    # Serial Wire Output.
    jlink = pylink.JLink(log=buf.write, detailed_log=buf.write)
    jlink.open(serial_no=jlink_serial)

    # Have to halt the CPU before getting its speed.
    jlink.reset()
jlink.halt()

cpu_speed = jlink.cpu_speed()```
```
```python
swo_speed = jlink.swo_supported_speeds(cpu_speed, 10)[0]

# Start logging serial wire output.
jlink.swo_start(swo_speed)
jlink.swo_flush()

# Output the information about the program.
sys.stdout.write('Serial Wire Viewer
')
sys.stdout.write('Press Ctrl-C to Exit
')
sys.stdout.write('Reading data from port 0:
')

# Reset the core without halting so that it runs.
jlink.reset(ms=10, halt=False)

# Use the `try` loop to catch a keyboard interrupt in order to stop logging
# serial wire output.
try:
    while True:
        # Check for any bytes in the stream.
        num_bytes = jlink.swo_num_bytes()

        if num_bytes == 0:
            # If no bytes exist, sleep for a bit before trying again.
            time.sleep(1)
            continue

        data = jlink.swo_read_stimulus(0, num_bytes)
        sys.stdout.write(''.join(map(chr, data)))
        sys.stdout.flush()
except KeyboardInterrupt:
    pass

sys.stdout.write('
')

# Stop logging serial wire output.
jlink.swo_stop()

return 0

if __name__ == '__main__':
    exit(serial_wire_viewer(sys.argv[1], sys.argv[2]))
```

10.3. Examples
PyLink is a Python library for interfacing with a J-Link. It leverages the J-Link C SDK. PyLink is in no way endorsed by or developed by SEGGER.

### 11.1 Goals

- Provide a Python interface for the J-Link C SDK.
- Provide a high-level API for flashing/running firmware via Python.
- Provide a high-level API for debugging devices.
- Provide a high-level API for unlocking locked devices.

### 11.2 License

PyLink is licensed under the Apache 2.0 Open-Source License.

### 11.3 Copyright

Copyright 2017 Square, Inc.

### 11.4 Sponsorship

This library was made possible by Square.
p

pylink.binpacker, 93
pylink.decorators, 93
pylink.enums, 78
pylink.errors, 11
pylink.jlink, 15
pylink.jlock, 14
pylink.library, 12
pylink.protocols.swd, 57
pylink.registers, 94
pylink.structs, 63
pylink.threads, 103
pylink.unlockers, 61
pylink.unlockers.unlock_kinetis, 61
pylink.util, 104
Symbols

_ standard_calls_ (pylink.library.Library attribute), 12

A

abDummy (pylink.structs.JLinkMemoryZone attribute), 70
abMACAddr (pylink.structs.JLinkConnectInfo attribute), 64, 65
AbortRegisterBits (class in pylink.registers), 94
AbortRegisterFlags (class in pylink.registers), 94
Access (pylink.structs.JLinkDataEvent attribute), 66
AccessMask (pylink.structs.JLinkDataEvent attribute), 66
AccessSize (pylink.structs.JLinkStraceEventInfo attribute), 74, 75
acFWString (pylink.structs.JLinkConnectInfo attribute), 64, 65
ack() (pylink.protocols.swd.Response method), 59
acName (pylink.structs.JLinkGPIODescriptor attribute), 68
acName (pylink.structs.JLinkRTTerminalBufDesc attribute), 71
acNickname (pylink.structs.JLinkConnectInfo attribute), 64, 65
acProduct (pylink.structs.JLinkConnectInfo attribute), 64, 65
acquire() (pylink.jlock.JLock method), 14
acquired (pylink.jlock.JLock attribute), 14
ADAPTIVE_JTAG_SPEED (pylink.jlink.JLink attribute), 15
add_license() (pylink.jlink.JLink method), 15
Addr (pylink.structs.JLinkBreakpointInfo attribute), 63
Addr (pylink.structs.JLinkDataEvent attribute), 66
Addr (pylink.structs.JLinkFlashArea attribute), 68
Addr (pylink.structs.JLinkRAMArea attribute), 70
Addr (pylink.structs.JLinkStraceEventInfo attribute), 74, 75
Addr (pylink.structs.JLinkWatchpointInfo attribute), 77
addr2 (pylink.protocols.swd.Request attribute), 58
addr2 (pylink.protocols.swd.RequestBits attribute), 59
address (pylink.protocols.swd.Request attribute), 58
AddrMask (pylink.structs.JLinkDataEvent attribute), 66
AddrMask (pylink.structs.JLinkWatchpointInfo attribute), 77
AddrRangeSize (pylink.structs.JLinkStraceEventInfo attribute), 74, 75
ADI_HALT_AFTER_KERNEL
(pylink.enums.JLinkResetStrategyCortexM3 attribute), 88, 89
aFlashArea (pylink.structs.JLinkDeviceInfo attribute), 67
AHBAP (pylink.enums.JLinkROMTable attribute), 87
aIPAddr (pylink.structs.JLinkConnectInfo attribute), 64, 65
ANY (pylink.enums.JLinkBreakpoint attribute), 79
ANY (pylink.enums.JLinkCore attribute), 80
ANY (pylink.enums.JLinkDeviceFamily attribute), 83
ap_dp (pylink.protocols.swd.Request attribute), 58
ap_dp (pylink.protocols.swd.RequestBits attribute), 59
aPadding (pylink.structs.JLinkConnectInfo attribute), 65
APBANKSEL (pylink.registers.SelectRegisterBits attribute), 102
APBANKSEL (pylink.registers.SelectRegisterFlags attribute), 103
APBAP (pylink.enums.JLinkROMTable attribute), 87
