Multi Bus Tester - Users Guide

Release 0.2.3

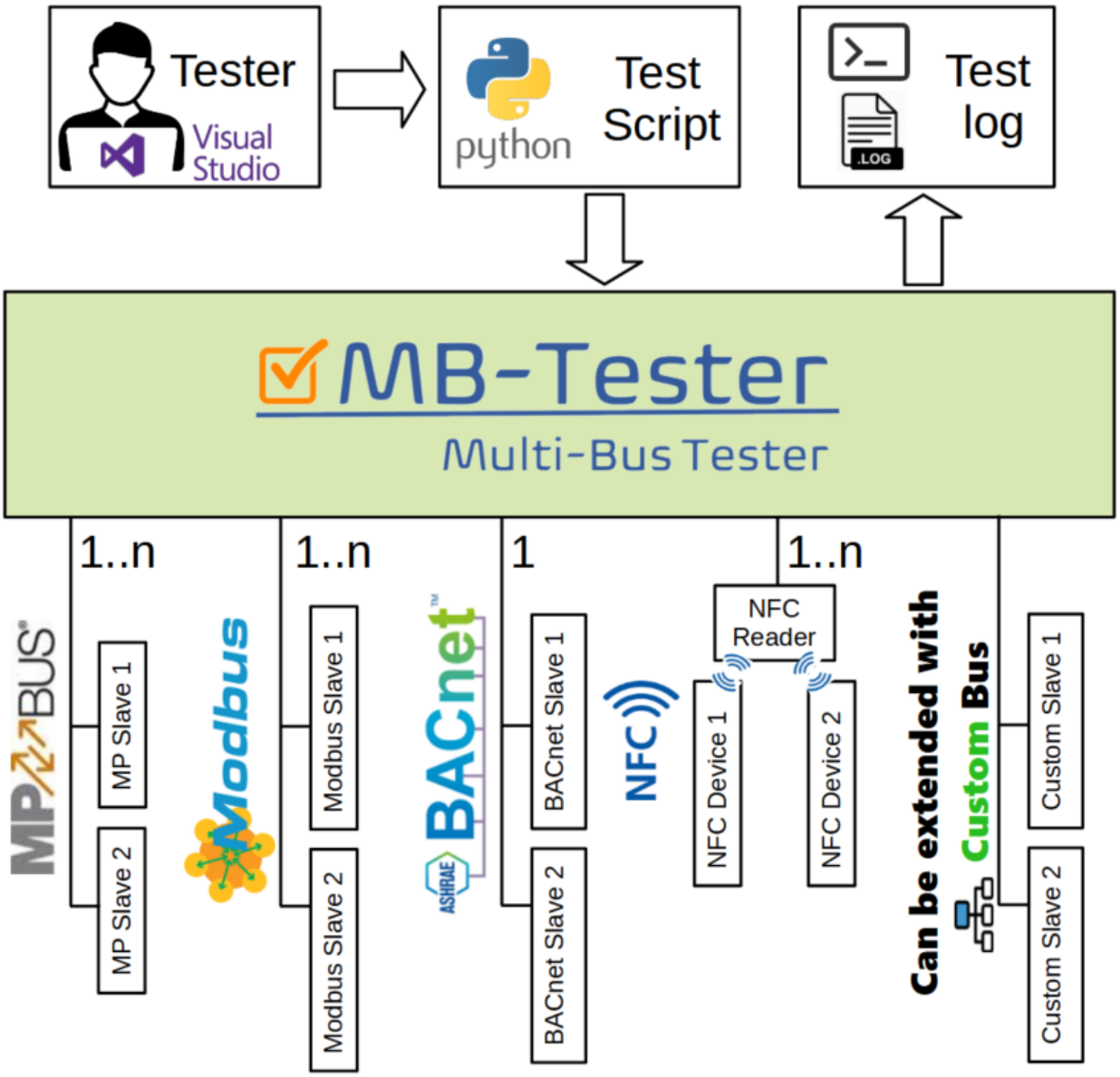
2022-12-21

CONTENTS:

CHAPTER

ONE

OVERVIEW



Multi-Bus Tester, abbreviated as MB-Tester, enables the *tester* of a device to

- easily test the functionality of a device or several devices

- create easily repeatable tests
- implement test cases for the purpose of regression testing
- test in a fully automated or partly automated (with user interaction) manner
- create longterm reliability and stress tests
- communicate over several different bus systems at the same time
- log the execution of the tests, which can be used for analysis later
- integrate device tests into a pytest environment
- easily integrate in most existing test environments
- develop and debug tests in Visual Studio or any other Python IDE

To test the devices, *test cases* are developed as Python scripts. Python is widely used for testing already and it is a very well supported language with many libraries already available for different purposes.

A *test case* in the MB-Tester context is a python script that executes actions to stimulate the device(s) under test and verify the responses. This can happen fully automated or with user interaction. It can be a standalone script or a pytest test case.

MB-Tester supports the following features to give an easy-to-use and powerful testing framework for device testing

- multiple communication protocols supported out of the box
- automatic logging of communication errors
- support for easy error and warning counting
- suppression of expected communication errors

1.1 Communication with Devices

MB-Tester makes it easy to communicate with your *devices under test* with built-in communication interfaces for several field bus protocols.

You can connect to multiple buses at the same time and you can use them in one test.

1.2 Test script editing, execution, debugging

Python test scripts can be edited, executed and debugged comfortably in Microsoft Visual Studio or any other IDE that supports Python. The test scripts can be also executed on the command line, allowing integration into any test environment.

1.3 Error and Warning Counters

There is an error and a warning counter integrated in the MB-Tester. The current number of errors and warnings is reported in the log on every line.

The error and warning counters can be incremented in two ways:

- A communication error in any of the field buses will increase the error counter. The tester can also define the errors that are *accepted* for each test step, which will not increase the counter.

- The tester can increment the counters in the python code based on user defined criteria.

1.4 Logging

Time-stamped logging data is shown on the console and written to a log file. The log level can be set with fine granularity to enable logging the exactly necessary information.

1.5 A Small Example

```
from mb_tester.MBTester     import MBTester, LogLevels;
from mb_tester.MPBus        import MPBus, CommunicationTypes;
from mb_tester.MPSerialPort import MPSerialPort,MPBaudRates;
from mb_tester.NFC          import NFC;
from mb_tester.NFCPort      import USBFeigReaderPort;

# define the log levels to be used
logLevel = [ LogLevels.ERROR
                  , LogLevels.WARNING
                  , LogLevels.INFO ];
# Initiate the MB Tester object
tester  = MBTester( aDefaultLogLevel = logLevel );

# Create an MP serial port for MP communication
mpserial = MPSerialPort( aTester            = tester
                       , aName              = "MPPort1"
                       , aDeviceId          = "COM6"
                       , aCommunicationType = CommunicationTypes.MP3
                       , aBaudRate          = MPBaudRates.B1200
                       , aRtsLevel          = True
                       , aDtrLevel          = True
                       , aIsDefault         = True );

# Create the MP Bus object that implements the MP stack to send
#  MP commands and receive the answers.
mpbus     = MPBus(tester, aDefaultPort = mpserial);

# Execute the MP Command. The call will return after the answer is
#   received. The Answer bytes are returned in the buffer.
(result, buffer) = mpbus.GetSeriesNo();

# Execute an MB Tester assertion
tester.IsTrue(result == 0);

# delete the created objects
del mpbus;
del mpserial;
del nfc_port;
del tester;
```

CHAPTER

TWO

GENERAL FEATURES

This chapter summarizes the general features that are valid for all modules of the MB-Tester.

