pyneovi Documentation

Release 1.0

John Kemp

Oct 06, 2020

Contents

CHAPTER 1

Overview

pyneovi is a Python wrapper around the API provided by Intrepid Control Systems for communicating with their NeoVI range of devices.

There are three layers to pyneovi and application code can be written against any of them:

- API wrapper layer. This is a Python wrapper around the API provided by ICS in their DLL/shared object. Code written against this will primarily use the *neovi* module, which contains the function wrappers and associated constants. Additionally, some constants are defined in the *can* module and structures are defined in the *structures* module.
- neoVI device layer. This provides a more Pythonic implementation of the API based around objects representing neoVI devices. Code written against this will primarily use the *neodevice* module.
- ECU layer. This shifts the focus to the devices being interacted with, providing objects representing ECUs on the network. Code written against this will primarily use the neoVI device layer to set up the interface plus the *neodevice* module.

The API wrapper layer and the neoVI device layer may both be used as provided. The ECU layer, however, requires additional information about the ECUs being communicated with. This may be provided by the application code directly, but it will usually be preferable to construct a "vehicle spec" file. This describes the networks provided by the vehicle, the ECUs on those networks, and the signals made available by the ECUs. ECUs and signals may then be identified by their names rather than addresses and conversion of the values read from the ECU will be performed automatically. The *spec* module is used to create a vehicle spec.

CHAPTER 2

Examples

examples

CHAPTER 3

Submodules

3.1 neovi.can module

```
# The file is part of the pyneovi project and is provided under the MIT License terms.
# For license information see LICENSE.txt.
"""

"""

import enum

# --------------------
# Service identifiers
# --------------------

DIAG_CURRENT_DATA          = 0x01
DIAG_FREEZE_FRAME_DATA     = 0x02
DIAG_STORED_DTCS           = 0x03
DIAG_CLEAR_DTCS            = 0x04
DIAG_PENDING_DTCS          = 0x07
DIAG_VEHICLE_INFORMATION   = 0x09
DIAG_PERMANENT_DTCS        = 0x0A

# Diagnostic and communications management
DIAGNOSTIC_SESSION_CONTROL = 0x10
ECU_RESET                  = 0x11
SECURITY_ACCESS            = 0x27
COMMUNICATION_CONTROL      = 0x28
AUTHENTICATION             = 0x29
TESTER_PRESENT             = 0x3E
ACCESS_TIMING_PARAMETERS   = 0x83
SECURED_DATA_TRANSMISSION  = 0x84
CONTROL_DTC_SETTING        = 0x85
RESPONSE_ON_EVENT          = 0x86
```

(continues on next page)

(continued from previous page)

```
LINK_CONTROL                = 0x87

# Data transmission
READ_DATA_BY_ID             = 0x22
READ_MEMORY_BY_ADDRESS      = 0x23
READ_SCALING_DATA_BY_ID     = 0x24
READ_DATA_BY_ID_PERIODIC    = 0x2A
DYNAMICALLY_DEFINE_DATA_ID  = 0x2C
WRITE_DATA_BY_ID            = 0x2E
WRITE_MEMORY_BY_ADDRESS     = 0x3D

# Stored data transmission
CLEAR_DIAGNOSTIC_INFORMATION = 0x14
READ_DTC_INFORMATION         = 0x19

# Input / output control
IO_CONTROL_BY_ID            = 0x2F

# Remote activation of routine
ROUTINE_CONTROL             = 0x31

# Upload / download
REQUEST_DOWNLOAD            = 0x34
REQUEST_UPLOAD              = 0x35
TRANSFER_DATA               = 0x36
REQUEST_TRANSFER_EXIT       = 0x37
REQUEST_FILE_TRANSFER       = 0x38


def get_response_sid(request_sid: int) -> int:
    return request_sid + 0x40


NEGATIVE_RESPONSE = 0x7F


# ------------------------------
# DIAG_VEHICLE_INFORMATION PIDs
# ------------------------------

SUPPORTED_PIDS                  = 0x00
VIN_MESSAGE_COUNT               = 0x01
VIN                             = 0x02
CALIBRATION_ID_COUNT            = 0x03
CALIBRATION_ID                  = 0x04
CALIBRATION_VERIFICATION_COUNT  = 0x05
CALIBRATION_VERIFICATION        = 0x06
PERFORMANCE_TRACKING_COUNT_S    = 0x07
PERFORMANCE_TRACKING            = 0x08
ECU_NAME_COUNT                  = 0x09
ECU_NAME                        = 0x0A
PERFORMANCE_TRACKING_COUNT_C    = 0x0B


class SessionType(enum.IntEnum):
    """
    Session types that can be requested with a DIAGNOSTIC_SESSION_CONTROL message.
```

(continues on next page)

(continued from previous page)

```
    """
    Default                = 0x01
    Programming            = 0x02
    ExtendedDiagnostic     = 0x03
    SafetySystemDiagnostic = 0x04


class ControlType(enum.IntEnum):
    """
    Control types that can be used with an ID_CONTROL_BY_ID message.
    """
    ReturnControlToECU  = 0x00
    ResetToDefault      = 0x01
    FreezeCurrentState  = 0x02
    ShortTermAdjustment = 0x03


error_code_lookup = {
    0x10: 'General reject',
    0x11: 'Service not supported',
    0x12: 'Subfunction not supported',
    0x13: 'Invalid message length or format',
    0x14: 'Response too long',
    0x21: 'Busy repeat request',
    0x22: 'System state not correct (is ignition on?)',
    0x24: 'Request sequence error',
    0x25: 'No response from sub-net component',
    0x26: 'Failure prevented execution',
    0x31: 'Bad message format / request out of range',
    0x33: 'Security access denied',
    0x35: 'Invalid key',
    0x36: 'Exceeded number of attempts',
    0x37: 'Required time delay not expired',
    0x70: 'Upload/download not accepted',
    0x71: 'Transfer data suspended',
    0x72: 'General programming failure',
    0x73: 'Wrong Block Sequence Counter',
    0x78: 'Request ok, response pending',
    0x7E: 'Sub-function not supported in active session',
    0x7F: 'Service not supported in active session',
    0x80: 'Not supported in active session',    # Not standard, where did this come␣
↪from?
}
```

3.2 neovi.ecu module

Provides the *ECU* class that represents an individual ECU.