APSEL (pylink.registers.SelectRegisterBits attribute), 102
APSEL (pylink.registers.SelectRegisterFlags attribute), 103
aRAMArea (pylink.structs.JLinkDeviceInfo attribute), 67
aRamArea (pylink.structs.JLinkDeviceInfo attribute), 67
ARM (pylink.enums.JLinkBreakpoint attribute), 79
ARM10 (pylink.enums.JLinkDeviceFamily attribute), 83
ARM11 (pylink.enums.JLinkCore attribute), 80
ARM11 (pylink.enums.JLinkDeviceFamily attribute), 83
ARM1136 (pylink.enums.JLinkCore attribute), 80
ARM1136J (pylink.enums.JLinkCore attribute), 80
ARM1136J_S (pylink.enums.JLinkCore attribute), 80
ARM1136JF (pylink.enums.JLinkCore attribute), 80
ARM1136JF_S (pylink.enums.JLinkCore attribute), 80
ARM1156 (pylink.enums.JLinkCore attribute), 80
ARM1176 (pylink.enums.JLinkCore attribute), 80
ARM1176F (pylink.enums.JLinkCore attribute), 80
ARM1176F_S (pylink.enums.JLinkCore attribute), 81
ARM1176IF (pylink.enums.JLinkCore attribute), 81
ARM1176IF_S (pylink.enums.JLinkCore attribute), 81
ARM7 (pylink.enums.JLinkCore attribute), 81
ARM7 (pylink.enums.JLinkDeviceFamily attribute), 83
ARM7TDMI (pylink.enums.JLinkCore attribute), 81
ARM7TDMI_R3 (pylink.enums.JLinkCore attribute), 81
ARM7TDMI_R4 (pylink.enums.JLinkCore attribute), 81
ARM7TDMI_S (pylink.enums.JLinkCore attribute), 81
ARM7TDMI_S_R3 (pylink.enums.JLinkCore attribute), 81
ARM7TDMI_S_R4 (pylink.enums.JLinkCore attribute), 81
ARM9 (pylink.enums.JLinkCore attribute), 81
ARM9 (pylink.enums.JLinkDeviceFamily attribute), 83
ARM920T (pylink.enums.JLinkCore attribute), 81
ARM922T (pylink.enums.JLinkCore attribute), 81
ARM926EJ_S (pylink.enums.JLinkCore attribute), 81
ARM946E_S (pylink.enums.JLinkCore attribute), 81
ARM966E_S (pylink.enums.JLinkCore attribute), 81
ARM968E_S (pylink.enums.JLinkCore attribute), 81
ARM9TDMLI_S (pylink.enums.JLinkCore attribute), 81
async_decorator () (in module pylink.decorators), 93
AUTO (pylink.enums.JLinkDeviceFamily attribute), 83
AUTO_JTAG_SPEED (pylink.jlink.JLink attribute), 15
breakpoint_find () (pylink.jlink.JLink method), 16
breakpoint_info () (pylink.jlink.JLink method), 16
breakpoint_set () (pylink.jlink.JLink method), 16
BufferIndex (pylink.structs.JLinkRTTerminalBufDesc attribute), 71
BUS_ERROR (pylink.enums.JLinkVectorCatchCortexM3 attribute), 92
C
C2 (pylink.enums.JLinkInterfaces attribute), 86
calculate_parity () (in module pylink.util), 104
capabilities (pylink.jlink.JLink attribute), 16
Caps (pylink.structs.JLinkGPIODescriptor attribute), 68
CDBGPWRUPACK (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
CDBGPWRUPACK (pylink.registers.ControlStatusRegisterFlags attribute), 97
CDBGPWRUPREQ (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
CDBGPWRUPREQ (pylink.registers.ControlStatusRegisterFlags attribute), 97
CDBGSTACK (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
CDBGSTACK (pylink.registers.ControlStatusRegisterFlags attribute), 97
CDBGSTREQ (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
CDBGSTREQ (pylink.registers.ControlStatusRegisterFlags attribute), 97
CHECK_ERROR (pylink.enums.JLinkVectorCatchCortexM3 attribute), 92
CIP51 (pylink.enums.JLinkCore attribute), 81
clear_error () (pylink.jlink.JLink method), 17
close () (pylink.jlink.JLink method), 17
CODE_BREAKPOINT (pylink.enums.JLinkHaltReasons attribute), 86
code_breakpoint () (pylink.structs.JLinkMOEInfo method), 69
CODE_FETCH (pylink.enums.JLinkStraceEvent attribute), 90
code_memory_read () (pylink.jlink.JLink method), 17
COLDFIRE (pylink.enums.JLinkCore attribute), 81
COLDFIRE (pylink.enums.JLinkDeviceFamily attribute), 83
comm_supported () (pylink.jlink.JLink method), 17
COMPARE_ERROR (pylink.enums.JLinkFlashErrors attribute), 84
compatible_firmware_version (pylink.jlink.JLink attribute), 17
compile_date (pylink.jlink.JLink attribute), 17
ConfigBlockAddress (pylink.structs.JLinkRTTerminalStart attribute), 71
customize() (pylink.jlink.JLink method), 17
CONNECT_UNDER_RESET
   (pylink.enums.JLinkResetStrategyCortexM3 attribute), 88, 89
connected() (pylink.jlink.JLink method), 18
connected_emulators() (pylink.jlink.JLink method), 18
Connection (pylink.structs.JLinkConnectInfo attribute), 64, 65
connection_required() (pylink.jlink.JLink method), 18
ControlStatusRegisterBits (class in pylink.registers), 95
ControlStatusRegisterFlags (class in pylink.registers), 96
COPROCESSOR_ERROR
   (pylink.enums.JLinkVectorCatchCortexM3 attribute), 91, 92
CORE (pylink.enums.JLinkResetStrategyCortexM3 attribute), 88, 89
Core (pylink.structs.JLinkDeviceInfo attribute), 67
CORE_AND_PERIPHERALS
   (pylink.enums.JLinkResetStrategyCortexM3 attribute), 88, 89
core_cpu() (pylink.jlink.JLink method), 18
core_deep_sleep (pylink.registers.MDMAPStatusRegisterBits attribute), 100, 101