2.1 Invoking a Script on the Command Line

MB-Tester provides an argument evaluator that will make it easy to get values from the command line. See the class ArgumentEvaluator for details. The following example initializes the MBTester using command line arguments.

```
tester = MBTester( aEvaluator = ArgumentEvaluator() );
```

The usage of the ArgumentEvaluator can be also omitted, in this case the script will ignore command line arguments.

2.2 Logging

The general format of the log messages are:

```
<YYYY.MM.DD h24.mm.ss.ms> [E:<Error Counter>,W:<Warning Counter>] <Log Level Prefix>[spces]<Log Message>
```

The following general rules apply to the log messages:

- The fatal error is never suppressible. If no output is configured (no file output and the console is in quiet mode), it will still be shown on the console.
- Each output line contains
  - the exact time of logging with millisecond resolution. Note that the accuracy may be restricted by the operating system.
  - the current error and warning counters.
  - log level information.
- The width of the log messages is controllable on the console as a possibility to improve performance. The minimal log width is 60 characters.
- The default log width is 400 chars.
- The default log level is possible to set inside of the script. The log level can be overridden from the command line, if the command line parameters are used.
- The first log entry of the script is always the version number of multi bus tester library. This text is not suppressible.

The field bus communication is logged according to these rules:

- when the function writes out data to a bus
  - log level : *LogLevels.OUTPUT*
  - message : The raw out data.
- when the function received data from a bus
  - log level : *LogLevels.INPUT*
  - message : The raw in data.
- when the function has received an error, which is accepted (accept is configured by a *aAcceptErrors*)
  - log level : *LogLevels.INFO*
  - message : "<label> : Ignored (<error number>) <error message>"
- when the function has received an error, which is not accepted (default)
  - log level : *LogLevels.ERROR*
  - message : "<label> : (<error numbe>) <error message>"

2.3 Ports and Buses

MB-Tester communicates on a field bus using a bus object. The bus object communicates with the communication device using a port object.

A port represents a connection to a communication device. It provides an interface that makes it possible to exchange data with the device.

- The port contains the following information:
  - name: unique string identifier for the port
  - device id: string identifier for a communication device (e.g.: COM1, COM2 . . . ., FEIG Reader id, Aardvark Adapter Id, etc.)
- All ports have to be freed at the end of the run by its destructor function if it is not deleted during the script run.
- Two ports can not use the same device identifier in the same test environment.

Device Id :

In the case of a bus based on serial communication, like MP-Bus or Modbus/RTU, this identifies a COM port, in the case of NFC it is the identification of the NFC reader.

Port name :

The name of the port, used to identify it from command line to override the device id.

Below is an *example* of initializing a port and then a bus with the previously initialized port as a default port.

```
# Create a serial port for the MP Bus to use
mpserial = MPSerialPort( aTester           = tester
                       , aName             = "MPPort1"
                       , aDeviceId         = "COM25"
                       , aCommunicationType = CommunicationTypes.MP3
                       , aBaudRate         = MPBaudRates.B1200);
# Create the MP Bus object that implements the MP stack to send
#  MP commands and receive the answers.
mpbus     = MPBus(tester, aDefaultPort = mpserial);
```

2.3.1 Specifying Port Devices on the Command Line

1. Define a port in the command line arguments:

```
python TestScript.py --dev-MPPort1 COM16
```

1. If you want to use this port in the script, you have to set the aName parameter of the initializer of the port to the same port name, and aDeviceId must be None.

```
port = MPSerialPort( aTester            = tester
                   , aName              = "MPPort1"
                   , aDeviceId          = "COM1"
                   , aCommunicationType = CommunicationTypes.MP3
                   , aIsDefault         = True );
```

You can override the command line argument by giving to the aDeviceId a value.

2.4 Communication Functions

Communication functions send a command to a field bus device, wait for the answer and return it if there is an answer.

If there is an error the error code is returned.

2.4.1 Common Function Parameters

The meaning of these parameters is the same for each communication function, where they are available.

- **aLabel** : This string is added at the beginning of the log message. If it is omitted or the value is 'None', the command name will be the label in the log message.
- **aPort** : The name of the selected port. If it is omitted or it is 'None', the default port will be used.
- **aAcceptErrors** : The list of the accepted errors during communication. If it is ommitted or 'None', no error will be accepted and the error counter will be incremented for every error. See the possible error codes in the ErrorCodes.pdf file.

2.4.2 Function Return Values

The execution of all communication functions returns a list with at least two elements. The first two return values have a fixed meaning:

- **first return value:** The error code if an error occured. If the result code is 0, no error occurred during execution. In case an accepted error occurred, also 0 is returned. If an error occurred during communication.
- **second return value:** The list of received bytes (List[int]). If an error occurred during communication or no data was received, the list is empty.

CHAPTER

THREE

USED DOCUMENTS

The MB-Tester is created using the following documentation. MBTester tries to adhere to the same naming convention.

Belimo MPBus:

- (Belimo internal) A91613-101 MP-Spec.pdf (05.11.2010.)
- (Belimo internal) A91621-003 MP-Befehle.pdf (05.05.2015.)
- (Belimo internal) MP-Bus Failure Codes.pdf
- **used library**: SWAP (11.2.0)

Belimo NFC Protocol (BNP):

- (Belimo internal) A91613-751-15_NFC_Interface_Description_Belimo_NFCApp.pdf (07.12.2020)

NFC:

- NXP NFC tag documentation: NT3H2111_2211.pdf (30.11.2017.)
- FEIG NFC reader API documentation
- **used library**: FEDM Core API (05.05.02)

Modbus:

- MODBUS APPLICATION PROTOCOL SPECIFICATION (V1.1b) (official Modbus standard)
- Modbus Protocol Reference Guide (EM-5650 Rev.10) (by M-System Co Ltd)
- **used library**: libmodbus 3.1.5 (2019-07-29)

CHAPTER

FOUR

MODULES

4.1 MBTester

4.1.1 Introduction

This module contains the basics to write a *test case* with the Multi-Bus Tester.

Its tasks are:

- Initialize the MBTester environment
- Provide a logging interface
- Provide a base implementation of the communication port for different protocols.
- Provide functions to validate the result of a test step and signal errors or warnings
- Write a summary of errors and warnings at the end of the test

4.1.1.1 Initialization

Below is an example of initializing the MBTester environment. The "tester" object will be used in the test case to invoke the different functions.

```
from mb_tester.MBTester import MBTester, ArgumentEvaluator;

# create a new MBTester instance
tester = MBTester( aEvaluator = ArgumentEvaluator() )
```

4.1.2 Types and Constants

Types used by the MBTester module.