Below is an example of reading values from an ECU. Note that it requires a network spec such as the one created in the example on the *spec* module page.

```
import neovi.neodevice as neodevice
import neovi.ecu as ecu
```

(continues on next page)

(continued from previous page)

```
import neovi.spec as spec
import neovi.neovi as neovi
import json


neodevice.init_api()
dev = neodevice.find_devices(neovi.NEODEVICE_FIRE)[0]
dev.open()


input_file = open('vehicle.spec', 'rt')
data = json.load(input_file, object_hook=spec.from_json)

hvac = ecu.ECU(data['ECUs']['HVAC'], dev)

wanted_values = ['Blower Speed Output', 'External Ambient Temperature', 'Left Solar␣
↪Radiation Sensor', 'Cabin Temperature']

for value_name in wanted_values:
    result = hvac.read_data_by_id(value_name)['value']
    print("%s = %.1f %s" % (value_name, result[0], result[1]))

dev.close()
```

exception CommandError(*code*)

Bases: Exception

An error response was received after sending a message on the bus.

exception NoResponseError

Bases: Exception

No response was received after sending a message on the bus.

class ECU(*ecu_info*, *interface*, *auto_tester_present_on_auth=True*, *log_unhandled_messages=False*)

Bases: object

Represents a vehicle ECU.

Parameters

- **ecu_info** (spec.ECU) – Details of the ECU (network, address...).
- **interface** (neodevice.NeoDevice) – The neoVI device to use for communication.
- **auto_tester_present_on_auth** (*bool*) – Should a "tester present" message be sent at a regular interval if successful authentication (by an external tool) is detected?
- **log_unhandled_messages** (*bool*) – Should messages to this ECU that have no associated subscription be logged?

read_data_by_id(*identifier*)

Send a "Read Data By ID" message and return the response value(s). The return value will be a dictionary of decoded values if an identifier name or an identifier object is passed in, or the raw message bytes if an integer array is passed in.

Parameters **identifier** – This may be one of three types of object/value: 1) the name of an identifier known to the *spec.ECU* that was passed to the constructor, 2) a *spec.Identifier* object, or 3) an array of integers specifying the identifier to read (e.g. [0x98, 0x05]).

io_control_by_id(*identifier*, *value=0*, *control_type=<ControlType.ShortTermAdjustment: 3>*)
Send a "IO Control By ID" message.

Parameters

- **identifier** – This may be one of three types of object/value: 1) the name of an identifier known to the *spec.ECU* that was passed to the constructor, 2) a *spec.Identifier* object, or 3) an array of integers specifying the identifier to read (e.g. [0x98, 0x05]). Note that due to limitations of the current implementation, the value will be assumed to correspond to the first signal defined for the identifier.
- **value** (*int*) – The value to set the output to. Conversion of the value into the form required by the underlying signal will be performed here. Default = 0x00.
- **control_type** (*can.ControlType*) – The type of control to apply. Default = SHORT_TERM_ADJUSTMENT.

diagnostic_session_control(*session_type*)
Send a "Diagnostic Session Control" message.

Parameters **session_type** (*can.SessionType*) – The type of session to switch to.

security_access(*subfunction*, *key=None*)

send_tester_present()
Send a "Tester Present" message to avoid a diagnostic session timing out.

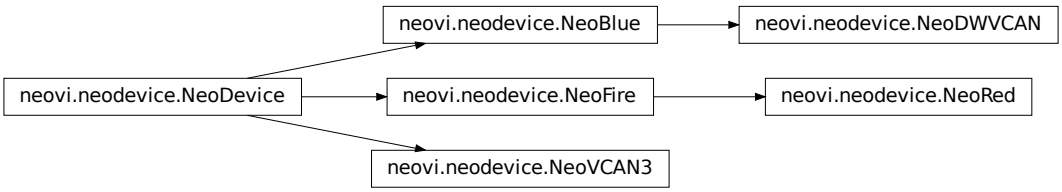
send_periodic_tester_present(*interval=2*)
Start sending a periodic "tester present" message to avoid a diagnostic session timing out. :param int interval: Period in seconds between messages.

stop_periodic_tester_present()
Stop sending the periodic "tester present" message.

3.3 neovi.neodevice module

This module provides classes representing neoVI devices, all derived from core functionality in the *NeoDevice* class. These classes form a layer of abstraction on top of the wrapper functions for the ICS DLL API defined in *neovi*.

The following neoVI classes are defined:



Below is a simple example that logs all messages to and from a specified ECU.

```
import time
import logging
```

(continues on next page)

(continued from previous page)

```
import neovi.neodevice as neodevice
import neovi.neovi as neovi
from neovi.structures import format_array, get_status_bits_set


TARGET_ECU = 0x456


def log_messages(result, msgs, errors):
    # result is -1 on success, the value returned by icsneoGetLastAPIError otherwise.
    # (See http://www.intrepidcs.com/support/ICSDocumentation/neoVIDLL/
↪apiErrorMessages.htm)
    # msgs is an array of icsSpyMessage.
    # (See structures.py)
    # errors is an array of c_int.
    # (See link as per meaning of "result")
    for msg in msgs:
        if msg.ArbIDOrHeader in [TARGET_ECU, TARGET_ECU + 8]:
            logging.debug('ArbIDOrHeader = %X, number data bytes = %X' % (msg.
↪ArbIDOrHeader, msg.NumberBytesData))
            logging.debug('Data = %s' % format_array(msg.Data))
            logging.debug('Ack = %s' % format_array(msg.AckBytes))
            logging.debug('Value = %f, misc data = %X' % (msg.Value, msg.MiscData))
            stat, stat2 = get_status_bits_set(msg)
            stat = stat + stat2
            for i in range(0, len(stat), 3):
                stats = stat[i:i+3]
                logging.debug('%s' % ', '.join(stats))
            logging.debug('')


# The below assumes the first device is the one you want, which will be true
# if there's only one attached.
neodevice.init_api()
dev = neodevice.find_devices(neovi.NEODEVICE_FIRE)[0]
dev.open()

dev.subscribe_to_all(log_messages)

# Quick hack to wait until the user hits Ctrl+C
try:
    while 1:
        time.sleep(1)
except KeyboardInterrupt:
    pass

dev.close()
```

init_api()
Load the neoVI API library and initialise it. This must be called before making any API calls.