core_deep_sleep (pylink.registers.MDMAPStatusRegisterFlags attribute), 102
core_halted (pylink.registers.MDMAPStatusRegisterBits attribute), 101
core_halted (pylink.registers.MDMAPStatusRegisterFlags attribute), 102
core_hold_reset (pylink.registers.MDMAPControlRegisterBits attribute), 98, 99
core_hold_reset (pylink.registers.MDMAPControlRegisterFlags attribute), 99
core_id() (pylink.jlink.JLink method), 18
core_name() (pylink.jlink.JLink method), 18
CORE_RESET (pylink.enums.JLinkVectorCatchCortexM3 attribute), 91, 92
core_sleeping (pylink.registers.MDMAPStatusRegisterBits attribute), 100, 101
core_sleeping (pylink.registers.MDMAPStatusRegisterFlags attribute), 102
CoreId (pylink.structs.JLinkDeviceInfo attribute), 66, 67
coresight_configuration_required() (pylink.jlink.JLink method), 18
coresight_configure() (pylink.jlink.JLink method), 19
coresight_read() (pylink.jlink.JLink method), 19
coresight_write() (pylink.jlink.JLink method), 19
CORTEX_A12 (pylink.enums.JLinkCore attribute), 81
CORTEX_A15 (pylink.enums.JLinkCore attribute), 81
CORTEX_A17 (pylink.enums.JLinkCore attribute), 81
CORTEX_A5 (pylink.enums.JLinkCore attribute), 81
CORTEX_A5 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_A8 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_A9 (pylink.enums.JLinkCore attribute), 81
CORTEX_A9 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_M0 (pylink.enums.JLinkCore attribute), 81
CORTEX_M0 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_M1 (pylink.enums.JLinkCore attribute), 81
CORTEX_M1 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_M3 (pylink.enums.JLinkCore attribute), 81
CORTEX_M3 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_M3_R1P0 (pylink.enums.JLinkCore attribute), 81
CORTEX_M3_R1P1 (pylink.enums.JLinkCore attribute), 81
CORTEX_M3_R2P0 (pylink.enums.JLinkCore attribute), 81
CORTEX_M4 (pylink.enums.JLinkCore attribute), 81
CORTEX_M4 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_M5 (pylink.enums.JLinkCore attribute), 81
CORTEX_M7 (pylink.enums.JLinkCore attribute), 81
CORTEX_M_V8BASEL (pylink.enums.JLinkCore attribute), 81
CORTEX_M_V8MAINL (pylink.enums.JLinkCore attribute), 82
CORTEX_R4 (pylink.enums.JLinkCore attribute), 82
CORTEX_R4 (pylink.enums.JLinkDeviceFamily attribute), 83
CORTEX_R5 (pylink.enums.JLinkCore attribute), 82
cp15_present() (pylink.jlink.JLink method), 20
cp15_register_read() (pylink.jlink.JLink method), 20
cp15_register_write() (pylink.jlink.JLink method), 20
cpu_capability() (pylink.jlink.JLink method), 20
cpu_halt_reasons() (pylink.jlink.JLink method), 21
CPU_IN_LOW_POWER_MODE (pylink.enums.JLinkGlobalErrors attribute), 85
cpu_speed() (pylink.jlink.JLink method), 21
CSYSPWRUPACK (pylink.registers.ControlStatusRegisterBits attribute), 96
CSYSPWRUPACK (pylink.registers.ControlStatusRegisterFlags attribute), 97
CSYSPWRUPREQ (pylink.registers.ControlStatusRegisterBits attribute), 97
CSYSPWRUPREQ (pylink.registers.ControlStatusRegisterFlags attribute), 97
Crtl (pylink.structs.JLinkWatchpointInfo attribute), 77
CrtlMask (pylink.structs.JLinkWatchpointInfo attribute), 77
CTRLSEL (pylink.registers.SelectRegisterBits attribute), 102
CTRLSEL (pylink.registers.SelectRegisterFlags attribute), 102
D

DAPABORT (pylink.registers.AbortRegisterBits attribute), 94
DAPABORT (pylink.registers.AbortRegisterFlags attribute), 94
Data (pylink.structs.JLinkDataEvent attribute), 66
Data (pylink.structs.JLinkTraceData attribute), 75
Data (pylink.structs.JLinkWatchpointInfo attribute), 77, 78
DATA_ACCESS (pylink.enums.JLinkStraceEvent attribute), 90
data_branch() (pylink.structs.JLinkTraceData method), 75
DATA_BREAKPOINT (pylink.enums.JLinkHaltReasons attribute), 86
data_breakpoint() (pylink.structs.JLinkMOEInfo method), 70
data_instruction() (pylink.structs.JLinkTraceData method), 76
DATA_LOAD (pylink.enums.JLinkStraceEvent attribute), 90
DATA_STORE (pylink.enums.JLinkStraceEvent attribute), 90
DataMask (pylink.structs.JLinkDataEvent attribute), 66
DataMask (pylink.structs.JLinkTraceData attribute), 75
DataMask (pylink.structs.JLinkWatchpointInfo attribute), 77, 78
DBG (pylink.enums.JLinkROMTable attribute), 87
DBGRQ (pylink.enums.JLinkHaltReasons attribute), 86
dbgreq() (pylink.structs.JLinkMOEInfo method), 70
DCC (pylink.enums.JLinkCPUCapabilities attribute), 80
debug_disable (pylink.registers.MDMAPControlRegisterBits attribute), 98, 99
debug_disable (pylink.registers.MDMAPControlRegisterFlags attribute), 99
debug_request (pylink.registers.MDMAPControlRegisterBits attribute), 99
debug_request (pylink.registers.MDMAPControlRegisterFlags attribute), 99
detailed_log_handler (pylink.jlink.JLink method), 21
devic...