4.1.2.1 LogLevels

Import : 'from mb_tester.MBTester import LogLevels'

```
class LogLevels(IntEnum):
    """description : Constants for available predefined log levels."""
    FATAL_ERROR      = 0;
    ERROR            = 1;
    WARNING          = 2;
    INFO             = 3;
    INPUT            = 4;
    OUTPUT           = 5;
    DEBUG            = 6;
    USER1            = 7;
    USER2            = 8;
    DEBUG_DLL_MPBUS  = 9;
    DEBUG_DLL_NFC    = 10;
    DEBUG_DLL_MODBUS = 11;
```

Log Level	Log Level Strings In the log	Details
LogLevels.FATAL_ERROR	"FATAL_ERR"	Critical error. Always visible, and not possible to suppress.
LogLevels.ERROR	"ERROR"	Non-critical error.
LogLevels.WARNING	"WARNING"	Warning.
LogLevels.INFO	"INFO"	Any information related to the command.
LogLevels.INPUT	"INPUT"	Received data from bus sides.
LogLevels.OUTPUT	"OUTPUT"	Sent data to bus sides.
LogLevels.DEBUG	"DEBUG"	Debugging information. It is used by SW.
LogLevels.USER1	"USER1"	First log level for the user. SW is not using it.
LogLevels.USER2	"USER2"	Second log level for the user. SW is not using it.
LogLevels.DEBUG_DLL_*	shortened form of the name.	These log levels are for internal debugging purposes.

Note: if the log level is defined on the command line, the full name of the enum value must be used, e.g. "DEBUG_DLL_MPBUS" and not the form that is displayed in the log.

4.1.2.2 ConvertTypes

These are types that are used by functions to read numbers from a byte list (ToNumber()) or write numbers to a byte list (ToBytes()). It is used for example to decode the answer bytes of a command.

Import : 'from mb_tester.MBTester import ConvertTypes'

```
class ConvertTypes(IntEnum):
    """Type of in or out value."""
    UINT8  : int = 0;
    UINT16 : int = 1;
    UINT32 : int = 2;
    UINT64 : int = 3;
    INT8   : int = 4;
    INT16  : int = 5;
    INT32  : int = 6;
    INT64  : int = 7;
    DOUBLE : int = 8;
    FLOAT  : int = 9;
    STRING : int = 10;
```

4.1.2.3 Endianness

It influences the endianness used by the conversion routines ToBytes() and ToNumber().

Import : 'from mb_tester.MBTester import Endianness'

```
class Endianness:
    """The constants of endianneses."""
    BIG_ENDIAN    : str = "big";
    LITTLE_ENDIAN : str = "little";
```

4.1.2.4 DEFAULT_LOG_LEVELS

Defines the default log levels used by the MB-Tester. The constant is of the *LogLevels* type.

Import : 'from mb_tester.MBTester import DEFAULT_LOG_LEVELS'

```
DEFAULT_LOG_LEVELS : TypeLogLevels = [
     LogLevels.ERROR
   , LogLevels.WARNING
   , LogLevels.INFO
   , LogLevels.INPUT
   , LogLevels.OUTPUT
];
```

4.1.3 Class ArgumentEvaluator

Evaluates the command-line parameters. Here is the list of arguments that are understood by the argument evaluator:

- **–log-level / -l** : Selected log levels separated by comma.
- **–log-file / -f** : Output file for logging with path.
- **–log-width** : Width of log text on the console. The log messages are truncated to the given width, thus improving the performance of the test execution. The log file contains the full log text.
- **–quiet / -q** : Disables the logging to standard output (console).
- **–dev-** : Specify which port device should be used for a given port identifier. A port device can be: COM1, COM2, , … etc. More than one port can be specified by using several –dev-X arguments. **For example** if the script uses the following ports:

  port1 = MPSerialPort(… aName="**Port1**", aDeviceId = "COM6" …) port2 = MPSerialPort(… aName="**Port2**", aDeviceId = "COM7" …)

  The used ports can be modified on the command line as follows: ….. –**dev-Port1** "COM27" –**dev-Port2** "COM32"
- **–help / -h** : Shows help and exits.
- **-v** : Shows the MBTester python lib version only and exits.
- **-V** : Show all versions (including MBTester and all used libraries) and exit.

The constructor of the *class MBTester* takes an argument of ArgumentEvaluator. If an ArgumentEvaluator object is instantiated without arguments, it will process the commandline arguments of the script. It is also possible to explicitely initialize the ArgumentEvaluator with a string list representing the argument list.

If it finds an unknown argument, or any other error occurs, it will log the error and the unknown argument, show the help string and exit.

```
:linenos:
:emphasize-lines: 2,3
from mb_tester.MBTester import ArgumentEvaluator;

# Evaluate the arguments from command line
cl_arguments = ArgumentEvaluator();

# Evaluate internal arguments
int_arguments = ArgumentEvaluator(aArgs = ["--log-file", "test.log"]);

# other examples
arguments = [ "--log-level", "ERROR,INFO" ];
data = ArgumentEvaluator(arguments);
# data.log_level -> [ ERROR : INFO ]
# data.log_file  -> None
# data.log_width -> None
# data.quiet     -> False
# data.dev       -> { 0 : None }

arguments = [ "--log-file", "logfile.txt" ];
data = ArgumentEvaluator(arguments);
# data.log_level -> None
# data.log_file  -> "logfile.txt"
# data.log_width -> None
# data.quiet     -> False
# data.dev       -> { 0 : None }

arguments = [ "--log-file", "logfile.txt", "--log-width", "400" ];
data = ArgumentEvaluator(arguments);
# data.log_level -> None
# data.log_file  -> "logfile.txt"
# data.log_width -> 400
# data.quiet     -> False
# data.dev       -> { 0 : None }

arguments = [ "--log-file", "logfile.txt", "--quiet" ];
data = ArgumentEvaluator(arguments);
# data.log_level -> None
# data.log_file  -> "logfile.txt"
# data.log_width -> None
# data.quiet     -> True
# data.dev       -> { 0 : None }

arguments = [ "--dev-SLIO", "COM5", "--dev-TEST", "COM7" ];
data = ArgumentEvaluator(arguments);
# data.log_level -> None
# data.log_file  -> None
# data.log_width -> None
# data.quiet     -> False
# data.dev       -> { 0      : None,
#                     "SLIO" : "COM5",
#                     "TEST" : "COM7" }
```

4.1.4 Class MBTester

4.1.4.1 Properties

4.1.4.1.1 endianness

Get and modify the current endianness setting. See *endianness* for more info. If the given string is not correct, an error message is printed.

```
class MBTester:
    @property
    def endianness(self):
```

Values

- "big"/*Endianness.BIG_ENDIAN* : Big endian (default)
- "little"/*Endianness.LITTLE_ENDIAN* : Little endian

Example

```
tester = MBTester(aEndianness = "little");
tester.endianness = "big";
tester.endianness = 15;
# Log:
# 2021.06.05 17:20:17.678 [E0000,W0000] DEBUG   The type of the endianness property is not str, the type ...
# ...was: int
tester.endianness = 'test';
# Log
# 2021.06.05 17:20:17.678 [E0000,W0000] DEBUG   The new value of endianness parameter is not valid: testfatal
```

4.1.4.1.2 fatal_error_handler

Callable property to change the reaction during tests when a *FatalError()* function called. The default is a function that calls the exit function.