find_devices(*types=neovi.NEODEVICE_ALL*, *auto_open=False*)
Find any attached neoVI devices.

Parameters

- **types** (*int*) – Filter the results to given types of devices. The NEODEVICE_* constants

defined in *neovi* can be OR'd together.

- **auto_open** (*bool*) – Determines whether the discovered devices should be automatically opened. If this is False then you must call *NeoDevice.open()* on any devices that are to be used.

exception OpenFailedError
Failed to open a NeoVI device.

exception InvalidConfigurationError
An invalid configuration was supplied to be written to the device.

class NeoDevice(*device*, *auto_open=False*, *launch_msg_queue_thread=True*)
Represents a generic neoVI device, providing an interface for transmitting and receiving messages as well as configuring the device.

Typically these objects would be created via a call to *find_devices()*, which returns a list of pre-constructed NeoDevice objects.

Parameters

- **device** (structures.NeoDevice) – Device identifier, as returned by *neovi.FindNeoDevices()*.
- **auto_open** (*bool*) – Determines whether the device should be automatically opened. If this is False then you must call *NeoDevice.open()* in order to open the device.
- **launch_msg_queue_thread** (*bool*) – Determines whether to start a thread to receive incoming messages. If this is False then you must process the message queue via the NeoDevice._process_msg_queue() method or fetch the messages via the *NeoDevice.get_messages()* method. Only applicable if auto_open = True.

open(*launch_msg_queue_thread=True*)
Open the device and optionally launch the message thread. This is only required if the object was constructed with auto_open = False.

Parameters **launch_msg_queue_thread** (*bool*) – Determines whether to start a thread to receive incoming messages. If this is False then you must process the message queue via the NeoDevice._process_msg_queue() method or fetch the messages via the *NeoDevice.get_messages()* method.

Raises ***OpenFailedError*** – If the device cannot be opened.

get_type()

Returns The type of device represented. See NEODEVICE_* in *neovi*.

get_settings()
Get the current device configuration.

Returns A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) a device-specific configuration object.

set_settings(*settings*, *save_to_eeprom=False*)
Set the device configuration.

Parameters

- **settings** – A device-specific configuration.
- **save_to_eeprom** (*bool*) – Determines if the configuration will be persisted across device power cycles.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

tx_raw_message(*msg*, *network_id*)
Transmits a pre-constructed message.

Parameters

- **msg** (*icsSpyMessage*) – The message to transmit.
- **network_id** (*int*) – The network to transmit on. See NETID_* in *neovi*.

Returns Status code. `neovi.SUCCESS` or an error code. See TxMessages in the ICS API documentation for possible codes (constants not yet defined within pyneovi).

tx_message(*network_id*, *dest*, *msg_type*, *payload*)
Transmits a pre-constructed message.

Parameters

- **network_id** (*int*) – The network to transmit on. See NETID_* in *neovi*.
- **dest** (*int*) – The address of the destination ECU.
- **msg_type** (*int*) – The message type to send. See `can`.
- **payload** (*list of ints*) – Up to 6 bytes to send.

Returns

Tuple of status code and transmission id. Status code is `neovi.SUCCESS` or an error code. See TxMessages in the ICS API documentation for possible codes (constants not yet defined within pyneovi). Transmission id can be safely ignored.

subscribe_to(*callback*, *network=None*, *address=None*, *msg_type=None*, *additional_bytes=None*, *auto_remove=False*, *user_data=None*)
Set a callback function to be called upon reception of a defined subset of messages. Note that this will only occur if the message thread has been started or if `NeoDevice._process_msg_queue()` is called manually. All parameters other than callback, auto_remove, and user_data define filtering criteria that will be used to determine whether to pass the message to this callback.

Parameters

- **callback** (*func*) – The method or function to be called. This must take two parameters: 1) a message as a *structures.icsSpyMessage* object, and 2) a user data parameter with no pre-defined meaning within pyneovi.
- **network** (*int*) – The network ID of interest (see NETID_* in *neovi*). If None (the default) then it will be ignored for filtering.
- **address** (*int*) – The address of the ECU of interest. If None (the default) then it will be ignored for filtering.
- **msg_type** (*int*) – The message type of interest (see `can`). If None (the default) then it will be ignored for filtering.
- **additional_bytes** (*list of ints*) – Additional payload bytes to use for filtering. May be an empty list. A value of None (the default) will be converted to an empty list automatically.
- **auto_remove** (*bool*) – If True then the subscription is removed once the first message matching the provided criteria has been received.
- **user_data** – This parameter has no pre-defined meaning within pyneovi - the value is passed to the callback function along with the received message.

subscribe_to_all(*callback*)
Set a callback function to be called upon reception of any message. Note that this will only occur if the message thread has been started or if `NeoDevice._process_msg_queue()` is called manually.

Parameters **callback** (*func*) – The method or function to be called. This must take three parameters: 1) a statue code of either `neovi.SUCCESS` or an error code (see TxMessages in the ICS API documentation for possible codes (constants not yet defined within pyneovi)), 2) a list of messages as *structures.icsSpyMessage* objects, and 3) a list of errors as integers (see Error Messages in the ICS API documentation for a complete error list).

get_messages()
Fetch pending received messages. Note that if the message thread was launched then this should not be called.

