Test Report on the Module libhps.tools.mock_serial

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 preparation

Implement the defined tests as unit-test cases within the module ut001_mock_serial.

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-100

Requirement ID(s): REQ-FUN-100

Verification method: A

Test goal: The module implements a mock serial port interface with the minimal required API compatibility with the **PySerial** library class **Serial**, i.e. methods to open and close connection, send and receive data, and the ability to change the read and write timeout values as well as the communication baudrate. It also emulates a connection to a 'dummy' device, which simply echoes the recieved 'command'.

Expected result: All defined unit-test cases are passed, i.e. the tests TEST-T-110, TEST-T-120, TEST-T-121, TEST-T-122, TEST-T-123, TEST-T-124, TEST-T-125, TEST-T-126, TEST-T-127, TEST-T-128, TEST-T-129, TEST-T-12A and TEST-T-12B

Test steps: Execute the unit-test suit module **ut001_mock_serial.py**.

Test result: PASS

Test definitions (Test)

Test Identifier: TEST-T-100

Requirement ID(s): REQ-FUN-110

Verification method: T

Test goal: Check the implementation of a 'dummy' mock device connected to a 'mock' serial port (emulation).

Expected result: The following functionality is implemented:

- The emulation can be executed in a separate thread using two queues as the input and output buffers as well as a signaling 'stop' event, which are shared with the main thread
- The execution of the emulation does not start or terminates immediately if the 'stop' signal is set during its start-up
- The execution thread terminates if the 'stop' event is set by the listener / client process (from another thread)
- The execution thread terminates when the command 'quit' is received, which also causes the setting of the 'stop' event internally within the 'device' thread
- The 'device' emulation treats the input and output as byte streams with b'\x00' terminators between packages / 'commands'; e.g. the (Unicode) strings must be encoded into and decoded from the bytestring before sending and after receipt by the client process.

Test steps: Implement as unit-test cases and execute the following checks:

- Define a set / list of standard baudarates as positive integers from 50 to 115200
- Define a set of strings to be used as the input, including only ASCII-printable symbols (latin letters and arabic numerals) of different length as well as a mixture of ASCII printable and generic Unicode characters, e.g. cyrilics characters

- Make sure that the two buffer queues are properly emptied and the 'stop' event is unset (cleared) before and after of each of the test cases below
- Test in the main thread (not threaded!). For each pair of the baudrate and test input:
    - Calculate the *ExpectedTime* for transmission of a single byte at the chosen baudrate
    - Encode the test input using UTF-8 and append b'\x00' to the end. Place that data into the 'input' buffer one byter per queue element
    - Set the 'stop' event
    - Start timer and call the device emulation function from the main thread passing the filled 'input' queue, empty 'output' queue, 'stop' event object and the selected baudrate as the arguments
    - Wait for the made call to return the control and stop the timer
    - Check that:
        - The value of the timer doesn't exceed 2 * *ExpectedTime*
        - The content of the both buffer queues is not modified
        - The 'stop' event is still set
- Test in the main thread (not threaded!). For each pair of the baudrate and test input:
    - Encode the test input using UTF-8 and append b'\x00quit\x00' to the end. Place that data into the 'input' buffer one byter per queue element
    - Calculate the *ExpectedTime* for transmission of (2 * N + 7) bytes at the chosen baudrate, where N is the length of UTF-8 encoded test input
    - Start timer and call the device emulation function from the main thread passing the filled 'input' queue, empty 'output' queue, 'stop' event object and the selected baudrate as the arguments
    - Wait for the made call to return the control and stop the timer
    - Pull all bytes accumulated in the 'output' buffer queue, remove the last one and encode the rest into a Unicode string using UTF-8 codec
    - Check that:
        - The value of the timer is within the range *ExpectedTime* to (1.1 * *ExpectedTime* + 0.01) due to the call overhead
        - The 'input' buffer is empty
        - The 'stop' event is set
        - The returned string (reconstructed from the 'output' buffer) equals to the intial test string
    - Unset (clear) the 'stop' event
- Test using a separate thread. For each baudrate:
    - Start the device emulator in a separate thread and pass the empty 'input' and 'output' queues, the unset 'stop' event object and the selected baudrate as the arguments
    - For each test string:
        - Encode the test input using UTF-8 and append b'\x00' to the end.
        - Calculate the *ExpectedTime* for transmission of 2 * (N + 1) bytes at the chosen baudrate, where N is the length of UTF-8 encoded test input
        - Start the timer
        - Place the encoded data into the 'input' buffer one byter per queue element, including the b'\x00' terminator
        - Immediately start pulling the 'output' buffer

