Test Report on the Module com_lib.serial_port_com

Conventions

Each test is defined following the same format. Each test receives a unique test identifier and a reference to the ID(s) of the requirements it covers (if applicable). The goal of the test is described to clarify what is to be tested. The test steps are described in brief but clear instructions. For each test it is defined what the expected results are for the test to pass. Finally, the test result is given, this can be only pass or fail.

The test format is as follows:

Test Identifier: TEST-[I/A/D/T]-XYZ

Requirement ID(s): REQ-uvw-xyz

Verification method: I/A/D/T

Test goal: Description of what is to be tested

Expected result: What test result is expected for the test to pass

Test steps: Step by step instructions on how to perform the test

Test result: PASS/FAIL

The test ID starts with the fixed prefix 'TEST'. The prefix is followed by a single letter, which defines the test type / verification method. The last part of the ID is a 3-digits *hexadecimal* number (0..9|A..F), with the first digit identifing the module, the second digit identifing a class / function, and the last digit - the test ordering number for this object. E.g. 'TEST-T-112'. Each test type has its own counter, thus 'TEST-T-112' and 'TEST-A-112' tests are different entities, but they refer to the same object (class or function) within the same module.

The verification method for a requirement is given by a single letter according to the table below:

Term	Definition
Inspection (I)	Control or visual verification
Analysis (A)	Verification based upon analytical evidences
Test (T)	Verification of quantitative characteristics with quantitative measurement
Demonstration (D)	Verification of operational characteristics without quantitative measurement

Tests preparations

In order to perform the tests the following preparations must be made.

Implement a sub-class of **serial.Serial** wrapper class, which must use the mock serial object **com_lib.mock_serial.MockSerial** instead.

Implement the relevant test cases using this sub-class as the test target in the same unit-test module ut002_serial_port_com.py.

Implement the functional testing of the port checker function - make a call to this function, print-out the results and compare them with the OS-provided information (Device Manager, etc.). See ft001_serial_port_com.py.

WARNING: the timing related tests are defined specifically for the Linux Mint environment. They may fail under over OS.

Test definitions (Analysis)

Test Identifier: TEST-A-200

Requirement ID(s): REQ-FUN-200

Verification method: A

Test goal: Check the completeness of the implementation of the functionality required from the module.

Expected result: The module provides the function to obtain a list of the ports, to which the USB devices able to communicate via virtual serial port connection are connected. The module provides the class wrapping **serial.Serial** class' functionality and implementing the synchronous and asynchronous data exchange modes as defined in the requirements.

Test steps: Analyze the source code. Implement and execute the tests defined by the tests TEST_T-210, (...). All those tests as well as TEST-A-201 should pass.

Test result: PASS

Test Identifier: TEST-A-201

Requirement ID(s): REQ-FUN-201

Verification method: A

Test goal: Check that the module wraps the functionality of the **PySerial** library, but any other API-compatible library can be used as a replacement.

Expected result: The module wrapper class indeed uses **serial.Serial** class' functionality, however, it can be sub-classed to use a different but API compatible class.

Test steps: Analyze the source code. Run the unit-tests defined in the ut002_serial_port_com.py.

Test result: PASS

Test definitions (Test)

Test Identifier: TEST-T-210

Requirement ID(s): REQ-FUN-210

Verification method: T

Test goal: Check that the USB devices connected to the PC are detected, if data can be send into and received from them.

Expected result: The tested function returns a list of 3-elements tuples identifing the port paths, VID and PID of the connected devices.

Test steps: Connect, at least, one USB device, to wich such communication is possible. Execute ft001_serial_port_com.py module. Check the results: post path, VID and PID. Check the system information using OS-specific methods (like Device Manager in)

Test result: PASS

Test Identifier: TEST-T-220

Requirement ID(s): REQ-FUN-220, REQ-FUN-227

Verification method: T

Test goal: Check that the **serial.Serial** wrapper class indeed uses this class, but the actual underlying API can be replaced by a compatible class, which uses b'\x00' terminated bytestrings for the communication.

Expected result: The implementation is clear in the source code. Replacement of the actual **serial.Serial** class by the mock serial class in theut002_serial_port_com.py is successfull.

Test steps: Run the said unit-test set. All tests must pass.

Test result: PASS

Test Identifier: TEST-T-221

Requirement ID(s): REQ-FUN-221, REQ-FUN-222, REQ-FUN-223

Verification method: T

Test goal: Check the connection settings are preserved when the connection is explicitely or implicitely (error) closed, and it can be safely re-opened.