Returns

A tuple containing 1) a statue code of either `neovi.SUCCESS` or an error code (see TxMessages in the ICS API documentation for possible codes (constants not yet defined within pyneovi)), 2) a list of messages as *structures.icsSpyMessage* objects, and 3) a list of errors as integers (see Error Messages in the ICS API documentation for a complete error list).

setup_networks(*network_settings*)

get_firmware_version()
Return the firmware version of the device.

Returns A firmware info object if the call succeeded, otherwise None.

Return type *structures.APIFirmwareInfo*

get_dll_firmware_version()
Return the firmware version stored within the DLL API.

Returns A firmware info object if the call succeeded, otherwise None.

Return type *structures.APIFirmwareInfo*

force_firmware_update()
Force the firmware on the device to be updated to the version stored in the DLL API.

close()
Close the device and shutdown the message thread (if there is one).

class NeoFire(*device*, *auto_open=True*, *launch_msg_queue_thread=True*)
Represents a neoVI Fire device. Should be used if settings specific to the neoVI Fire must be read/written.

Base class: *NeoDevice*

get_settings()
Get the current device configuration.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) a *structures.SFireSettings* object.

set_settings(*settings*, *save_to_eeprom=False*)
Set the device configuration.

Parameters

- **settings** (`structures.SFireSettings`) – The new configuration.

- **save_to_eeprom** (*bool*) – Determines if the configuration will be persisted across device power cycles.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

class NeoRed(*device*, *auto_open=True*, *launch_msg_queue_thread=True*)
Represents a neoVI Red device. Should be used if settings specific to the neoVI Red must be read/written.

Base class: *NeoFire*

class NeoVCAN3(*device*, *auto_open=True*, *launch_msg_queue_thread=True*)
Represents a ValueCAN3 device. Should be used if settings specific to the ValueCAN3 must be read/written.

Base class: *NeoDevice*

get_settings()
Get the current device configuration.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) a *structures.SVCAN3Settings* object.

set_settings(*settings*, *save_to_eeprom=False*)
Set the device configuration.

Parameters

- **settings** (structures.SVCAN3Settings) – The new configuration.
- **save_to_eeprom** (*bool*) – Determines if the configuration will be persisted across device power cycles.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

class NeoBlue(*device*, *auto_open=True*, *launch_msg_queue_thread=True*)
Represents a neoVI Blue device. Should be used if settings specific to the neoVI Blue must be read/written.

Base class: *NeoDevice*

set_settings(*settings*, *save_to_eeprom=False*)
Set the device configuration.

Parameters

- **settings** – An array of configuration bytes (see Configuration Array in the ICS API documentation).
- **save_to_eeprom** (*bool*) – Determines if the configuration will be persisted across device power cycles.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

Raises ***InvalidConfigurationError*** – If the size of the settings array is incorrect.

class NeoDWVCAN(*device*, *auto_open=True*, *launch_msg_queue_thread=True*)

3.4 neovi.neovi module

This module wraps the low-level interface to the neoVI range of devices. It is unlikely that the functions defined here will be directly used as *neodevice.NeoDevice* and related classes offer a more convenient interface.

Of more interest will be the various constants defined here.

Device type identifiers (used by *neodevice.find_devices()*):

NEODEVICE_BLUE	NEODEVICE_SW_VCAN	NEODEVICE_FIRE
NEODEVICE_VCAN3	NEODEVICE_YELLOW	NEODEVICE_RED
NEODEVICE_ECU	NEODEVICE_IEVB	NEODEVICE_PENDANT
NEODEVICE_VIRTUAL_NEOVI	NEODEVICE_ECUCHIP_UART	NEODEVICE_PLASMA_1_11
NEODEVICE_FIRE_VNET	NEODEVICE_NEOANALOG	NEODEVICE_CT_OBD
NEODEVICE_PLASMA_1_12	NEODEVICE_PLASMA_1_13	NEODEVICE_ION_2
NEODEVICE_RADSTAR	NEODEVICE_ION_3	NEODEVICE_VCANFD
NEODEVICE_ECU15	NEODEVICE_ECU25	NEODEVICE_EEVB
NEODEVICE_VCANRF	NEODEVICE_FIRE2	NEODEVICE_FLEX
NEODEVICE_RADGALAXY	NEODEVICE_NEOECUCHIP	

Additional special identifiers are available:

NEODEVICE_ANY_PLASMA	NEODEVICE_ANY_ION	NEODEVICE_ALL

Network IDs:

NETID_DEVICE		
NETID_MSCAN	NETID_SWCAN	NETID_SWCAN2
NETID_LSFTCAN	NETID_LSFTCAN2	NETID_HSCAN
NETID_HSCAN2	NETID_HSCAN3	NETID_HSCAN4
NETID_HSCAN5	NETID_HSCAN6	NETID_HSCAN7
NETID_LIN	NETID_LIN2	NETID_LIN3
NETID_LIN4	NETID_LIN5	NETID_LIN6
NETID_ETHERNET	NETID_OP_ETHERNET1	NETID_OP_ETHERNET2
NETID_OP_ETHERNET3	NETID_OP_ETHERNET4	NETID_OP_ETHERNET5
NETID_OP_ETHERNET6	NETID_OP_ETHERNET7	NETID_OP_ETHERNET8
NETID_OP_ETHERNET9	NETID_OP_ETHERNET10	NETID_OP_ETHERNET11
NETID_OP_ETHERNET12	NETID_ETHERNET_DAQ	
NETID_RS232	NETID_UART	NETID_UART2
NETID_UART3	NETID_UART4	
NETID_FLEXRAY	NETID_FLEXRAY1A	NETID_FLEXRAY1B
NETID_FLEXRAY2	NETID_FLEXRAY2A	NETID_FLEXRAY2B
NETID_3G_RESET_STATUS	NETID_3G_FB_STATUS	NETID_3G_APP_SIGNAL_STATUS
NETID_3G_READ_DATALINK_CM_TX_MSG	NETID_3G_READ_DATALINK_CM_RX_MSG	NETID_3G_LOGGING_OVERFLOW
NETID_3G_READ_SETTINGS_EX		
NETID_MOST	NETID_MOST25	NETID_MOST50
NETID_MOST150		
NETID_FORDSCP	NETID_J1708	NETID_AUX
NETID_JVPW	NETID_ISO	NETID_ISO2
NETID_ISO3	NETID_ISO4	NETID_ISOPIC
NETID_MAIN51	NETID_RED	NETID_SCI
NETID_ISO14230	NETID_RED_APP_ERROR	NETID_CGI
NETID_DATA_TO_HOST	NETID_TEXTAPI_TO_HOST	NETID_I2C1
NETID_SPI1	NETID_RED_VBAT	NETID_GMFSA
NETID_TCP		