Example

```
tester = MBTester();

def RaiseException():
    raise Exception("FATAL_ERROR occurred");
tester.fatal_error_handler = RaiseException;
tester.FatalError("This will raise an exception")

def ExitFromTheScript():
    exit(-1);

tester.fatal_error_handler = ExitFromTheScript;
tester.FatalError("This will exit from the script")
```

4.1.4.1.3 error_counter

The current value of the error counter as an integer. This value is readonly. This value is incremented by the *Error()* function.

```
@property
def error_counter(self) -> int
```

4.1.4.1.4 warning_counter

The current value of the warning counter as an integer. This value is readonly. This value is incrementaed by the *Warning()* function.

```
@property
def warning_counter(self) -> int
```

4.1.4.2 Methods

4.1.4.2.1 Constructor

```
def __init__( self
            , aDefaultLogLevel : Union[None, LogLevels, List[LogLevels]] = DEFAULT_LOG_LEVELS
            , aEndianness      : str                                     = Endianness.BIG_ENDIAN
            , aEvaluator       : Union[None, ArgumentEvaluator]          = None ):
```

It initializes the test environment based on default values and pre evaluated incoming command line arguments. If any error occurrs during the initialisation process it shows error string and calls the exit (Cannot change the default fatal error handler).

Arguments

- **aDefaultLogLevel**: When the log level wasn't set via the command line, then the value of this will be the default log level. If the default log level is set to 'None' value nor is it set from command line, then the FATAL_ERROR log level will be used only. The type of this parameter is block of log level constants. The default value of this argument is the *DEFAULT_LOG_LEVELS*.
- **aEndianness**: Set endianness with these strings:
  - "big"/*Endianness.BIG_ENDIAN* : Big endian (default)
  - "little"/*Endianness.LITTLE_ENDIAN* : Little endian
- **aEvaluator**: The command line argument evaluator object with the evaluated command line arguments. (default: 'None', See *ArgumentEvaluator* for more information)

Example

```
from mb_tester.MBTester import MBTester, LogLevels, ArgumentEvaluator, DEFAULT_LOG_LEVEL, Endianness;

# 1. simple script mode - the tester script is parameterizable from commandline
#  note: ArgumentEvaluator() without arguments will process the command line parameters
tester = MBTester( aDefaultLogLevel = [LogLevels.INFO, LogLevels.ERROR]
                 , aEndianness      = "big"
                 , aEvaluator       = ArgumentEvaluator() );

# 2. pytest mode - the tester script is not parameterizable from commandline
tester = MBTester( aDefaultLogLevel = DEFAULT_LOG_LEVEL
                 , aEndianness      = Endianness.LITTLE_ENDIAN)

# 3. pytest mode - the tester script is not parameterizable from commandline but logging to file instead
#    of console
#  create an empty evaluator by specifying None as an input argument
evaluator          = ArgumentEvaluator(None);
#  set values in the evaluator
evaluator.log_file = "filename.log";
evaluator.quiet    = True;
tester             = MBTester( aEndianness = Endianness.LITTLE_ENDIAN
                             , aEvaluator  = evaluator)
```

4.1.4.2.2 Destructor

Logs the result with the log level of 'INFO'. You can expect the following results:

- **error_counter=0,warning_counter=0** : "DONE - Test done."
- **error_counter=0,warning_counter>0** : "DONE - Test done with warnings."
- **error_counter>0** : "FAILED - Test failed."

This does not affect the return value of the script, if the user wants to return a value based on the test result, the following example could be used.

```
# return 1 on error at the end of the script
if (tester.error_counter > 0):
    exit(1);
```

4.1.4.2.3 Log()

Write a log message to a file and/or the console, according to the configuration.

Note that this function can be used to write also ERROR and WARNING messages but it will not increment the corresponding counters. For this reason it is better to use Error() and Warning() respectively.

```
def Log( self
       , aLogLevel : LogLevels
       , aMessage  : str       ):
```

Arguments

- **aLogLevel** : Selected log level for the message. Must use one of the predefined log levels (See in *LogLevels*).
- **aMessage** : The log message.

Return Value

No return value.

Example

```
tester.Log( aLogLevel = LogLevels.INFO
          , aMessage = 'Info Text');
# Log:
# 2021.02.11 14:30:27.120 [E0000,W0000] INFO    Info Text
tester.Log( aLogLevel = LogLevels.USER1
          , aMessage = 'User1 Text');
# Log:
# 2021.02.11 14:30:28.420 [E0001,W0000] USER1   User1 Text
tester.Log( aLogLevel = LogLevels.DEBUG
          , aMessage = 'Debug Text');
# Log:
# 2021.02.11 14:30:28.520 [E0001,W0001] DEBUG   Debug Text
```

4.1.4.2.4 LogBytes()

Logs bytes to a file and/or the console, according to the configuration.

```
def LogBytes( self
            , aLogLevel   : LogLevels
            , aTextPrefix : str
            , aBytes      : Union[List[int], bytearray] ):
```

Arguments

- **aLogLevel** : Selected log level for logging. Must use one of the predefined log levels (See in *LogLevels*).
- **aTextPrefix** : Message before the logged bytes in the log.
- **aBytes** : Bytes to log.

Return Value

No return value.

Example

```
tester = MBTester();
tester.LogBytes( aLogLevel   = LogLevels.DEBUG
               , aTextPrefix = 'ToBytes (66051) -> \'UINT32\''
               , aBytes      = [0x00,0x01,0x02,0x03]);
# Log:
# 2021.06.05 17:20:13.123 [E0000,W0000] DEBUG   ToBytes (66051) -> 'UINT32' [0x00010203]
tester.LogBytes( aLogLevel   = LogLevels.INPUT
               , aTextPrefix = 'GetData'
               , aBytes      = bytearray(b'\x00\x11\x22\x33\x44\x55'));
# Log:
# 2021.02.11 14:30:28.220 [E0000,W0000] INPUT   GetData [0x001122334455]
tester.LogBytes( aLogLevel   = LogLevels.OUTPUT
               , aTextPrefix = 'SetData'
               , aBytes      = b'\x77\x66\x55\x44\x33\x22' );
# Log:
# 2021.02.11 14:30:28.320 [E0000,W0000] OUTPUT  SetData [0x776655443322]
```

4.1.4.2.5 LogException()

This logging interface logs to a file and/or the console, according to configuration.

```
def LogException( self
                , aTextPrefix: str
                , aException : Exception
                , aLogLevel  : LogLevels = LogLevels.FATAL_ERROR ):
```

Arguments

- **aLogLevel** : Selected log level for logging (default: LogLevels.FATAL_ERROR). Use one of the predefined log levels (See in *LogLevels*).
- **aTextPrefix** : Message before the logged bytes in the log.
- **aException** : The exception to log.

Return Value

No return value.

Example

```
tester = MBTester();

tester.LogException( aPrefix    = "Dummy"
                   , aException = Exception("Test exception"));
# Log:
# 2022.08.11 16:00:00.000 [E0000,W0000] FATAL_ERR Dummy Exception [Test exception]
```

4.1.4.2.6 FatalError()

Log the fatal error message and call the *fatal_error* handler.

```
def FatalError( self
              , aMessage : str ):
```

Arguments

- **aMessage** : The fatal error output text.