Expected result:

- Property *IsOpen* properly reflects the status of the connection
- Property *Settings* returns the same dictionary upon re-opening as before closing
- The closed connection due to an error or explicit request from user can be re-opened and used further
- The cached data is cleared

Test steps:

- Instantiate the class with the following arguments 'mock', port = 'whatever', timeout = None, write_timeout = None, baudrate = 115200, xonxoff = True, something = 1
- Check that property *IsOpen* is True

- Obtain the value of the property *Settings*, compare it with the dictionary {'port' : 'mock', 'timeout' : 0, 'write_timeout' : 0, 'baudrate' : 115200, 'xonxoff' : True, 'something' : 1}
- Close the connection, check that *IsOpen* is False
- Re-open the connection, check that *IsOpen* is True and *Settings* is the same dictionary
- Delete the instance
- Instantiate the class with the single argument 'mock2'
- Check that property *IsOpen* is True
- Obtain the value of the property *Settings*, compare it with the dictionary {'port' : 'mock2', 'timeout' : 0, 'write_timeout' : 0, 'baudrate' : 9600}
- Send 10 short strings in asynchronous mode
- Try to send a short string in the synchronous mode with a short timeout (0.1 sec). The sub-class of **serial.SerialTimeoutException** should be raised.
- Check that *IsOpen* is False
- Try to send a short string in the synchronous mode with a long timeout (1 sec).
- Check the result - the package index must be 1, check that *IsOpen* is True and *Settings* is the same dictionary
- Send asynchronously any short string, wait for 1 sec
- Try to send (asynchronously) an integer (unsupported input type). The sub-class of **TypeError** should be raised.
- Check that *IsOpen* is False, but *Settings* is the same dictionary
- Wait 1 sec. Call *getResponse*() method - check that it returns None.
- Check that *IsOpen* is True and *Settings* is the same dictionary.
- Send asynchronously any short string, wait for 1 sec
- Call *getResponse*() method with the keyword argument *ReturnType* = **int** (unsupported type). The sub-class of **TypeError** should be raised.
- Check that *IsOpen* is False, but *Settings* is the same dictionary
- Call *sendSync*() method with a short string and *ReturnType* = **str** arguments - check that it returns the same string and the package index is 1.
- Check that *IsOpen* is True and *Settings* is the same dictionary.
- Delete the instance

Implemented as method Test_SimpleCOM_API.test_Reopening() in ut002_serial_port_com.py

Test result: PASS

Test Identifier: TEST-T-222

Requirement ID(s): REQ-FUN-224, REQ-FUN-225

Verification method: T

Test goal: Proper implementation of the asynchronous sending and receving

Expected result: The following functionality is implemented:

- Multiple packages can be sent in a sequence without checking the response
- The sending method is non-blocking, and it returns the control almost immediately
- Received responses are accumulated in order of incoming untill they are claimed

- The response(s) can be checked at any time using a non-blocking method, which can return either:
  - **None** value if there are no unclaimed responses at the moment, OR
  - The earliest received and not yet claimed response
- The number of sent and received packages is tracked, thus the response can be attributed to the sending by the corresponding number

Test steps: Perform the following test:

- Instantiate the class using 'mock' port name and w/o *baudrate* keyword argument (9600 by default)
- Send three short but different string messages using asynchronous method *send*() and store them together with the returned sent indexes
- Repeatedly check for the response (asynchronously) using *getResponse*(ReturnType = str) until a non-**None** values is returned, which should be a tuple of a string and an integer
- Compare the received pair with the first sent message and its sent index - should be the same values
- Repeat checking for response until the second is received - compare with the second sent pair
- Repeat the process for the third sending / response pair
- Check the response 100 more time - each time **None** value must be returned
- Delete the created instance
- Repeat the entire test using different *baudrate* values during instantiation: 50, 2400, 115200

Implemented as method Test_SimpleCOM_API.test_Asynchronous() in ut002_serial_port_com.py

Test result: PASS

Test Identifier: TEST-T-223

Requirement ID(s): REQ-FUN-222, REQ-FUN-226 and REQ-AWM-222

Verification method: T

Test goal: Proper implementation of the synchronous sending and receving in the both blocking and timeout modes

Expected result: The following functionality is implemented:

- The synchronous sending is a blocking call, since it waits for the response to this particular sending
- All yet unclaimed responses to the previous (asynchronous) sendings are discarded
- The method can operate in:
  - Fully blocking mode - untill the response is received
  - Timeout blocking mode - untill the response is received OR timeout period is expired; the later case should result in a exception being raised
- The synchronous mode can be mixed with asynchronous only if the device connected to the port always sends responses (at least, confirmation of receipt) for each sending, but does not generate extra sendings on its own

Test steps: Perform the following test:

Blocking mode

- Instantiate the class using 'mock' port name and w/o *baudrate* keyword argument (9600 by default)

- Send two short but different string messages using asynchronous method *send*()
- Send a short but different thrid message using synchronous method *sendSync*(message, ReturnType = str)
- Compare the returned pair with the last sent message and number 3
- Check the response asynchronously (*getResponse*(ReturnType = str) method call) 100 more time - each time **None** value must be returned
- Delete the created instance
- Repeat the entire test using different *baudrate* values during instantiation: 50, 2400, 115200

Timeout mode

- Instantiate the class using 'mock' port name and *baudrate* = 115200
- Send a short string message using synchronous method *sendSync*(message, ReturnType = str, Timeout = 0.1) - 100 ms timeout
- Check that it returns the sent message and the package index 1
- Delete the created instance
- Instantiate the class using 'mock' port name and *baudrate* = 2400
- Send a short string message using synchronous method *sendSync*(message, ReturnType = str, Timeout = 0.1) - **serial.SerialTimeoutException** should be raised
- Re-open the connection
- Send a short string message using synchronous method *sendSync*(message, ReturnType = str, Timeout = 0) - explicitely blocking
- The send string should be returned and the package index should be 1
- Send another (different) string *sendSync*(message, ReturnType = str) - blocking by the default value
- The send string should be returned and the package index should be 2
- Send another (different) string *sendSync*(message, ReturnType = str, Timeout = 0.2) - 200 ms timeout
- The send string should be returned and the package index should be 3
- Delete the created instance

Implemented as methods Test_SimpleCOM_API.test_SynchronousBlocking() and Test_SimpleCOM_API.test_SynchronousTimeout() in ut002_serial_port_com.py

Test result: PASS

Test Identifier: TEST-T-224

Requirement ID(s): REQ-FUN-228

Verification method: T

Test goal: Check the support for the different input / output data types

Expected result: The following data types should be supported, i.e. accepted as the input for the sending and could be used to convert the binary response into:

- Bytestrings
- Byte arrays
- Unicode strings

- Instances of the classes having methods: *packToBytes*() - returns a bytestring representing the content of the object; *unpackFromBytes*() - class method / constructor returning a new instance of the class based on the content of a bytestring.

Note, COBS encoding / decoding should be applied to the bytestrings before sending / after receipt, so possible inclusion of b'\x00' characters will not interfere with the packages delimiters.

Test steps: Perform the following test:

- Instantiate the class using 'mock' port name and *baudrate* = 115200
- Create a string containing a character with the ASCII / Unicode code 0 in the middle
- Send this string using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*(str)
- Check that the same string is returned and the package index is 1
- Send the same string using *sendSync*(message, ReturnType = str), i.e. blocking call
- It should return the same string and the package index 2
- Convert the string into a bytestring using UTF-8 codec
- Send this bytestring using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*() - bytes as the return type by default
- Check that the same bytestring is returned and the package index is 3
- Send the same bytestring using *sendSync*(message), i.e. blocking call, bytes as the return type by default
- It should return the same bytestring and the package index 4
- Send this bytestring using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*(bytes)
- Check that the same bytestring is returned and the package index is 5
- Send the same bytestring using *sendSync*(message, ReturnType = bytes), i.e. blocking call
- It should return the same bytestring and the package index 6
- Convert the used bytestring into bytes array
- Send this bytes array using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*(bytearray)
- Check that the same bytes array is returned and the package index is 7
- Send the same bytes array using *sendSync*(message, ReturnType = bytearray), i.e. blocking call
- It should return the same bytes array and the package index 8
- Import class **SerNULL** from the module *com_lib.serialization*
- Instantiate it
- Send this instance using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*(SerNULL)
- Check that
    - the package index is 9
    - an instance of **SerNULL** class is returned
    - its method *getNative*() returns **None** value
- Send the same (initial) instance using *sendSync*(message, ReturnType = SerNULL), i.e. blocking call
- Check that
    - the package index is 10
    - an instance of **SerNULL** class is returned
    - its method *getNative*() returns **None** value