class BitRate

Bases: `enum.IntEnum`

Available bit rates for NeoVI network interfaces.

BR_20000 = 0

BR_33333 = 1

BR_50000 = 2

BR_62500 = 3

BR_83333 = 4

BR_100000 = 5

BR_125000 = 6

BR_250000 = 7

BR_500000 = 8

BR_800000 = 9

BR_1000000 = 10

DEFAULT = 8

network_name(*nid*)
Lookup the friendly name for a network ID.

Parameters **nid** (*int*) – The network ID to look up.

Returns The friendly name for the network.

Return type str

InitializeAPI()
Initialise the API and prepares it for use. It must be called before any other API calls are made.

Note that the underlying neoVI library only requires this on Linux, but pyneovi requires it on all platforms. This provides a consistent interface without platform-specific checks in application code and also allows pyneovi to perform its own initialisation.

ShutdownAPI()
Shut down the API and releases any resources allocated during its use.

Note that the underlying neoVI library only requires this on Linux, but pyneovi requires it on all platforms. This provides a consistent interface without platform-specific checks in application code.

GetLastAPIError(*hObject*)
Get the error generated by the last API call.

API errors are generated and stored on a 'per-thread' basis. The calling thread will only receive errors generated within it's own context. If a new API error is generated before the previous error has been retrieved, the previous error will be lost. All errors generated can still be retrieved using GetErrorMessages. However, GetErrorMessages will return errors generated in all threads, not just the current thread.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns A 2-tuple consisting of i) a boolean indicating if a new error was generated since the port was opened or GetLastAPIError was last called and ii) the error generated by the last API call.

GetErrorMessages(*hObject*)
Read the neoVI DLL error message queue. The error queue will be reset after this function is called.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns A 2-tuple consisting of i) a boolean indicating success or failure of the function and ii) a list of errors generated by all threads.

FindNeoDevices(*types*)
Find any attached neoVI devices.

Parameters **types** (*int*) – Filter the results to given types of devices. The NEODEVICE_* constants can be OR'd together.

Returns A list of *structures.NeoDevice* objects.

OpenNeoDevice(*device*, *network_ids=None*, *config_read=True*)
Open a communication link the a neoVI device.

Parameters

- **device** (structures.NeoDevice) – A device information structure returned by *neovi.FindNeoDevices()*.
- **network_ids** – An array of network IDs to assign to the available networks. The default is to use the predefined network IDs.
- **config_read** (*bool*) – Should the device's configuration be read before enabling it? It is recommended that this be set to true.

Returns An object handle for use in other API functions or None if the call failed.

ClosePort (*hObject*)
Close the communication link with the neoVI hardware.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns A 2-tuple containing i) a bool representing the success or failure of the function and ii) a list of error codes if the bool is false (empty otherwise). See Error Messages in the ICS API documentation.

FreeObject (*hObject*)
Release any resources that were allocated by OpenNeoDevice.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

GetMessages (*hObject*)
Read messages from the neoVI device. The driver object will hold 20,000 received messages before it will generate an RX buffer overflow error.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns

A 3-tuple containing i) either SUCCESS or an error code (see Error Messages in the ICS API documentation), ii) a list of *structures.icsSpyMessage* objects, and iii) a list of error codes obtained via *neovi.GetErrorMessages()*.

TxMessages (*hObject*, *msgs*, *network*, *num_msgs=1*)
Transmit messages asynchronously to vehicle networks using the neoVI hardware.

After the messages has been transmitted there will be a transmit report message returned from the device. The transmit report will be read out with *neovi.GetMessages()*. Any message read which has the SPY_STATUS_TX_MSG bit set in the status bitfield is a transmit report.

You can also identify a particular transmitted message with DescriptionID field. This two byte field (only 14 bits are used) allows the programmer to assign an arbitrary number to a message. This number is then returned in the transmit report.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **msgs** – A list of *structures.icsSpyMessage* objects.
- **network** (*int*) – The network to transmit the message on.
- **num_msgs** (*int*) – The number of messages to transmit. This should always be 1 unless you are transmitting a long Message on ISO or J1708.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

WaitForRxMessagesWithTimeOut (*hObject*, *timeout_ms*)
Wait a specified amount of time for received messages.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **timeout_ms** (*int*) – A timeout in ms to wait for a received message before returning.

Returns

Either MESSAGES_RECVD, NO_MESSAGES, or an error code (see Error Messages in the ICS API documentation).

GetTimeStampForMsg(*hObject*, *msg*)
Calculate the timestamp for a message, based on the connected hardware type, and convert it to a usable variable.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **msg** – A *structures.icsSpyMessage* object.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) the calculated timestamp.