4.1.4.2.7 Error()

Write the error log message and increment the *error_counter* if the condition is true.

```
def Error( self
         , aMessage          : str
         , aIncrementCounter : bool = True ):
```

Arguments

- **aMessage** : The error message.
- **aIncrementCounter** : The condition for the incrementing of the error counter.

4.1.4.2.8 Warning()

Write the warning log message and increment the *warning_counter* if the condition is true.

```
def Warning( self
           , aMessage          : str
           , aIncrementCounter : bool = True ):
```

Arguments

- **aMessage** : The warning message.
- **aIncrementCounter** : The condition for the incrementing of the warning counter.

4.1.4.2.9 ToNumber()

Converts a byte list to the selected number type. The conversion will start at the given index and take as many bytes as necessary based on the type. The conversion will use the current endianness setting.

The result is logged with DEBUG log level, except if an error occurs, then an ERROR is logged and teh error_counter incremented.

```
def ToNumber( self
            , aType       : ConvertTypes
            , aBuffer     : List[int]
            , aStartIndex : int = 0 ) -> Union[None, int, float]
```

Arguments

- **aType** : Type of the output value. It is a number type constant. (See *ConvertTypes*)
- **aBuffer** : Array of bytes (integers between 0-255).
- **aStartIndex** : It gives the start index of the conversion inside aBuffer array. (default: 0)

Return Value

The converted number.

Example

```
tester     = MBTester();
inputBlock = [0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06];
retVal     = tester.ToNumber( aType       = ConvertTypes.UINT16
                            , aBuffer     = inputBlock
                            , aStartIndex = 2); # retVal : 515 (0x0203)
# Log:
# 2022.11.18 12:41:11.762 [E0000,W0000] DEBUG     ToNumber [0x000102030405060] -> [0x0203]
# 2022.11.18 12:41:11.762 [E0000,W0000] DEBUG     ToNumber [0x0203] -> 'UINT16' (515)

retVal     = tester.ToNumber( aType   = ConvertTypes.UINT16
                            , aBuffer = [0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06]
                            , aStartIndex = 2);  # retVal : 515 (0x0203)
# Log:
# 2022.11.18 12:41:11.763 [E0000,W0000] DEBUG     ToNumber [0x000102030405060] -> [0x0203]
# 2022.11.18 12:41:11.764 [E0000,W0000] DEBUG     ToNumber [0x0203] -> 'UINT16' (515)

retVal     = tester.ToNumber( aType   = ConvertTypes.UINT16
                            , aBuffer =  [0x02, 0x03] ); # retVal : 515 (0x0203)
# Log:
# 2022.11.18 12:41:11.765 [E0000,W0000] DEBUG     ToNumber [0x0203] -> [0x0203]
# 2022.11.18 12:41:11.766 [E0000,W0000] DEBUG     ToNumber [0x0203] -> 'UINT16' (515)

retval     = tester.ToNumber( aType   = ConvertTypes.UINT32
                            , aBuffer =  [0x00, 0x01, 0x02, 0x03] );
# Log:
# 2022.11.18 12:41:11.767 [E0000,W0000] DEBUG     ToNumber [0x00010203] -> [0x00010203]
# 2022.11.18 12:41:11.768 [E0000,W0000] DEBUG     ToNumber [0x00010203] -> 'UINT32' (66051)

retVal     = tester.ToNumber( aType       = ConvertTypes.UINT16
                            , aBuffer     = [0x02, 0x03]
                            , aStartIndex = 5 );
# Log:
# 2022.11.18 12:41:11.769 [E0001,W0000] ERROR     ToNumber Index is out of the range (Index: 5, Len: 2)
retVal     = tester.ToNumber( aType   = ConvertTypes.UINT64
                            , aBuffer =  [0x03, 0x04, 0x05] );
# Log:
# 2022.11.18 12:41:11.769 [E0002,W0000] ERROR     ToNumber The input buffer is too small (Min len: 8, Len: 3)
```

4.1.4.2.10 ToBytes()

Converts from a number to a byte list based on the given type. The conversion will use the current endianness setting.

The result is logged with DEBUG log level, except if an error occurs, then an ERROR is logged and teh error_counter incremented.

```
def ToBytes( self
           , aType  : ConvertTypes
           , aValue : Union[int, float, str]) -> List[int]
```

Arguments

- **aType** : Type of the input value. It is a number type constant. (See *ConvertTypes*)
- **aValue** : The number for converting to byte block.

Return Value

Returns 'None' if any error occurs during conversion, otherwise the converted block.

Example

```
tester = MBTester();
retVal = tester.ToBytes( aType  = ConvertTypes.UINT32
                       , aValue = 66051 ); # retVal: [0x00, 0x01, 0x02, 0x03]
# Log:
# 2022.11.18 13:04:50.042 [E0000,W0000] DEBUG     ToBytes (66051) -> 'UINT32' [0x00010203]
retVal = tester.ToBytes( aType  = ConvertTypes.UINT8
                       , aValue = 255 ); # retVal: [0xff]
# Log:
# 2022.11.18 13:04:50.043 [E0000,W0000] DEBUG     ToBytes (255) -> 'UINT8' [0xff]
retVal = tester.ToBytes( aType  = ConvertTypes.FLOAT
                       , aValue = 123412341234.1212341234 ); # retVal: [0x97,0xdf,0xe5,0x51]
# Log:
# 2022.11.18 13:04:50.046 [E0000,W0000] DEBUG     ToBytes (123412341234.12123) -> 'FLOAT' [0x97dfe551]
retVal = tester.ToBytes( aType  = ConvertTypes.STRING
                       , aValue = 'abab' ); # retVal: [0x65,0x66,0x65,0x66]
# Log:
# 2022.11.18 13:04:50.047 [E0000,W0000] DEBUG     ToBytes (abab) -> 'STRING' [0x61626162]
```

4.1.4.2.11 ToSubBlock()

Creates a sub-block from another block according to start index and length.

The result is logged with DEBUG log level, except if an error occurs, then an ERROR is logged and teh error_counter incremented.

```
def ToSubBlock( self
              , aBuffer     : List[int]
              , aOutLength  : int
              , aStartIndex : int = 0) -> List[int]
```

Arguments

- **aBuffer** : Input byte block.
- **aOutLength** : Length of the output block.
- **aStartIndex** : Start index from where to extract the output string.

Return Value

Returns 'None' if any error occurs during conversion, otherwise the converted block.