- Import class **CompexStruct** from the module *com_lib.tests.ut003_serialization*
- Instantiate it with the dictionary {'a' : 1, 'b' : 1.0, 'c' : {'a' : 2, 'b' : 1.0, 'c' : [3, 4]}}
- Send this instance using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*(CompexStruct)
- Check that
    - the package index is 11
    - an instance of **CompexStruct** class is returned
    - its method *getNative*() returns {'a' : 1, 'b' : 1.0, 'c' : {'a' : 2, 'b' : 1.0, 'c' : [3, 4]}}
- Send the same (initial) instance using *sendSync*(message, ReturnType = CompexStruct), i.e. blocking call
- Check that
    - the package index is 12
    - an instance of **CompexStruct** class is returned
    - its method *getNative*() returns {'a' : 1, 'b' : 1.0, 'c' : {'a' : 2, 'b' : 1.0, 'c' : [3, 4]}}
- Import class **NestedArray** from the module *com_lib.tests.ut003_serialization*
- Instantiate it with the list [{'a' : 1, 'b' : 1.0}, {'a' : 2, 'b' : 1.0}]
- Send this instance using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*(NestedArray)
- Check that
    - the package index is 13
    - an instance of **NestedArray** class is returned
    - its method *getNative*() returns [{'a' : 1, 'b' : 1.0}, {'a' : 2, 'b' : 1.0}]
- Send the same (initial) instance using *sendSync*(message, ReturnType = NestedArray), i.e. blocking call
- Check that
    - the package index is 14
    - an instance of **NestedArray** class is returned
    - its method *getNative*() returns [{'a' : 1, 'b' : 1.0}, {'a' : 2, 'b' : 1.0}]
- Import class **DynamicArrayArray** from the module *com_lib.tests.ut003_serialization*
- Instantiate it with the list [[1, 2], [3, 4], [5, 6]]
- Send this instance using *send*() method (asynchronous)
- Wait for 0.5 s and get the response calling *getResponse*(DynamicArrayArray)
- Check that
    - the package index is 15
    - an instance of **DynamicArrayArray** class is returned
    - its method *getNative*() returns [[1, 2], [3, 4], [5, 6]]
- Send the same (initial) instance using *sendSync*(message, ReturnType = DynamicArrayArray), i.e. blocking call
- Check that
    - the package index is 16
    - an instance of **DynamicArrayArray** class is returned
    - its method *getNative*() returns [[1, 2], [3, 4], [5, 6]]
- Delete the created instance of communication object

Test result: PASS

Test Identifier: TEST-T-225

Requirement ID(s): REQ-AWM-220

Verification method: T

Test goal: Check the suppression / not raising the unnecessary exceptions

Expected result: No exceptions are raised when:

- An already opened connection is requested to be open again - just ignored
- An already closed connection is requested to be closed - just ignored
- An attempt is made to send into or read from a closed, but properly configured and available port - the connection is re-opened automatically

Test steps: Perform the following test:

- Instantiate the class using 'mock' port name and *baudrate* = 115200
- Check that the connection is open, i.e. property *IsOpen*
- Send a short string message in the asynchronous mode
- Call method *open*() - no exception is raised
- Check that the connection is stil open, i.e. property *IsOpen*
- Wait for 0.5 sec and check the response *getResponse*(str) - it must return the previously sent string
- Close the connection by method *close*(), check its status, i.e. property *IsOpen*
- Call *close*() again - no exception should be raised, and the status should be closed
- Send a short string message in the asynchronous mode - no exception is raised
- Check the connection is open, close it and check the status
- Wait for 0.5 sec, than check the response - must return **None** but no exception is raised, and the connection is opened
- Close the connection, check its status
- Send a short message in the blocking synchronous mode - it should be returned, no exception is raised, and the connection is opened
- Delete the created instance

Test result: PASS

Test Identifier: TEST-T-226

Requirement ID(s): REQ-AWM-221

Verification method: T

Test goal: The **serial.SerialException** or its sub-class instance is raised each time when a port cannot be opened

Expected result: The said exception or an instance of its sub-class is raised when:

- The port cannot be openned directly during instantiation
- The port cannot be explicitely (method *open*()) or excplicitely (sending / receiving methods) re-openned