EnableNetworkRXQueue(*hObject*, *enable*)
Enable or disable the received message queue (and thus reception of network traffic). This applies only to the application that called this function, other applications are not affected.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **enable** (*bool*) – Specifies whether to enable (true) or disable (false) reception of network traffic.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

GetISO15765Status(*hObject*, *network*, *clear_rx_status*)
Get, and optionally clear, the current receive status of the CAN ISO15765-2 network layer.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **network** (*int*) – Which CAN network the status is requested for.
- **clear_rx_status** (*bool*) – If true will clear the receive status and reset the receive state machine.

Returns The receive status. See GetISO15765Status in the ICS API documentation.

SetISO15765RxParameters(*hObject*, *network*, *enable*, *msg_filter*, *flow_ctx_msg*, *timeout_ms*, *flow_control_block_size*, *uses_extended_addressing*)
set the parameters necessary to control CAN ISO15765-2 network layer message reception.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **network** (*int*) – The network to transmit the message on.
- **enable** (*bool*) – If true ISO15765 services will be enabled for this network.

See SetISO15765RxParameters in the ICS API documentation for full descriptions of the remaining parameters.

GetConfiguration(*hObject*)
Read the configuration from a NeoVI Blue or ValueCAN device.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) an array of 1024 configuration bytes (see Configuration Array in the ICS API documentation).

SendConfiguration (*hObject*, *pData*)
Send configuration information to a NeoVI Blue or ValueCAN device.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **pData** – An array of configuration bytes (see Configuration Array in the ICS API documentation).

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

GetFireSettings (*hObject*)
Read the configuration settings from a neoVI Fire device.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) a *structures.SFireSettings* object.

SetFireSettings (*hObject*, *pSettings*, *bSaveToEEPROM*)
Write configuration settings to a neoVI Fire device.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **pSettings** (*structures.SFireSettings*) – The configuration to write.
- **bSaveToEEPROM** (*bool*) – Overwrite the stored EEPROM values so that these settings persist across power-cycles.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

GetVCAN3Settings (*hObject*)
Read the configuration settings from a ValueCAN3 device.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) a *structures.SVCAN3Settings* object.

SetVCAN3Settings (*hObject*, *pSettings*, *bSaveToEEPROM*)
Write configuration settings to a ValueCAN3 device.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **pSettings** (*structures.SVCAN3Settings*) – The configuration to write.
- **bSaveToEEPROM** (*bool*) – Overwrite the stored EEPROM values so that these settings persist across power-cycles.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

SetBitRate (*hObject*, *iBitRate*, *iNetworkID*)
Set the bit rates for networks.

Parameters

- **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.
- **iBitRate** (*int*) – The new bit rate setting.
- **iNetworkID** (*int*) – The network to set the bit rate on.

Returns

Either SUCCESS or an error code (see Error Messages in the ICS API documentation).

GetHWFirmwareInfo (*hObject*)
Return the firmware version of the open neoVI device.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) a *structures.APIFirmwareInfo* object.

GetDLLFirmwareInfo (*hObject*)
Return the firmware version stored within the DLL API.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

Returns

A 2-tuple of i) either SUCCESS or an error code (see Error Messages in the ICS API documentation) and ii) a *structures.APIFirmwareInfo* object.

ForceFirmwareUpdate (*hObject*)
Force the firmware on a neoVI device to be updated to the version stored in the DLL API.

Parameters **hObject** – An object handle returned by *neovi.OpenNeoDevice()*.

GetDLLVersion ()
Return the software version of the DLL.

Returns The version of the API DLL as an integer.

Return type int

3.5 neovi.spec module

This module provides classes and functions to allow a network specification to be created for use with pyneovi. A network specification defines the valid networks for a given model of vehicle, along with the ECUs on those networks and the identifiers and signals made available by the ECUs.

Below is an example of a simple network specification. Only the medium speed CAN bus is defined and only a single ECU is associated with that network. The ECU provides four identifiers, each made up of a single signal.

```
from neovi.spec import *
from neovi.neovi import NETID_MSCAN, BitRate
import json


networks = {NETID_MSCAN: {'bitrate': BitRate.BR_125000}}

hvac_ecu = ECU(NETID_MSCAN, 0x456, 'HVAC')

hvac_identifiers = [
    Identifier('Blower Speed Output',          0x9601, [Signal(units='%', m=100./255.
↪)], IOType.ReadWrite),
    Identifier('External Ambient Temperature', 0x9628, [Signal(units='Deg C', m=0.
↪25)], IOType.ReadWrite),
    Identifier('Left Solar Radiation Sensor',  0x9734, [Signal(units='W', m=10, max_
↪val=1250)]),
    Identifier('Cabin Temperature',            0x97A5, [Signal(units='Deg C',␣
↪length=16, m=0.01, b=-100, min_val=-50, max_val=100)]),
]


ECUs = {
    'HVAC': {'ecu': hvac_ecu, 'identifiers': hvac_identifiers}
}


output_file = open('vehicle.spec', 'wt')
json.dump({'networks': networks, 'ECUs': ECUs}, output_file, cls=PyNeoViJSONEncoder)
output_file.close()
```

See the *ecu* module page for an example of using a network specification.

class SignalType
Bases: enum.Enum

An enumeration.

Analog = 0

class ValueType
Bases: enum.Enum

An enumeration.

UnsignedInt = 0

class IOType
Bases: enum.Enum

An enumeration.

Read = 0

Write = 1

ReadWrite = 2

exception NonByteBoundarySignalError
Bases: Exception

exception ValueTooLargeError
Bases: Exception

exception UnsupportedValueTypeError
Bases: Exception

class Signal(*self*, *name='value'*, *signal_type=ANALOG*, *value_type=UNSIGNED_INT*, *start=0*, *length=8*, *min=None*, *max=None*, *units=''*, *m=1*, *b=0*, *description=''*)
Bases: object

Represents a signal. An *Identifier* will contain one or more signals.