ERROR_INVALID_DATA_MASK (pylink.enums.JLinkDataErrors attribute), 83
ERROR_NO_MORE_ADDR_COMP (pylink.enums.JLinkDataErrors attribute), 83
ERROR_NO_MORE_DATA_COMP (pylink.enums.JLinkDataErrors attribute), 83
ERROR_NO_MORE_EVENTS (pylink.enums.JLinkDataErrors attribute), 83
ERROR_UNKNOWN (pylink.enums.JLinkDataErrors attribute), 83
ETB (pylink.enums.JLinkROMTable attribute), 87
ETB (pylink.enums.JLinkTraceSource attribute), 91
ETM (pylink.enums.JLinkROMTable attribute), 87
ETM (pylink.enums.JLinkTraceSource attribute), 91
etm_register_read() (pylink.jlink.JLink method), 24
etm_register_write() (pylink.jlink.JLink method), 24
eqm_supported() (pylink.jlink.JLink method), 24
extended_capabilities (pylink.jlink.JLink attribute), 24
exec_command() (pylink.jlink.JLink method), 24
FlashAddr (pylink.structs.JLinkDeviceInfo attribute), 66, 67
FlashSize (pylink.structs.JLinkDeviceInfo attribute), 67
FLUSH (pylink.enums.JLinkSWOCommands attribute), 89
FLUSH (pylink.enums.JLinkTraceCommand attribute), 90
FORMAT_16BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_1BIT (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_2BIT (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_4BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_8BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_DEMULTIPLEXED (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_DOUBLE_EDGE (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_4BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_8BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_DEMULTIPLEXED (pylink.enums.JLinkTraceFormat attribute), 91
INDEX 125
flashprogress_callback() (in module pylink.util), 104
flash_read (pylink.registers.MDMAPStatusRegisterBits attribute), 100, 101
flash_read (pylink.registers.MDMAPStatusRegisterFlags attribute), 102
flash_write() (pylink.jlink.JLink method), 26
flash_write16() (pylink.jlink.JLink method), 26
flash_write32() (pylink.jlink.JLink method), 27
flash_write8() (pylink.jlink.JLink method), 27
FlashAddr (pylink.structs.JLinkDeviceInfo attribute), 66, 67
FlashSize (pylink.structs.JLinkDeviceInfo attribute), 67
FLUSH (pylink.enums.JLinkSWOCommands attribute), 89
FLUSH (pylink.enums.JLinkTraceCommand attribute), 90
FORMAT_16BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_1BIT (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_2BIT (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_4BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_8BIT (pylink.enums.JLinkTraceFormat attribute), 90, 91
FORMAT_DEMULTIPLEXED (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_DOUBLE_EDGE (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_ETM10 (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_ETM7_9 (pylink.enums.JLinkTraceFormat attribute), 91
FORMAT_MULTIPLEXED (pylink.enums.JLinkTraceFormat attribute), 91
FPB (pylink.enums.JLinkROMTable attribute), 87
get_appropriate_windows_sdk_name() (pylink.library.Library class method), 13
get_conf_capacity (pylink.enums.JLinkTraceCommand attribute), 90
get_device_index() (pylink.jlink.JLink method), 27
PyLink Documentation, Release 0.11.1

GET_FORMAT (pylink.enums.JLinkTraceCommand attribute), 90
GET_MAX_CAPACITY (pylink.enums.JLinkTraceCommand attribute), 90
GET_MIN_CAPACITY (pylink.enums.JLinkTraceCommand attribute), 90
GET_NUM_BYTES (pylink.enums.JLinkSWOCommands attribute), 89
GET_NUM_REGIONS (pylink.enums.JLinkTraceCommand attribute), 90
GET_NUM_SAMPLES (pylink.enums.JLinkTraceCommand attribute), 90
GET_REGION_PROPS (pylink.enums.JLinkTraceCommand attribute), 90
GET_REGION_PROPS_EX (pylink.enums.JLinkTraceCommand attribute), 90
GET_SPEED_INFO (pylink.enums.JLinkSWOCommands attribute), 89
GETDESC (pylink.enums.JLinkRTTCommand attribute), 87
GETNUMBUF (pylink.enums.JLinkRTTCommand attribute), 87
GO (pylink.enums.JLinkCPUCapabilities attribute), 80
GO_OVERSTEP_BP (pylink.enums.JLinkFlags attribute), 84
gpio_get() (pylink.jlink.JLink method), 27
gpio_properties() (pylink.jlink.JLink method), 27
gpio_set() (pylink.jlink.JLink method), 28

H
HALT (pylink.enums.JLinkCPUCapabilities attribute), 80
halt() (pylink.jlink.JLink method), 28
HALT_AFTER_BTL (pylink.enums.JLinkResetStrategyCortexM3 attribute), 88, 89
HALT_BEFORE_BTL (pylink.enums.JLinkResetStrategyCortexM3 attribute), 88, 89
halted() (pylink.jlink.JLink method), 28
HaltReason (pylink.structs.JLinkMOEInfo attribute), 69