Example

```
tester     = MBTester();
inputBlock = [0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06];
retVal = tester.ToSubBlock( aBuffer     = inputBlock
                          , aOutLength  = 5
                          , aStartIndex = 1 ); # retVal: [0x01, 0x02, 0x03, 0x04, 0x05]
# Log:
# 2022.11.18 13:26:45.643 [E0000,W0000] DEBUG     ToSubBlock [0x00010203040506] -> [0x01020304055]
retVal = tester.ToSubBlock( aBuffer     = [0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06]
                          , aOutLength  = 5
                          , aStartIndex = 1 ); # retVal: [0x01, 0x02, 0x03, 0x04, 0x05]
# Log:
# 2022.11.18 13:26:45.643 [E0000,W0000] DEBUG     ToSubBlock [0x00010203040506] -> [0x0102030405]
retVal = tester.ToSubBlock( aBuffer     = inputBlock
                          , aOutLength  = 3
                          , aStartIndex = 2 ); # retVal: [0x02, 0x03, 0x04]
# Log:
# 2022.11.18 13:26:45.644 [E0000,W0000] DEBUG     ToSubBlock [0x00010203040506] -> [0x020304]
retVal = tester.ToSubBlock( aBuffer     = [0x01,0x03]
                          , aOutLength  = 2
                          , aStartIndex = 5 ); # retVal: None
# Log:
# 2022.11.18 13:57:48.632 [E0001,W0000] ERROR     ToSubBlock The index is out of range (Index: 5, Len: 2)2)
```

4.1.4.2.12 ToString()

Converts the input block to a string.

The result is logged with DEBUG log level, except if an error occurs, then an ERROR is logged and teh error_counter incremented.

```
def ToString( self
            , aBuffer     : List[int]
            , aStartIndex : int = 0) -> str
```

Arguments

- **aBuffer** : Input byte block.
- **aStartIndex** : Start index from where to extract the output string.

Return Value

Returns 'None' if any error occurs during conversion, otherwise the converted string.

Example

```
tester     = MBTester();
inputBlock = [0x65, 0x66, 0x65, 0x66];
retVal = tester.ToString( aBuffer = inputBlock ); # retVal : 'efef'
# Log:
# 2022.11.18 14:17:26.044 [E0000,W0000] DEBUG     ToString [0x65666566] -> [0x65666566]
# 2022.11.18 14:17:26.045 [E0000,W0000] DEBUG     ToString [0x65666566](0) -> 'efef'

retVal = tester.ToString( aBuffer     = inputBlock
                        , aStartIndex = 1 );  # retVal : 'fef'
# Log:
# 2022.11.18 14:17:36.263 [E0000,W0000] DEBUG     ToString [0x65666566] -> [0x666566]
# 2022.11.18 14:17:36.263 [E0000,W0000] DEBUG     ToString [0x666566](1) -> 'fef'
```

4.1.4.2.13 IsTrue() and IsTrueW()

Note: The version with W generates a warning instead of an error

Tests the input condition for being true.

If the test fails (the input is evaluated as false), the error counter will be incremented. If the test passes, log level INFO is logged.

```
def IsTrue( self
          , aCondition : bool
          , aLabel     : Union[None, str] = None ) -> bool
```

Arguments

- **aCondition** : The condition to test for being true.
- **aLabel** : See *Common Function Parameters*.

Return Value:

Returns 'True' if the condition is 'True', otherwise it returns 'False'.

Example

```
tester = MBTester();
retVal = tester.IsTrue(aCondition = True); # retVal : True
# Log:
# 2022.11.18 14:21:42.949 [E0000,W0000] INFO      IsTrue : Correct
retVal = tester.IsTrue(aCondition = False); # retVal : False
# Log:
# 2022.11.18 14:21:42.959 [E0001,W0000] ERROR     IsTrue : Incorrect
retVal = tester.IsTrue(aCondition = True, aLabel = "Additional info"); # retVal : True
# Log:
# 2022.11.18 14:21:42.963 [E0001,W0000] INFO      IsTrue : Correct - Additional info
retVal = tester.IsTrue(aCondition = False, aLabel = "Additional info"); # retVal : False
# Log:
# 2022.11.18 14:21:42.964 [E0002,W0000] ERROR     IsTrue : Incorrect - Additional info
```

4.1.4.2.14 IsFalse() and IsFalseW()

Note: The version with W generates a warning instead of an error

Tests the input condition for being false.

If the test fails (the input is evaluated as true), the error counter will be incremented.

```
def IsFalse( self
           , aCondition : bool
           , aLabel     : Union[None, str] = None ) -> bool
```

Arguments

- **aCondition** : The condition to test for being false.
- **aLabel** : See *Common Function Parameters*.

Return Value:

Returns 'True' if the condition is 'False', otherwise it returns 'False'.

Example

```
tester = MBTester();
retVal = tester.IsFalse(aCondition = False); # retVal : True
# Log:
# 2022.11.18 14:47:49.136 [E0000,W0000] INFO      IsFalse : Correct
retVal = tester.IsFalse(aCondition = True); # retVal : False
# Log:
# 2022.11.18 14:47:49.136 [E0001,W0000] ERROR     IsFalse : Incorrect
retVal = tester.IsFalse(aCondition = False, aLabel = "Additional info"); # retVal : True
# Log:
# 2022.11.18 14:47:49.136 [E0001,W0000] INFO      IsFalse : Correct - Additional info
retVal = tester.IsFalse(aCondition = True, aLabel = "Additional info"); # retVal : False
# Log:
# 2022.11.18 14:47:49.136 [E0002,W0000] ERROR     IsFalse : Incorrect - Additional info
```

4.1.4.2.15 AreEqual() and AreEqualW()

Note: The version with W generates a warning instead of an error

Tests two values for being equal. The valid input types of the values are: *int, float, list, set, str, NoneType*.

If the test fails (the inputs are not equal), the error counter will be incremented.

```
def AreEqual( self
            , aExpected : AnyType
            , aValue    : AnyType
            , aStrict   : bool             = True
            , aLabel    : Union[None, str] = None ) -> bool
```

Arguments

- **aExpected** : The expected value.
- **aValue** : The value for comparison.
- **aStrict** : If this argument is 'True', the comparison of strings will be case sensitive, otherwise not. (default: True)
- **aLabel** : See *Common Function Parameters*.

Return Value

Returns 'True' if the two values/objects are the same, otherwise it returns 'False'.

Example

```
tester = MBTester();
result = tester.AreEqual(
              aExpected = int(120)
            , aValue    = float(120.0) );
# The result is True
# Log:
# 2021.05.17 13:42:01.340 [E0000,W0000] INFO    Expected : type:<class 'int'> value:120
# 2021.05.17 13:42:01.341 [E0000,W0000] INFO    Value    : type:<class 'float'> value:120.0
# 2021.05.17 13:42:01.342 [E0000,W0000] INFO    AreEqual : Match.
result = tester.AreEqual(
              aExpected = int(120)
            , aValue    = int(120) );
# The result is True
# Log:
# 2021.05.17 13:42:01.343 [E0000,W0000] INFO    Expected : type:<class 'int'> value:120
# 2021.05.17 13:42:01.344 [E0000,W0000] INFO    Value    : type:<class 'int'> value:120
# 2021.05.17 13:42:01.345 [E0000,W0000] INFO    AreEqual : Match.
result = tester.AreEqual(
              aExpected = "Test string"
```

(continues on next page)