Parameters

- **name** (*str*) – The name of this signal.
- **signal_type** (*SignalType*) – The type of signal. Currently, only *SignalType.Analog* is supported.
- **value_type** (*ValueType*) – The type of value returned for the signal. Currently only *ValueType.UnsignedInt* is supported.
- **start** (*int*) – The bit within the identifier's aggregate value at which this signal starts.
- **length** (*int*) – The length in bits of this signal.
- **min_val** (*float*) – The minimum value for this signal. This is not enforced and is for reference only. If not specified then *b* is used as the default value.
- **max_val** (*float*) – The minimum value for this signal. This is not enforced and is for reference only. If not specified then *(255. * m) + b* is used as the default value.
- **units** (*str*) – The units of the signal once converted.
- **m** (*float*) – The slope for conversion of the signal into engineering units.
- **b** (*float*) – The offset for conversion of the signal into engineering units.
- **description** (*str*) – A human-readable description of the signal.

Raises ***NonByteBoundarySignalError*** – If length or start are not integer multiples of 8. This is a limitation of the current implementation.

decode_signal_bytes(*signal_bytes*)
Converts the individual data bytes into the final value for the signal.

Parameters **signal_bytes** (*list of ints*) – The bytes received from the ECU.

Returns Tuple of final value (float) and the units of the signal (str).

Raises ***UnsupportedValueTypeError*** – If the signal is of a type that is currently unsupported.

encode_value(*value*)
Converts a value into the individual data bytes required for transmission.

Parameters **value** (*float/int*) – The value for conversion.

Raises

- ***UnsupportedValueTypeError*** – If the signal is of a type that is currently unsupported.
- ***ValueTooLargeError*** – If the value does not fit into the number of bits specified for the signal.

class Identifier(*self*, *name*, *data_id*, *signals*, *io_type = IOType.Read*)
Bases: object

Represents an identifier (an example of which would be "Blower Speed Output" in a HVAC ECU). The value of an identifier can consist of multiple individual signals but often there is only a single signal associated with an identifier.

Parameters

- **name** (*str*) – The name of the identifier.
- **data_id** (*int*) – The address (internal to the ECU) with which this identifier is associated.
- **signals** (list of *Signal*) – The signals that form this identifier.
- **io_type** (IOType) – Allowed IO for this identifier.

get_data_id_bytes()

Returns The data_id as individual bytes. Used when constructing a message.

class ECU(*network*, *address*, *short_desc=''*, *long_desc=''*)
Bases: object

Represents an ECU for the purpose of building a network specification. Note that if you need an object that allows you to communicate with the ECU then you're looking for *ecu.ECU*.

Parameters

- **network** (*int*) – The network to transmit on. See NETID_* in *neovi*.
- **address** (*int*) – The address of the ECU.
- **short_desc** (*str*) – A short name for the ECU (e.g. "HVAC").
- **long_desc** (*str*) – A long description for the ECU (e.g. including model number or other information).

get_network()

get_request_id()

get_response_id()

class PyNeoViJSONEncoder(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)
Bases: json.encoder.JSONEncoder

Can be passed to json.dump as the cls parameter to handle the additional object types defined here.

default(*o*)
Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

from_json(*json_object*)
Can be passed to json.load as the object_hook parameter to handle the additional object types defined here.

3.6 neovi.structures module

Classes provided by this module:

- *NeoDevice*
- *icsSpyMessage*
- *spyFilterLong*
- *CAN_SETTINGS*
- *SWCAN_SETTINGS*
- *LIN_SETTINGS*
- *ISO9141_KEYWORD2000__INIT_STEP*
- *ISO9141_KEYWORD2000_SETTINGS*
- *UART_SETTINGS*
- *STextAPISettings*
- *SFireSettings*
- *SVCAN3Settings*

array_equal(*a1*, *a2*)

format_array(*array*)

get_status_bits_set(*msg*)