Handle (pylink.structs.JLinkBreakpointInfo attribute), 63
Handle (pylink.structs.JLinkWatchpointInfo attribute), 77, 78
HARD (pylink.enums.JLinkBreakpointImplementation attribute), 79, 80
HARD_ERROR (pylink.enums.JLinkVectorCatchCortexM3 attribute), 92
hardware_breakpoint() (pylink.structs.JLinkBreakpointInfo method), 64
hardware_breakpoint_set() (pylink.jlink.JLink method), 28
hardware_info (pylink.jlink.JLink attribute), 28
hardware_status (pylink.jlink.JLink attribute), 29
hardware_version (pylink.jlink.JLink attribute), 29
HostOverflowCount (pylink.structs.JLinkRTTerminalStatus attribute), 72
ice_register_read() (pylink.jlink.JLink method), 29
ice_register_write() (pylink.jlink.JLink method), 29
ICSP (pylink.enums.JLinkInterfaces attribute), 86
IDCodeRegisterBits (class in pylink.registers), 97
IDCodeRegisterFlags (class in pylink.registers), 98
ILLEGAL_COMMAND (pylink.enums.JLinkEraseErrors attribute), 84
ImpFlags (pylink.structs.JLinkBreakpointInfo attribute), 63
index (pylink.jlink.JLink attribute), 30
Index (pylink.structs.JLinkMOEInfo attribute), 69
instruction() (pylink.structs.JLinkTraceData method), 69
INT_ERROR (pylink.enums.JLinkVectorCatchCortexM3 attribute), 92
Interface (pylink.structs.JLinkSWOSpeedInfo attribute), 72, 73
Interface (pylink.structs.JLinkSWOStartInfo attribute), 73
interface_required() (pylink.jlink.JLink method), 30
is_valid() (pylink.protocols.swd.Response method), 59
INVALID_HANDLE (pylink.enums.JLinkGlobalErrors attribute), 85
INVALID_JTAG_SPEED (pylink.jlink.JLink attribute), 15
invalidate_firmware() (pylink.jlink.JLink method), 30
IP (pylink.enums.JLinkHost attribute), 86
IPADDR_NAME_FMT (pylink.jlock.JLock attribute), 14
ir_len() (pylink.jlink.JLink method), 30
IS_HALTED (pylink.enums.JLinkCPUCapabilities attribute), 80
is_integer() (in module pylink.util), 104
is_natural() (in module pylink.util), 104
is_os_64bit() (in module pylink.util), 104
IsDHCPAssignedIP (pylink.structs.JLinkConnectInfo attribute), 64, 65
manufacturer (pylink.registers.IDCodeRegisterBits attribute), 98
manufacturer (pylink.registers.IDCodeRegisterFlags attribute), 98
manufacturer (pylink.structs.JLinkDeviceInfo attribute), 67
MASKLANE (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
MASKLANE (pylink.registers.ControlStatusRegisterFlags attribute), 97
mass_erase_enabled (pylinkregisters.MDMAPStatusRegisterBits attribute), 100, 101
mass_erase_enabled (pylink.registers.MDMAPStatusRegisterFlags attribute), 102
MAX_BUF_SIZE (pylink.jlink.JLink attribute), 15
MAX_JTAG_SPEED (pylink.jlink.JLink attribute), 15
MAX_NUM_CPU_REGISTERS (pylink.jlink.JLink attribute), 15
MAX_NUM_MOES (pylink.jlink.JLink attribute), 15
MaxDiv (pylink.structs.JLinkSWOSpeedInfo attribute), 73
MaxPrescale (pylink.structs.JLinkSWOSpeedInfo attribute), 73
MDMAPControlRegisterBits (class in pylink.registers), 98
MDMAPControlRegisterFlags (class in pylink.registers), 99
MDMAPStatusRegisterBits (class in pylink.registers), 100
MDMAPStatusRegisterFlags (class in pylink.registers), 101
MEM_ERROR (pylink.enums.JLinkVectorCatchCortexM3 attribute), 91, 92
memory_read() (pylink.jlink.JLink method), 31
memory_read16() (pylink.jlink.JLink method), 32
memory_read32() (pylink.jlink.JLink method), 32
memory_read64() (pylink.jlink.JLink method), 32
memory_write() (pylink.jlink.JLink method), 33
memory_write16() (pylink.jlink.JLink method), 33
memory_write32() (pylink.jlink.JLink method), 33
memory_write64() (pylink.jlink.JLink method), 34
memory_zones() (pylink.jlink.JLink method), 34
MIN_JTAG_SPEED (pylink.jlink.JLink attribute), 15
MinDiv (pylink.structs.JLinkSpeedInfo attribute), 74
MinDiv (pylink.structs.JLinkSWOSpeedInfo attribute), 73
minimum_required() (pylink.jlink.JLink method), 34
MinPrescale (pylink.structs.JLinkSWOSpeedInfo attribute), 73
MIPS (pylink.enums.JLinkCore attribute), 82
MIPS (pylink.enums.JLinkDeviceFamily attribute), 83
MIPS_M4K (pylink.enums.JLinkCore attribute), 82
MIPS_MICROAPTIV (pylink.enums.JLinkCore attribute), 82
MTB (pylink.enums.JLinkROMTable attribute), 87
MTB (pylink.enums.JLinkTraceSource attribute), 91
name (pylink.jlock.JLock attribute), 14
name (pylink.structs.JLinkDeviceInfo attribute), 67
name (pylink.structs.JLinkMemoryZone attribute), 70
name (pylink.structs.JLinkRTTerminalBufDesc attribute), 71
NO_CPU_FOUND (pylink.enums.JLinkGlobalErrors attribute), 85