        - Stop the timer as soon as the b'\x00' is pulled, encode all previously pulled data into a Unicode string using UTF-8 codec
        - Check that:
            - The value of the timer is within the range 0.75 * *ExpectedTime* to (1.1 * *ExpectedTime* + 0.02 * (N+1)) due to the call overhead and threads switching
            - The 'input' buffer is empty
            - The returned string (reconstructed from the 'output' buffer) equals to the intial test string
    - Place per-byte b'quit\x00' into the 'input' buffer
    - Wait for the 'device' thread to terminate and join the main thead
    - Check that the both queues are empty and the 'stop' signal is set
    - Clear the 'stop' signal
- Test using a separate thread. For each baudrate:
    - Start the device emulator in a separate thread and pass the empty 'input' and 'output' queues, the unset 'stop' event object and the selected baudrate as the arguments
    - Place a single integer in the range 1 to 127 inclusively into the 'input' buffer
    - Wait for the time required to send a single character at the selected baudrate
    - Set the 'stop' signal
    - Wait for the 'device' thread to terminate and join the main thead
    - Check that the both queues are empty and the 'stop' signal is set
    - Clear the 'stop' signal

This test is implemented as com_lib.tests.ut001_mock_serial.Test_MockDevice class.

Test result: PASS

Test Identifier: TEST-T-120

Requirement ID(s): REQ-FUN-120

Verification method: T

Test goal: The implementation of the mock serial port connection provides the minimum required API, compatible with the **serial.Serial** class.

Expected result: The following attributes are available on an instance of the class being tested: *open*, *close*, *read*, *write*, *is_open*, *port*, *baudrate*, *in_waiting*, *out_waiting*, *timeout*, *write_timeout*.

Test steps: Implement and execute the following test case:

- Instantiate the class being tested without arguments, except for port = 'mock'
- Check that it has all required attributes
- Delete the created instance

This test is implemented as the method *test_HasAttributes* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-121

Requirement ID(s): REQ-FUN-121

Verification method: T

Test goal: The mock serial class can be instantiated without arguments (default settings are applied) or with any amount of any keyword arguments, from which only the minimum required ones: *port*, *timeout*, *write_timeout* and *baudrate* - are recognized and applied, as long as their values are of the proper data type and values range.

Expected result: When instantiated without arguments the following default values are applied (as retrived using the respective getter):

- port = None
- baudrate = 9600
- timeout = None
- write_timeout = None

When instantiated with the keyword arguments, the respective values are assigned.

Test steps: Implement and execute the following test case:

- Instantiate the class being tested without arguments
- Check the values of the attributes (properties) *port*, *timeout*, *write_timeout* and *baudrate*
- Delete the created instance
- Instantiate the class being tested with the following keyword arguments: port = 'mock', timeout = 0.5, write_timeout = 0, baudrate = 115200
- Check the values of the attributes (properties) *port*, *timeout*, *write_timeout* and *baudrate*
- Check that the connection is opened
- Delete the created instance

This test is implemented as the method *test_initOk* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-122

Requirement ID(s): REQ-AWM-120

Verification method: T

Test goal: Check that **TypeError**-type exception is raised when a connection setting gets an improper data type value