class NeoDevice
Bases: _ctypes.Structure

DeviceType
Structure/Union member

Handle
Structure/Union member

MaxAllowedClients
Structure/Union member

NumberOfClients
Structure/Union member

SerialNumber
Structure/Union member

class icsSpyMessage
Bases: _ctypes.Structure

AckBytes
Structure/Union member

ArbIDOrHeader
Structure/Union member

ColorID
Structure/Union member

Data
Structure/Union member

DescriptionID
Structure/Union member

MessagePieceID
Structure/Union member

MiscData
Structure/Union member

NetworkID
Structure/Union member

NodeID
Structure/Union member

NumberBytesData
Structure/Union member

NumberBytesHeader
Structure/Union member

Protocol
Structure/Union member

StatusBitField
Structure/Union member

StatusBitField2
Structure/Union member

TimeHardware
Structure/Union member

TimeHardware2
Structure/Union member

TimeStampHardwareID
Structure/Union member

TimeStampSystemID
Structure/Union member

TimeSystem
Structure/Union member

TimeSystem2
Structure/Union member

Value
Structure/Union member

class spyFilterLong
Bases: `_ctypes.Structure`

ByteDataLSB
Structure/Union member

ByteDataLength
Structure/Union member

ByteDataMSB
Structure/Union member

ByteDataMaskLSB
Structure/Union member

ByteDataMaskMSB
Structure/Union member

ExpectedLength
Structure/Union member

FrameMaster
Structure/Union member

Header
Structure/Union member

HeaderLength
Structure/Union member

HeaderMask
Structure/Union member

MiscData
Structure/Union member

MiscDataMask
Structure/Union member

NetworkID
Structure/Union member

NodeID
Structure/Union member

Status2Mask
Structure/Union member

Status2Value
Structure/Union member

StatusMask
Structure/Union member

StatusValue
Structure/Union member

bStuff2
Structure/Union member

bUseArbIdRangeFilter
Structure/Union member

class CAN_SETTINGS
Bases: `_ctypes.Structure`

BRP
Structure/Union member

Baudrate
Structure/Union member

Mode
Structure/Union member

NetworkType
Structure/Union member

SetBaudrate
Structure/Union member

TqProp
Structure/Union member

TqSeg1
Structure/Union member

TqSeg2
Structure/Union member

TqSync
Structure/Union member

auto_baud
Structure/Union member

class SWCAN_SETTINGS
Bases: `_ctypes.Structure`

BRP
Structure/Union member

Baudrate
Structure/Union member

Mode
Structure/Union member

NetworkType
Structure/Union member

SetBaudrate
Structure/Union member

TqProp
Structure/Union member

TqSeg1
Structure/Union member

TqSeg2
Structure/Union member

TqSync
Structure/Union member

auto_baud
Structure/Union member

high_speed_auto_switch
Structure/Union member

class LIN_SETTINGS
Bases: _ctypes.Structure

Baudrate
Structure/Union member

MasterResistor
Structure/Union member

Mode
Structure/Union member

brgh
Structure/Union member

spbrg
Structure/Union member

class ISO9141_KEYWORD2000__INIT_STEP
Bases: _ctypes.Structure

k
Structure/Union member

l
Structure/Union member

time_500us
Structure/Union member

class ISO9141_KEYWORD2000_SETTINGS
Bases: _ctypes.Structure

Baudrate
Structure/Union member

brgh
Structure/Union member

chksum_enabled
Structure/Union member

init_step_count
Structure/Union member

init_steps
Structure/Union member

p2_500us
Structure/Union member

p3_500us
Structure/Union member

p4_500us
Structure/Union member

spbrg
Structure/Union member

class UART_SETTINGS
Bases: _ctypes.Structure

Baudrate
Structure/Union member

bOptions
Structure/Union member

brgh
Structure/Union member

flow_control
Structure/Union member

parity
Structure/Union member

reserved_1
Structure/Union member

spbrg
Structure/Union member

stop_bits
Structure/Union member

class STextAPISettings
Bases: `_ctypes.Structure`

Reserved0
Structure/Union member

Reserved1
Structure/Union member

Reserved2
Structure/Union member

Reserved3
Structure/Union member

Reserved4
Structure/Union member

can1_options
Structure/Union member

can1_rx_id
Structure/Union member

can1_tx_id
Structure/Union member

can2_options
Structure/Union member

can2_rx_id
Structure/Union member

can2_tx_id
Structure/Union member

can3_options
Structure/Union member

can3_rx_id3
Structure/Union member

can3_tx_id3
Structure/Union member

can4_options
Structure/Union member

can4_rx_id4
Structure/Union member

can4_tx_id4
Structure/Union member

network_enables
Structure/Union member

class SFireSettings
Bases: `_ctypes.Structure`

ain_sample_period
Structure/Union member

ain_threshold
Structure/Union member

can1
Structure/Union member

can2
Structure/Union member

can3
Structure/Union member

can4
Structure/Union member

cgi_baud
Structure/Union member

cgi_chksum_enable
Structure/Union member

cgi_enable_reserved
Structure/Union member

cgi_rx_ifs_bit_times
Structure/Union member

cgi_tx_ifs_bit_times
Structure/Union member

fast_init_network_enables_1
Structure/Union member

fast_init_network_enables_2
Structure/Union member

iso15765_separation_time_offset
Structure/Union member

iso9141_kwp_enable_reserved
Structure/Union member

iso9141_kwp_settings
Structure/Union member

iso9141_kwp_settings2
Structure/Union member

iso9141_kwp_settings_3
Structure/Union member

iso9141_kwp_settings_4
Structure/Union member

iso_msg_termination
Structure/Union member

iso_msg_termination_2
Structure/Union member

iso_msg_termination_3
Structure/Union member

iso_msg_termination_4
Structure/Union member

iso_parity
Structure/Union member

iso_parity_2
Structure/Union member

iso_parity_3
Structure/Union member

iso_parity_4
Structure/Union member

iso_tester_pullup_enable
Structure/Union member

lin1
Structure/Union member

lin2
Structure/Union member

lin3
Structure/Union member

lin4
Structure/Union member

lsftcan
Structure/Union member

misc_io_analog_enable
Structure/Union member

misc_io_initial_ddr
Structure/Union member

misc_io_initial_latch
Structure/Union member

misc_io_on_report_events
Structure/Union member

misc_io_report_period
Structure/Union member

network_enabled_on_boot
Structure/Union member

network_enables
Structure/Union member

network_enables_2
Structure/Union member

perf_en
Structure/Union member

pwm_man_timeout
Structure/Union member

pwr_man_enable
Structure/Union member

swcan
Structure/Union member

text_api
Structure/Union member

uart
Structure/Union member

uart2
Structure/Union member

class SVCAN3Settings
Bases: `_ctypes.Structure`

can1
Structure/Union member

can2
Structure/Union member

iso15765_separation_time_offset
Structure/Union member

misc_io_initial_ddr
Structure/Union member

misc_io_initial_latch
Structure/Union member

misc_io_on_report_events
Structure/Union member

misc_io_report_period
Structure/Union member

network_enabled_on_boot
Structure/Union member

network_enables
Structure/Union member

perf_en
Structure/Union member

class APIFirmwareInfo
Bases: _ctypes.Structure

iAppMajor
Structure/Union member

iAppMinor
Structure/Union member

iBoardRevMajor
Structure/Union member

iBoardRevMinor
Structure/Union member

iBootLoaderVersionMajor
Structure/Union member

iBootLoaderVersionMinor
Structure/Union member

iMainFirmChkSum
Structure/Union member

iMainFirmDateDay
Structure/Union member

iMainFirmDateHour
Structure/Union member

iMainFirmDateMin
Structure/Union member

iMainFirmDateMonth
Structure/Union member

iMainFirmDateSecond
Structure/Union member

iMainFirmDateYear
Structure/Union member

iManufactureDay
Structure/Union member

iManufactureMonth
Structure/Union member

iManufactureYear
Structure/Union member

iType
Structure/Union member

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Contact

Web: http://kempj.co.uk

Email: kemp@kempj.co.uk

Python Module Index

n

Index

K

L

M

N

O

P

R

S

T

U

V

W