NO_TARGET_DEVICE_SELECTED (pylink.enums.JLinkGlobalErrors attribute), 85
non_instruction() (pylink.structs.JLinkTraceData method), 76
NONE (pylink.enums.JLinkCore attribute), 82
NONE (pylink.enums.JLinkROMTable attribute), 87
noop() (in module pylink.util), 104
NORMAL (pylink.enums.JLinkResetStrategyCortexM3 attribute), 88, 89
num_active_breakpoints() (pylink.jlink.JLink method), 34
num_active_watchpoints() (pylink.jlink.JLink method), 34
num_available_breakpoints() (pylink.jlink.JLink method), 35
num_available_watchpoints() (pylink.jlink.JLink method), 35
num_connected_emulators() (pylink.jlink.JLink method), 35
num_memory_zones() (pylink.jlink.JLink method), 35
num_supported_devices() (pylink.jlink.JLink method), 35
NumBytesRead (pylink.structs.JLinkRTTerminalStatus attribute), 72
NumBytesTransferred (pylink.structs.JLinkRTTerminalStatus attribute), 72
NumDownBuffers (pylink.structs.JLinkRTTerminalStatus attribute), 72
NumIPConnections (pylink.structs.JLinkConnectInfo attribute), 65
NumIPConnectionsIsValid (pylink.structs.JLinkConnectInfo attribute), 65
NumSamples (pylink.structs.JLinkTraceRegion attribute), 76, 77
NumUpBuffers (pylink.structs.JLinkRTTerminalStatus attribute), 72
NVIC (pylink.enums.JLinkROMTable attribute), 87
O
oem (pylink.jlink.JLink attribute), 35
Off (pylink.structs.JLinkTraceRegion attribute), 76, 77
set_tck_pin_low() (pylink.jlink.JLink method), 42
set_tdi_pin_high() (pylink.jlink.JLink method), 43
set_tdi_pin_low() (pylink.jlink.JLink method), 43
set_tif() (pylink.jlink.JLink method), 43
set_tms_pin_high() (pylink.jlink.JLink method), 43
set_tms_pin_low() (pylink.jlink.JLink method), 43
set_trace_source() (pylink.jlink.JLink method), 44
set_trst_pin_high() (pylink.jlink.JLink method), 43
set_trst_pin_low() (pylink.jlink.JLink method), 44
SIM (pylink.enums.JLinkCore attribute), 82
SIMULATOR (pylink.enums.JLinkDeviceFamily attribute), 84
SIZE (pylink.enums.JLinkAccessMaskFlags attribute), 78, 79
Size (pylink.structs.JLinkFlashArea attribute), 68
Size (pylink.structs.JLinkRAMArea attribute), 70
SIZE_16BIT (pylink.enums.JLinkAccessFlags attribute), 78
SIZE_32BIT (pylink.enums.JLinkAccessFlags attribute), 78
SIZE_8BIT (pylink.enums.JLinkAccessFlags attribute), 78
SizeOfBuffer (pylink.structs.JLinkRTTerminalBufDesc attribute), 71
SizeOfStruct (pylink.structs.JLinkBreakpointInfo attribute), 63
SizeOfStruct (pylink.structs.JLinkDataEvent attribute), 66
SizeOfStruct (pylink.structs.JLinkDeviceInfo attribute), 66
SizeOfStruct (pylink.structs.JLinkDeviceInfo attribute), 67
SizeOfStruct (pylink.structs.JLinkSpeedInfo attribute), 74
SizeOfStruct (pylink.structs.JLinkStraceEventInfo attribute), 74, 75
SizeOfStruct (pylink.structs.JLinkSWOSpeedInfo attribute), 72, 73
SizeOfStruct (pylink.structs.JLinkSWOSpeedInfo attribute), 73
SizeOfStruct (pylink.structs.JLinkTraceRegion attribute), 76, 77
SizeOfStruct (pylink.structs.JLinkWatchpointInfo attribute), 77, 78
sManu (pylink.structs.JLinkDeviceInfo attribute), 67, 68
sName (pylink.structs.JLinkDeviceInfo attribute), 66, 68
sName (pylink.structs.JLinkMemoryZone attribute), 70
SOFTWARE (pylink.enums.JLinkBreakpointImplementation attribute), 79, 80
software_breakpoint() (pylink.structs.JLinkBreakpointInfo method), 64
software_breakpoint_set() (pylink.jlink.JLink method), 44
speed (pylink.jlink.JLink attribute), 44
Speed (pylink.structs.JLinkSWOStartInfo attribute), 73
speed_info (pylink.jlink.JLink attribute), 44
SPI (pylink.enums.JLinkInterfaces attribute), 86
START (pylink.enums.JLinkRTTCommand attribute), 87
START (pylink.enums.JLinkSWOCommands attribute), 89
START (pylink.enums.JLinkTraceCommand attribute), 90
start (pylink.protocols.swd.Request attribute), 58
start (pylink.protocols.swd.RequestBits attribute), 59
strace_clear() (pylink.jlink.JLink method), 45
strace_clear_all() (pylink.jlink.JLink method), 45
strace_code_fetch_event() (pylink.jlink.JLink method), 45
strace_configure() (pylink.jlink.JLink method), 45
strace_data_access_event() (pylink.jlink.JLink method),
strace_data_load_event() (pylink.jlink.JLink method), 46
strace_data_store_event() (pylink.jlink.JLink method), 46
strace_read() (pylink.jlink.JLink method), 47
strace_set_buffer_size() (pylink.jlink.JLink method), 47
strace_start() (pylink.jlink.JLink method), 47
strace_stop() (pylink.jlink.JLink method), 47
SupportAdaptive (pylink.structs.JLinkSpeedInfo attribute), 74