Test steps: Perform the following test:

- Instantiate the class using 'unmock' port name - this emulates connection to a port with non-existing path or not connected device - the expected exception should be raised
- **NB** - the following steps require knowledge of the internal implementation of the tested class - must be checked carefully upon refactoring
    - Intantitate the class anew using proper port name 'mock'
    - Immediately close the connection
    - Change the values of the 'private' attributes: *_Settings*['port'] = 'unmock' and *_Connection* = None
- Call the method *open*() - emulation of an attempt to reconnect to the disconnected device - exception should be raised
- Call the method *send*('test') - emulation of an attempt to reconnect to the disconnected device - exception should be raised
- Call the method *getResponse*() - emulation of an attempt to reconnect to the disconnected device - exception should be raised
- Call the method *sendSync*('test') - emulation of an attempt to reconnect to the disconnected device - exception should be raised

Test result: PASS

Test Identifier: TEST-T-227

Requirement ID(s): REQ-AWM-223

Verification method: T

Test goal: Check that the **TypeError** or its sub-class is raised or propagated in the case of improper input data type

Expected result: The expected exception is raised when:

- Any data type except string is passed as the positional mandatory port name into the initialization of the class being tested
- Any data type except integer is passed as the keyword argument *baudrate* into the initialization of the class being tested
- Any data type except string, bytestring, bytes array or instance of self-packing class is passed as the mandatory positional attribute of the sending methods
- Any data type (as type, not object) except string, bytestring, bytes array or self-unpacking class is passed as the optional / keyword argument *ReturnType* into *getResponse*() or *sendSync*() methods

Test steps: Perform the following test:

- Try to instantiate the class passing different non-string values as the port name - sub-class of **TypeError** should be raised each time
- Try to instantiate the class with the proper 'mock' value as the port name, but passing different non-integer values for the *baudrate* keyword argument - sub-class of **TypeError** should be raised each time
- Instantiate the class with the 'mock' port name

- Try to pass different improper values (e.g. integers, floating point, etc.) as the attribute of *send*() method - sub-class of **TypeError** should be raised each time
- Try to pass different improper values (e.g. integers, floating point, etc.) as the attribute of *sendSync*() method - sub-class of **TypeError** should be raised each time
- Send 'a' string. Wait for (0.5) sec. Try to pass different improper data types (e.g. **int**, **float**, etc.) as the return type keyword argument of *getResponse*() method - sub-class of **TypeError** should be raised each time
- Try to pass different improper data types (e.g. **int**, **float**, etc.) as the return type keyword argument of *sendSync*() method with the 'test' value as the message - sub-class of **TypeError** should be raised each time

Test result: PASS

Test Identifier: TEST-T-228

Requirement ID(s): REQ-AWM-224

Verification method: T

Test goal: Check that the **ValueError** or its sub-class is raised or propagated in the case of the proper input data type but unacceptable value

Expected result: The expected exception is raised when a connection setting passed into the initialization method of the class is of the proper data type, but of the unacceptable value.

Test steps: Try to instantiate the class being tested passing the keyword argument *baudrate* = 25 (not recognized value by the mock serial object). Check that a sub-class of **ValueError* is raised.

Test result: PASS

Traceability

For traceability the relation between tests and requirements is summarized in the table below:

Requirement ID	Covered in test(s)	Verified [YES/NO])
REQ-FUN-200	TEST-A-200	YES
REQ-FUN-201	TEST-A-201	YES
REQ-FUN-210	TEST-T-210	YES
REQ-FUN-220	TEST-T-220	YES
REQ-FUN-221	TEST-T-221	YES
REQ-FUN-222	TEST-T-221, TEST-T-223	YES
REQ-FUN-223	TEST-T-221	YES
REQ-FUN-224	TEST-T-222	YES
REQ-FUN-225	TEST-T-222	YES

Requirement ID	Covered in test(s)	Verified [YES/NO])
REQ-FUN-226	TEST-T-223	YES
REQ-FUN-227	TEST-T-220	YES
REQ-FUN-228	TEST-T-224	YES
REQ-AWM-220	TEST-T-225	YES
REQ-AWM-221	TEST-T-226	YES
REQ-AWM-222	TEST-T-223	YES
REQ-AWM-223	TEST-T-227	YES
REQ-AWM-224	TEST-T-228	YES

Software ready for production [YES/NO]	Rationale
YES	All tests passed