Expected result: The exception is raised when the respective keyword's value or the value assigned to the respective property is:

- *port* - any type except **None** and **str**
- *baudrate* - any type except **int**

- *timeout* and *write_timeout* - any type except **None**, **int** or **float**

Test steps: Implement and execute the following test case:

- Try to instantiate the class passing different unacceptable data types for the keyword argument *port* - check that a sub-class of **TypeError** is raised
- Try to instantiate the class passing different unacceptable data types for the keyword argument *baudrate* - check that a sub-class of **TypeError** is raised
- Try to instantiate the class passing different unacceptable data types for the keyword argument *timeout* - check that a sub-class of **TypeError** is raised
- Try to instantiate the class passing different unacceptable data types for the keyword argument *write_timeout* - check that a sub-class of **TypeError** is raised
- Instantiate the class being tested without arguments
- Try to assign different unacceptable data types to the property *port* - check that a sub-class of **TypeError** is raised
- Try to assign different unacceptable data types to the property *baudrate* - check that a sub-class of **TypeError** is raised
- Try to assign different unacceptable data types to the property *timeout* - check that a sub-class of **TypeError** is raised
- Try to assign different unacceptable data types to the property *write_timeout* - check that a sub-class of **TypeError** is raised
- Delete the created instance

This test is implemented as the methods *test_init_TypeError*, *test_port_TypeError*, *test_baudrate_TypeError*, *test_timeout_TypeError* and *test_write_timeout_TypeError* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-123

Requirement ID(s): REQ-AWM-121

Verification method: T

Test goal: Check that **ValueError**-type exception is raised when a connection setting gets an improper data type value

Expected result: The exception is raised when the respective keyword's value or the value assigned to the respective property is:

- *port* - any string except 'mock'
- *baudrate* - any **int** except 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 - i.e. the standard values recognized by **serial.Serial**
- *timeout* and *write_timeout* - any negative **int** or **float**

Test steps: Implement and execute the following test case:

- Try to instantiate the class passing any string but 'mock' for the keyword argument *port* - check that a sub-class of **serial.SerialException** is raised

- Try to instantiate the class passing any other interger (except for the recognized values) for the keyword argument *baudrate* - check that a sub-class of **ValueError** is raised
- Try to instantiate the class passing a negative float and / or integer for the keyword argument *timeout* - check that a sub-class of **ValueError** is raised
- Try to instantiate the class passing a negative float and / or integer for the keyword argument *write_timeout* - check that a sub-class of **ValueError** is raised
- Instantiate the class being tested without arguments
- Try to assign any string but 'mock' to the property *port* - check that a sub-class of **serial.SerialException** is raised
- Try to assign any other interger (except for the recognized values) to the property *baudrate* - check that a sub-class of **ValueError** is raised
- Try to assign a negative float and / or integer to the property *timeout* - check that a sub-class of **ValueError** is raised
- Try to assign a negative float and / or integer to the property *write_timeout* - check that a sub-class of **ValueError** is raised
- Delete the created instance

This test is implemented as the method *test_init_ValueError*, *test_port_serial.SerialException*, *test_baudrate_ValueError*, *test_timeout_ValueError* and *test_write_timeout_ValueError* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-124

Requirement ID(s): REQ-AWM-122

Verification method: T

Test goal: Check that **serial.SerialException** is raised upon opening of a connection if the port is not yet assigned

Expected result: The said exception is raised if *port* = None when the method *open*() is called

Test steps: Implement and execute the following test case:

- Instantiate the class being tested without arguments - default None is used for port
- Check that the connection is not opened
- Try to call the method *open*() - check that the required exception is raised
- Check that the connection is still closed
- Destroy the created instance

This test is implemented as the method *test_SerialException_No_Port* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-125

Requirement ID(s): REQ-AWM-123

Verification method: T

Test goal: Check that **serial.SerialException** is raised upon re-opening of the already active connection

Expected result: The said exception is raised if the method *open*() is called with the already active connection