supported_device() (pylink.jlink.JLink method), 47
supported_tifs() (pylink.jlink.JLink method), 47
SW (pylink.enums.JLinkBreakpoint attribute), 79
SW_FLASH (pylink.enums.JLinkBreakpoint attribute), 79
SW_RAM (pylink.enums.JLinkBreakpoint attribute), 79
SWD (pylink.enums.JLinkInterfaces attribute), 86
swd_read16() (pylink.jlink.JLink method), 48
swd_read32() (pylink.jlink.JLink method), 48
swd_read8() (pylink.jlink.JLink method), 48
swd_sync() (pylink.jlink.JLink method), 48
swd_write() (pylink.jlink.JLink method), 48
swd_write16() (pylink.jlink.JLink method), 48
swd_write32() (pylink.jlink.JLink method), 49
swd_write8() (pylink.jlink.JLink method), 49
swo_disable() (pylink.jlink.JLink method), 49
swo_enable() (pylink.jlink.JLink method), 49
swo_flush() (pylink.jlink.JLink method), 50
swo_num_bytes() (pylink.jlink.JLink method), 50
swo_read() (pylink.jlink.JLink method), 50
swo_read_stimulus() (pylink.jlink.JLink method), 50
swo_set_emu_buffer_size() (pylink.jlink.JLink method), 51
swo_set_host_buffer_size() (pylink.jlink.JLink method), 51
swo_speed_info() (pylink.jlink.JLink method), 51
swo_start() (pylink.jlink.JLink method), 51
swo_stop() (pylink.jlink.JLink method), 51
swo_supported_speeds() (pylink.jlink.JLink method), 52
Sync (pylink.structs.JLinkTraceData attribute), 75
sync_firmware() (pylink.registers.MDMAPControlRegisterBits attribute), 98
sys_reset_request (pylink.registers.MDMAPControlRegisterFlags attribute), 100
system_reset (pylink.registers.MDMAPStatusRegisterBits attribute), 100
system_reset (pylink.registers.MDMAPStatusRegisterFlags attribute), 102
system_security (pylink.registers.MDMAPStatusRegisterBits attribute), 100
system_security (pylink.registers.MDMAPStatusRegisterFlags attribute), 102
T
target_connected() (pylink.jlink.JLink method), 52
tck (pylink.struts.JLinkHardwareStatus attribute), 68, 69
tdi (pylink.struts.JLinkHardwareStatus attribute), 68, 69
tdo (pylink.struts.JLinkHardwareStatus attribute), 68, 69
TERMINAL (pylink.enums.JLinkCPUCapabilities attribute), 80
test() (pylink.jlink.JLink method), 52
TF (pylink.enums.JLinkROMTable attribute), 87
ThreadReturn (class in pylink.threads), 103
THUMB (pylink.enums.JLinkBreakpoint attribute), 79
tif (pylink.jlink.JLink attribute), 52
TIF_STATUS_ERROR (pylink.enums.JLinkGlobalErrors attribute), 85
Time (pylink.struts.JLinkConnectInfo attribute), 64, 65
time_us (pylink.struts.JLinkConnectInfo attribute), 64, 65
Timestamp (pylink.struts.JLinkTraceRegion attribute), 76, 77
TMC (pylink.enums.JLinkROMTable attribute), 87
tms (pylink.struts.JLinkHardwareStatus attribute), 68, 69
to_string() (pylink.enums.JLinkDataErrors class method), 83
to_string() (pylink.enums.JLinkEraseErrors class method), 84
to_string() (pylink.enums.JLinkFlashErrors class method), 85
to_string() (pylink.enums.JLinkGlobalErrors class method), 86
to_string() (pylink.enums.JLinkReadErrors class method), 87
to_string() (pylink.enums.JLinkRTTErrors class method), 87
TPIU (pylink.enums.JLinkROMTable attribute), 87
trace_buffer_capacity() (pylink.jlink.JLink method), 52
trace_disabled() (pylink.struts.JLinkTraceData method), 76
TRACE_EVENT_CLR (pylink.enums.JLinkStraceCommand attribute), 89
TRACE_EVENT_CLR_ALL (pylink.enums.JLinkStraceCommand attribute), 89
TRACE_EVENT_SET (pylink.enums.JLinkStraceCommand attribute), 89
TRACE_EXCLUDE_RANGE (pylink.enums.JLinkStraceOperation attribute), 90
trace_flush() (pylink.jlink.JLink method), 52
trace_format() (pylink.jlink.JLink method), 52
TRACE_INCLUDE_RANGE (pylink.enums.JLinkStraceOperation attribute), 92
trace_max_buffer_capacity() (pylink.jlink.JLink method), 53
trace_min_buffer_capacity() (pylink.jlink.JLink method), 53
trace_read() (pylink.jlink.JLink method), 53
trace_region() (pylink.jlink.JLink method), 53
trace_region_count() (pylink.jlink.JLink method), 53
trace_sample_count() (pylink.jlink.JLink method), 53
trace_set_buffer_capacity() (pylink.jlink.JLink method), 53
trace_set_format() (pylink.jlink.JLink method), 54
TRACE_START (pylink.enums.JLinkStraceOperation attribute), 90
trace_start() (pylink.jlink.JLink method), 54
trace_stop() (pylink.jlink.JLink method), 54
tres (pylink.structs.JLinkHardwareStatus attribute), 69
trigger() (pylink.structs.JLinkTraceData method), 76
TRNCNT (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
TRNCNT (pylink.registers.ControlStatusRegisterFlags attribute), 97