(continued from previous page)

```
            , aValue    = "Test string" );
# The result is True
# Log:
# 2021.05.17 13:42:01.346 [E0000,W0000] INFO    Expected : type:<class 'str'> len:11 value:"Test string"
# 2021.05.17 13:42:01.347 [E0000,W0000] INFO    Value    : type:<class 'str'> len:11 value:"Test string"
# 2021.05.17 13:42:01.348 [E0000,W0000] INFO    AreEqual : Match.
result = tester.AreEqual(
              aExpected = [ 0x00, 0x01, 0x02 ]
            , aValue    = [ 0x00, 0x01, 0x02 ] );
# The result is False
# Log:
# 2021.05.17 13:42:01.349 [E0000,W0000] INFO    Expected : type:<class 'list'> len:3 value:[ '0', '1', '2' ]
# 2021.05.17 13:42:01.350 [E0000,W0000] INFO    Value    : type:<class 'list'> len:3 value:[ '0', '1', '2' ]
# 2021.05.17 13:42:01.351 [E0000,W0000] INFO    AreEqual : Match.
result = tester.AreEqual(
              aExpected = [ 0x00, 0x01, 0x02 ]
            , aValue    = [ 0x01, 0x01, 0x02 ] );
# The result is False
# Log:
# 2021.05.17 13:42:01.352 [E0000,W0000] INFO    Expected : type:<class 'list'> len:3 value:[ '0', '1', '2' ]
# 2021.05.17 13:42:01.353 [E0000,W0000] INFO    Value    : type:<class 'list'> len:3 value:[ '1', '1', '2' ]
# 2021.05.17 13:42:01.354 [E0001,W0000] ERROR   AreEqual : Value Mismatch.
result = tester.AreEqual(
              aExpected = "ABC"
            , aValue    = [ 65, 66, 67 ] );
# The result is False due to type difference
# Log:
# 2021.05.17 13:42:01.355 [E0001,W0000] INFO    Expected : type:<class 'str'> len:3 value:"ABC"
# 2021.05.17 13:42:01.356 [E0001,W0000] INFO    Value    : type:<class 'list'> len:3 value:[ '65', '66', '67' ]
# 2021.05.17 13:42:01.357 [E0002,W0000] ERROR   AreEqual : Type Mismatch.
tester.AreEqual(
              aExpected = 120
            , aValue    = 120
            , aLabel    = "Additional info" );
# Log:
# 2021.05.17 13:42:01.340 [E0000,W0000] INFO    Expected : type:<class 'int'> value:120
# 2021.05.17 13:42:01.341 [E0000,W0000] INFO    Value    : type:<class 'int'> value:120
# 2021.05.17 13:42:01.342 [E0000,W0000] INFO    AreEqual : Match. - Additional info
tester.AreEqual(
              aExpected = 120
            , aValue    = 121
            , aLabel    = "Additional info" );
# Log:
# 2021.05.17 13:42:01.340 [E0000,W0000] INFO    Expected : type:<class 'int'> value:120
# 2021.05.17 13:42:01.341 [E0000,W0000] INFO    Value    : type:<class 'int'> value:121
# 2021.05.17 13:42:01.342 [E0001,W0000] ERROR   AreEqual : Value Mismatch. - Additional info
result = tester.AreEqual(
              aExpected = "Test string"
            , aValue    = "Test string" );
# strict : True (default) , result : True
result = tester.AreEqual(
              aExpected = "TesT String"
            , aValue    = "Test string" );
# strict : True (default) , result : False
result = tester.AreEqual(
              aExpected = "Test string"
            , aValue    = "Test string"
            , aStrict   = True );
# result : True
result = tester.AreEqual(
              aExpected = "TesT String"
            , aValue    = "Test string"
            , aStrict   = True );
# result : False
result = tester.AreEqual(
              aExpected = "Test string"
```

(continues on next page)

(continued from previous page)

```
            , aValue    = "Test string"
            , aStrict   = False );
# result : True
result = tester.AreEqual(
              aExpected = "TesT String"
            , aValue    = "Test string"
            , aStrict   = False );
# result : True
result = tester.AreEqual(
              aExpected = "Test string"
            , aValue    = "Longer string"
            , aStrict   = True );
# result : False
# Log:
# 2021.05.17 13:42:01.510 [E0002,W0000] INFO    Expected : type:<class 'str'> len:11 value:"Test string"
# 2021.05.17 13:42:01.511 [E0002,W0000] INFO    Value    : type:<class 'str'> len:13 value:"Longer string"
# 2021.05.17 13:42:01.512 [E0003,W0000] ERROR   AreEqual : Value Mismatch.
result = tester.AreEqual(
              aExpected = "TesT String"
            , aValue = "Longer string"
            , aStrict = False );
# result : False
# Log:
# 2021.05.17 13:42:01.513 [E0003,W0000] INFO    Expected : type:<class 'str'> len:11 value:"Test string"
# 2021.05.17 13:42:01.514 [E0003,W0000] INFO    Value    : type:<class 'str'> len:13 value:"Longer string"
# 2021.05.17 13:42:01.515 [E0004,W0000] ERROR   AreEqual : Value Mismatch.
```

4.1.4.2.16 AreNotEqual() and AreNotEqualW()

Note: The version with W generates a warning instead of an error

Tests two values for not being equal. The valid input types of the values are: *int, float, list, set, str, NoneType*.

If the test fails (the inputs are equal), the error counter will be incremented.

```
def AreNotEqual( self
               , aNotExpected : AnyType
               , aValue       : AnyType
               , aStrict      : bool             = True
               , aLabel       : Union[None, str] = None ) -> bool
```

Arguments

- **aNotExpected** : The expected wrong value.
- **aValue** : The value for comparison.
- **aStrict** : If the type of values are 'set' or 'dict' or 'list', and aStrict is 'True', then an error is generated when the sizes are different. If the type of values are 'str' and the value of aStrict is True, then the comparison will be case sensitive. (default: True)
- **aLabel** : See *Common Function Parameters*.

Return Value

Returns 'True' if the two values/objects are the not the same, otherwise it returns 'False'.

Example

```
tester = MBTester();
retVal = tester.AreNotEqual( aNotExpected = int(120)
                           , aValue       = float(120.0) ); # retVal: False
```

(continues on next page)