Test steps: Implement and execute the following test case:

- Instantiate the class being tested witout port = 'mock' keyword argument
- Check that the connection is active and port == 'mock'
- Try to call the method *open*() - check that the required exception is raised
- Check that the connection is now closed but port == 'mock'
- Destroy the created instance

This test is implemented as the method *test_SerialException_AlreadyOpen* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-126

Requirement ID(s): REQ-AWM-124

Verification method: T

Test goal: Check that **serial.SerialException** is raised upon closing of the already inactive connection

Expected result: The said exception is raised if the method *close*() is called with the already inactive connection

Test steps: Implement and execute the following test case:

- Instantiate the class being tested without port keyword argument
- Check that the connection is not active and port == None
- Try to call the method *close*() - check that the required exception is raised
- Check that the connection is still closed but port == None
- Destroy the created instance

This test is implemented as the method *test_SerialException_AlreadyClosed* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-127

Requirement ID(s): REQ-AWM-125

Verification method: T

Test goal: Check that **serial.SerialException** is raised during attempted reading from, writing to or quering status of the buffers if the connection is not opened yet.

Expected result: The said exception is raised if the following methods / properties are called with the inactive connection:

- *read*()
- *write*()
- *in_waiting*
- *out_waiting*

Test steps: Implement and execute the following test case:

- Instantiate the class being tested without port keyword argument
- Check that the connection is not active and port == None
- Try to call the listed methods and properties - check that the required exception is raised
- Check that the connection is still closed but port == None after each call
- Destroy the created instance

This test is implemented as the method *test_SerialException_Inactive* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-128

Requirement ID(s): REQ-AWM-126

Verification method: T

Test goal: Check that **TypeError** is raised by *read*() method if the argument is not an integer, and by *write*() method if the argument is not a bytestring

Expected result: The said exception is raised each time by the mentioned methods when an acceptable data type is passed as the argument

Test steps: Implement and execute the following test case:

- Instantiate the class being tested with port = 'mock'
- For each data type inappropriate for the *read*() method:
    - Check that the connection is active
    - Try to call the *read*() method with an inappropriate argument - check that the required exception is raised
    - Check that the connection is closed but port == 'mock'
    - Re-open the connection - call *open*()
- Perform similar checks (except the data types) on the *write*() method
- Destroy the created instance

This test is implemented as the methods *test_read_TypeError* and *test_write_TypeError* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-129

Requirement ID(s): REQ-AWM-127

Verification method: T

Test goal: Check that **ValueError** is raised by *read*() method if the argument is an integer but not positive

Expected result: The said exception is raised each time by the mentioned method when zero or negative integer is passed as the argument

Test steps: Implement and execute the following test case:

- Instantiate the class being tested with port = 'mock'
- For zero and several negative integers do the following:
  - Check that the connection is active
  - Try to call the *read*() method with an inappropriate argument - check that the required exception is raised
  - Check that the connection is closed but port == 'mock'
  - Re-open the connection - call *open*()
- Destroy the created instance

This test is implemented as the method *test_read_ValueError* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-12A

Requirement ID(s): REQ-FUN-122, REQ-FUN-123, REQ-FUN-124, REQ-FUN-125 and REQ-AWM-128

Verification method: T

Test goal: Check the implemented functionality of the class, i.e. that it emulates the standard behaviour of the **serial.Serial** class

Expected result: The following functionality is shown:

- Getter / setter properties *port*, *baudrate*, *timeout* and *write_timeout* not only change the values of the internal atributes, which are properly returned, but also affects the work flow of the methods *read*() and *write*() as defined by the specification of the **PySerial** library
- Getter only properties *is_open*, *in_waiting* and *out_waiting* properly report the status of the connection and the incoming and outgoing buffers respectively
- Work flow of the *read*() and *write*() methods follows the specification of the **PySerial** library

Test steps: Implement and execute the following test cases:

Blocking sending and receiving

- Instantiate the class being tested without keyword arguments
- Check that the connection is not open, *port* is None, *baudrate* is 9600 and both *timeout* and *write_timeout* are None
- Set properties *baudrate* = 2400 (about 300 bytes per second), *port* = 'mock' - check these properties for the values, and that the connection is open
- Check that both buffers are empty
- Prepare a test bytestring to be sent - b'test_case\x00' (10 characters), i.e. ~ 1/30 sec (~ 30 ms) to be sent or received
- Start timer and call *write*() method with this bytestring as the argument
- Clock the first time interval *t1* and get the values of the properties *out_waiting* as *out* and *in_waiting* as *in1*
- Call the *read*(10) method (10 characters to read) -> *result*
- Clock the second time interval *t2* (since *t1* moment), get the value of the property *in_waiting* as *in2*
- Check that:
  - *t1* is between 30 and 100 ms
  - *t2* is between 30 and 100 ms
  - *out* is zero
  - *in1* is >= 0 but < 10
  - *in2* is 0
  - *result* is b'test_case\x00'
- Call *write*() method this the same bytestring argument again
- Start timer
- Do 10 times in a loop - call *read*() w/o argument (1 byte by default) and accumulate the bytes in *result* bytestring
- Stop the timer, check that it is between 30 and 100 ms
- Check that:
  - *result* is b'test_case\x00'
  - *in_waiting* is 0
  - connection is open and *port* is 'mock'
- Assign *port* = 'mock', check that connection is open and *port* is 'mock'
- Assign *port* = 'mock2', check that connection is open and *port* is 'mock2'
- Close the connection and test that *is_open* is False
- Re-open the connection (method *open*()) and check that connection is open and *port* is 'mock2'
- Assign *port* = None, check that connection is closed and *port* is None
- Delete the created instance

Non-blocking sending and receiving

- Instantiate the class being tested with the keyword arguments *port* = 'mock', *baudrate* = 2400, *timeout* = 0 and *write_timeout* = 0
- Check that the connection is open, *port* is 'mock', *baudrate* is 2400 and both *timeout* and *write_timeout* are 0
- Check that both buffers are empty
- Prepare a test bytestring to be sent - b'test_case\x00' (10 characters), i.e. ~ 1/30 sec (~ 30 ms) to be sent or received
- Start timer and call *write*() method with this bytestring as the argument
- Clock the first time interval *t1*

- Wait for 10 ms and get the values of the properties *out_waiting* as *out* and *in_waiting* as *in1*
- Loop idling until *out_waiting* is 0, then wait for 10 ms
- Get the value of *in_waiting* as *in2*
- Start the timer and call *read*(10) -> *result*, stop the timer and get the second time interval *t2*
- Wait for ~ 100 ms and get *in_waiting* as *in3*
- Check that:
    - 0 < len(*result*) < 10 and b'test_case\x00'.startswith(*result*)
    - 0 < *out* < 10
    - *in1* = 0
    - 0 < *in2* < 10
    - 0 < *in3* < 10
    - both timinings are below 10 ms
- Try to read 20 bytes; the returned string should be shorter and represent the remainder of the previously sent test bytestring
- Repeat sending and loop idling until *in_waiting* = 10, when call *read*(20) -> *result*
- Check that *result* = b'test_case\x00'
- Close the connection and delete the created instance

Timed sending and receiving

- Instantiate the class being tested with the keyword arguments *port* = 'mock', *baudrate* = 2400, *timeout* = 0.01 and *write_timeout* = None
- Check that the connection is open, *port* is 'mock', *baudrate* is 2400, *timeout* is 0.01 and *write_timeout* is None
- Check that both buffers are empty
- Prepare a test bytestring to be sent - b'test_case\x00' (10 characters), i.e. ~ 1/30 sec (~ 30 ms) to be sent or received
- Send the string (blocking)
- Start the timer, call *read*(10) -> *result*, stop the timer as *t1*
- Start the timer again, call *read*(), stop the timer as *t2*
- Check that:
    - 0 < len(*result*) < 10 and b'test_case\x00'.startswith(*result*)
    - the first timing is between 10 and 20 ms
    - the second call returns exactly 1 byte
    - the second timming is below 10 ms
- Set *write_timeout* = 0.01, check that it is set
- Check that the connection is open
- Try to send the same bytestring (*write*() in the time-out mode) - check that **serial.SerialTimeoutException** is raised
- Check that the connection is closed
- Delete the created instance