TRNMODE (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
TRNMODE (pylink.registers.ControlStatusRegisterFlags attribute), 97
trst (pylink.structs.JLinkHardwareStatus attribute), 69
Type (pylink.structs.JLinkBreakpointInfo attribute), 63
Type (pylink.structs.JLinkDataEvent attribute), 66
Type (pylink.structs.JLinkStraceEventInfo attribute), 74, 75

UART (pylink.enums.JLinkSWOInterfaces attribute), 89
UNKNOWN_FILE_FORMAT (pylink.enums.JLinkGlobalErrors attribute), 85
unload() (pylink.library.Library method), 13
unlock() (in module pylink.unlockers), 61
unlock() (pylink.jlink.JLink method), 54
unlock_kinetis() (in module pylink.jlink), 54
unsecure_hook_dialog() (in module pylink.util), 105
UNSECURE_HOOK_PROTOTYPE (pylink.enums.JLinkFunctions attribute), 85
UNSPECIFIED_ERROR (pylink.enums.JLinkGlobalErrors attribute), 85
UP (pylink.enums.JLinkRTTDirection attribute), 87
up (pylink.structs.JLinkRTTerminalBufDesc attribute), 71
update_firmware() (pylink.jlink.JLink method), 54

USB (pylink.enums.JLinkHost attribute), 86
USBAddr (pylink.structs.JLinkConnectInfo attribute), 64, 65
UseCnt (pylink.structs.JLinkBreakpointInfo attribute), 63

V
valid (pylink.registers.IDCodeRegisterBits attribute), 97, 98
value (pylink.registers.IDCodeRegisterFlags attribute), 98
value (pylink.protocols.swd.Request attribute), 58
value (pylink.registers.AbortRegisterFlags attribute), 94, 95
value (pylink.registers.ControlStatusRegisterFlags attribute), 96, 97
value (pylink.registers.IDCodeRegisterFlags attribute), 98
value (pylink.registers.MDMAPControlRegisterFlags attribute), 99, 100
value (pylink.registers.MDMAPStatusRegisterFlags attribute), 101, 102
value (pylink.registers.SelectRegisterFlags attribute), 103
VCC_FAILURE (pylink.enums.JLinkGlobalErrors attribute), 85
VECTOR_CATCH (pylink.enums.JLinkHaltReasons attribute), 86
vector_catch() (pylink.structs.JLinkMOEInfo method), 70
VERIFICATION_ERROR (pylink.enums.JLinkFlashErrors attribute), 84

version (pylink.jlink.JLink attribute), 54
version_code (pylink.registers.IDCodeRegisterBits attribute), 98
version_code (pylink.registers.IDCodeRegisterFlags attribute), 98
very_low_power_mode (pylink.registers.MDMAPStatusRegisterBits attribute), 100, 101
very_low_power_mode (pylink.registers.MDMAPStatusRegisterFlags attribute), 100, 101
VirtAddr (pylink.structs.JLinkMemoryZone attribute), 70
VLLDBGACK (pylink.registers.MDMAPControlRegisterBits attribute), 99
VLLDBGACK (pylink.registers.MDMAPControlRegisterFlags attribute), 99
VLLDBGREQ (pylink.registers.MDMAPControlRegisterBits attribute), 99
VLLDBGREQ (pylink.registers.MDMAPControlRegisterFlags attribute), 99
VLLSxMODEEXIT (pylink.registers.MDMAPStatusRegisterBits attribute), 100, 101
VLLSxMODEEXIT (pylinkregisters.MDMAPStatusRegisterFlags attribute), 101
voltage (pylinkstructs.JLinkHardwareStatus attribute), 69
VTarget (pylinkstructs.JLinkHardwareStatus attribute), 68, 69

W
wait() (pylinkprotocols.swd.Response method), 59
wait() (pylinkstructs.JLinkTraceData method), 76
warning_handler (pylink.jlink.JLink attribute), 54
watchpoint_clear() (pylink.jlink.JLink method), 55
watchpoint_clear_all() (pylink.jlink.JLink method), 55
watchpoint_info() (pylink.jlink.JLink method), 55
watchpoint_set() (pylink.jlink.JLink method), 55
WDATAERR (pylink.registers.ControlStatusRegisterBits attribute), 95, 96
WDATAERR (pylink.registers.ControlStatusRegisterFlags attribute), 97
WDERRCLR (pylink.registers.AbortRegisterBits attribute), 94
WDERRCLR (pylink.registers.AbortRegisterFlags attribute), 95
WINDOWS_32_JLINK_SDK_NAME (pylink.library.Library attribute), 12
WINDOWS_64_JLINK_SDK_NAME (pylink.library.Library attribute), 12
WINDOWS_JLINK_SDK_NAME (pylink.library.Library attribute), 12
WPUnit (pylink.structs.JLinkWatchpointInfo attribute), 77, 78
WRITE (pylink.enums.JLinkAccessFlags attribute), 78
WRITE_MEMORY (pylink.enums.JLinkCPUCapabilities attribute), 80
WRITE_REGISTERS (pylink.enums.JLinkCPUCapabilities attribute), 80
WRITE_TARGET_MEMORY_FAILED (pylink.enums.JLinkGlobalErrors attribute), 85
WriteRequest (class in pylink.protocols.swd), 60
WRONG_USER_CONFIG (pylink.enums.JLinkGlobalErrors attribute), 86

X
XSCALE (pylink.enums.JLinkCore attribute), 82
XSCALE (pylink.enums.JLinkDeviceFamily attribute), 84

Z
ZONE_NOT_FOUND_ERROR (pylink.enums.JLinkReadErrors attribute), 88
ZONE_NOT_FOUND_ERROR (pylink.enums.JLinkWriteErrors attribute), 92