This test is implemented as the methods *test_Blocking*, *test_NonBlocking* and *test_Timed* of com_lib.tests.ut001_mock_serial.Test_MockSerial class.

Test result: PASS

Test Identifier: TEST-T-12B

Requirement ID(s): REQ-FUN-126

Verification method: T

Test goal: The closed port can be re-opened

Expected result: The previously opened but now closed port can be re-opened and operated, which can be the consequence of:

- User called *close*() method
- **TypeError** or **ValueError** is raised due to wrong input
- **serial.SerialTimeoutException** is raised due to reached write timeout

Test steps:

- Instantiate the class being tested with the keyword arguments *port* = 'mock', *baudrate* = 2400 and *write_timeout* = 0.01
- Check that the connection is open, *port* is 'mock', *baudrate* is 2400 and *write_timeout* is 0.01
- Check that both buffers are empty
- Prepare a test bytestring to be sent - b'test_case\x00' (10 characters), i.e. ~ 1/30 sec (~ 30 ms) to be sent or received
- Try to send the test bytestring (*write*() in the time-out mode) - check that **serial.SerialTimeoutException** is raised
- Check that the connection is closed
- Re-open the connection, check the connection is open, *port* is 'mock', *baudrate* is 2400 and *write_timeout* is 0.01, and both buffers are empty
- Set *write_timeout* to None and check the result
- Send the data in the blockng mode, wait for 1 sec and read 10 bytes from the port; check that the bytestring is received
- Close the connection, method *close*()
- Re-open the connection, check the connection is open, *port* is 'mock', *baudrate* is 2400 and *write_timeout* is None, and both buffers are empty
- Send the data in the blockng mode, wait for 1 sec and read 10 bytes from the port; check that the bytestring is received
- Try to set *write_timeout* to a negative value - **ValueError** should be raised; check that the connection is closed
- Re-open the connection, check the connection is open, *port* is 'mock', *baudrate* is 2400 and *write_timeout* is None, and both buffers are empty
- Send the data in the blockng mode, wait for 1 sec and read 10 bytes from the port; check that the bytestring is received
- Try to set *write_timeout* to a string value - **TypeError** should be raised; check that the connection is closed
- Re-open the connection, check the connection is open, *port* is 'mock', *baudrate* is 2400 and *write_timeout* is None, and both buffers are empty
- Send the data in the blockng mode, wait for 1 sec and read 10 bytes from the port; check that the bytestring is received
- Delete the created instance

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-100	TEST-A-100	YES
REQ-FUN-110	TEST-T-110	YES
REQ-FUN-120	TEST-T-120	YES
REQ-FUN-121	TEST-T-121	YES
REQ-FUN-122	TEST-T-12A	YES
REQ-FUN-123	TEST-T-12A	YES
REQ-FUN-124	TEST-T-12A	YES
REQ-FUN-125	TEST-T-12A	YES
REQ-FUN-126	TEST-T-12B	YES
REQ-AWM-120	TEST-T-122	YES
REQ-AWM-121	TEST-T-123	YES
REQ-AWM-122	TEST-T-124	YES
REQ-AWM-123	TEST-T-125	YES
REQ-AWM-124	TEST-T-126	YES
REQ-AWM-125	TEST-T-127	YES
REQ-AWM-126	TEST-T-128	YES
REQ-AWM-127	TEST-T-129	YES
REQ-AWM-128	TEST-T-12A	YES

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