(continued from previous page)

```
# Log:
# 2021.09.21 17:51:50.573 [E0000,W0000] INFO    Not Expected : type:<class 'int'> value:120
# 2021.09.21 17:51:50.574 [E0000,W0000] INFO    Value        : type:<class 'float'> value:120.0
# 2021.09.21 17:51:50.574 [E0001,W0000] ERROR   AreNotEqual  : Value Match.
retVal = tester.AreNotEqual( aNotExpected = int(120)
                           , aValue       = int(120) ); # retVal: False
# Log:
# 2021.09.21 17:52:28.549 [E0001,W0000] INFO    Not Expected : type:<class 'int'> value:120
# 2021.09.21 17:52:28.549 [E0001,W0000] INFO    Value        : type:<class 'int'> value:120
# 2021.09.21 17:52:28.550 [E0002,W0000] ERROR   AreNotEqual  : Value Match.
retVal = tester.AreNotEqual( aNotExpected = "Test string"
                           , aValue       = "Test string"); # retVal: False
# Log:
# 2021.09.21 17:52:54.451 [E0002,W0000] INFO    Not Expected : type:<class 'str'> len:11 value:"Test string"
# 2021.09.21 17:52:54.451 [E0002,W0000] INFO    Value        : type:<class 'str'> len:11 value:"Test string"
# 2021.09.21 17:52:54.451 [E0003,W0000] ERROR   AreNotEqual  : Value Match.
retVal = tester.AreNotEqual( aNotExpected = int(120)
                           , aValue       = float(123.0) ); # retVal: True
# Log:
# 2021.09.21 17:53:16.631 [E0003,W0000] INFO    Not Expected : type:<class 'int'> value:120
# 2021.09.21 17:53:16.631 [E0003,W0000] INFO    Value        : type:<class 'float'> value:123.0
# 2021.09.21 17:53:16.631 [E0003,W0000] INFO    AreNotEqual  : Mismatch.
retVal = tester.AreNotEqual( aNotExpected = int(124)
                           , aValue       = int(120) ); # retVal: True
# Log:
# 2021.09.21 17:53:34.811 [E0003,W0000] INFO    Not Expected : type:<class 'int'> value:124
# 2021.09.21 17:53:34.811 [E0003,W0000] INFO    Value        : type:<class 'int'> value:120
# 2021.09.21 17:53:34.811 [E0003,W0000] INFO    AreNotEqual  : Mismatch.
retVal = tester.AreNotEqual( aNotExpected = "Test String"
                           , aValue       = "Test string" ); # retVal: True
# Log:
# 2021.09.21 17:53:58.677 [E0003,W0000] INFO    Not Expected : type:<class 'str'> len:11 value:"Test String"
# 2021.09.21 17:53:58.677 [E0003,W0000] INFO    Value        : type:<class 'str'> len:11 value:"Test string"
# 2021.09.21 17:53:58.678 [E0003,W0000] INFO    AreNotEqual  : Mismatch.
retVal = tester.AreNotEqual( aNotExpected = [ 0x00, 0x01, 0x02 ]
                           , aValue       = [ 0x01, 0x01, 0x02 ] ); # retVal: True
# Log:
# 2021.09.21 17:54:18.739 [E0003,W0000] INFO    Not Expected : type:<class 'list'> len:3 value:['0', '1', '2']
# 2021.09.21 17:54:18.739 [E0003,W0000] INFO    Value        : type:<class 'list'> len:3 value:['1', '1', '2']
# 2021.09.21 17:54:18.739 [E0003,W0000] INFO    AreNotEqual  : Mismatch.
retVal = tester.AreNotEqual( aNotExpected = [ 0x00, 0x01, 0x02, 0x02 ]
                           , aValue       = [ 0x01, 0x01, 0x02 ] ); # retVal: True
# Log:
# 2021.09.21 17:57:20.952 [E0000,W0000] INFO    Not Expected : type:<class 'list'> len:4 value:['0', '1', '2', '2']
# 2021.09.21 17:57:20.952 [E0000,W0000] INFO    Value        : type:<class 'list'> len:3 value:['1', '1', '2']
# 2021.09.21 17:57:20.953 [E0001,W0000] ERROR   AreNotEqual  : Mismatch.
retVal = tester.AreNotEqual( aNotExpected = "ABC"
                           , aValue       = [ 65, 66, 67 ] ); # retVal: False due to type difference
# Log:
# 2021.09.21 17:57:43.072 [E0001,W0000] INFO    Not Expected : type:<class 'str'> len:3 value:"ABC"
# 2021.09.21 17:57:43.072 [E0001,W0000] INFO    Value        : type:<class 'list'> len:3 value:['65', '66', '67']
# 2021.09.21 17:57:43.072 [E0002,W0000] ERROR   AreNotEqual  : Type Mismatch.
retVal = tester.AreNotEqual( aNotExpected = None
                           , aValue       = [ 65, 66, 67 ]
                           , aStrict      = False ); # retVal: True due to not strict
# Log:
# 2021.09.21 17:57:55.998 [E0002,W0000] INFO    Not Expected : type:<class 'NoneType'> value:None
# 2021.09.21 17:57:55.998 [E0002,W0000] INFO    Value        : type:<class 'list'> len:3 value:['65', '66', '67']
# 2021.09.21 17:57:55.998 [E0002,W0000] INFO    AreNotEqual  : Type Mismatch.
retVal = tester.AreNotEqual( aNotExpected = 120
                           , aValue       = 120
                           , aLabel       = "Additional info" );
# Log:
# 2021.09.21 17:58:08.037 [E0002,W0000] INFO    Not Expected : type:<class 'int'> value:120
# 2021.09.21 17:58:08.037 [E0002,W0000] INFO    Value        : type:<class 'int'> value:120
# 2021.09.21 17:58:08.037 [E0003,W0000] ERROR   AreNotEqual  : Value Match. - Additional info
```

(continues on next page)

(continued from previous page)

```
retVal = tester.AreNotEqual( aNotExpected = 120
                           , aValue       = 121
                           , aLabel       = "Additional info" );
# Log:
# 2021.09.21 17:58:27.084 [E0003,W0000] INFO    Not Expected : type:<class 'int'> value:120
# 2021.09.21 17:58:27.084 [E0003,W0000] INFO    Value        : type:<class 'int'> value:121
# 2021.09.21 17:58:27.085 [E0003,W0000] INFO    AreNotEqual  : Mismatch. - Additional info
retVal = tester.AreNotEqual( aNotExpected = "Test string"
                           , aValue       = "Test string" ); # strict : True (default) , retVal: False
retVal = tester.AreNotEqual( aNotExpected = "TesT String"
                           , aValue       = "Test string" ); # strict : True (default) , result : True
retVal = tester.AreNotEqual( aNotExpected = "Test string"
                           , aValue       = "Test string"
                           , aStrict      = True ); # retVal: False
retVal = tester.AreNotEqual( aNotExpected = "TesT String"
                           , aValue       = "Test string"
                           , aStrict      =  True ); # retVal: True
retVal = tester.AreNotEqual( aNotExpected = "Test string"
                           , aValue       = "Test string"
                           , aStrict      =  False ); # retVal: False
retVal = tester.AreNotEqual( aNotExpected = "TesT String"
                           , aValue       = "Test string"
                           , aStrict      =  False ); # retVal: False
retVal = tester.AreNotEqual( aNotExpected = "Test string"
                           , aValue       = "Longer string"
                           , aStrict      =  True ); # retVal: True
# Log:
# 2021.09.21 17:59:13.981 [E0012,W0000] INFO    Not Expected : type:<class 'str'> len:11 value:"Test string"
# 2021.09.21 17:59:13.981 [E0012,W0000] INFO    Value        : type:<class 'str'> len:13 value:"Longer string"
# 2021.09.21 17:59:13.981 [E0012,W0000] INFO    AreNotEqual  : Mismatch.
retVal = tester.AreNotEqual( aNotExpected = "TesT String"
                           , aValue       = "Longer string"
                           , aStrict      = False ); # retVal: True
# Log:
# 2021.09.21 17:59:30.516 [E0012,W0000] INFO    Not Expected : type:<class 'str'> len:11 value:"TesT String"
# 2021.09.21 17:59:30.516 [E0012,W0000] INFO    Value        : type:<class 'str'> len:13 value:"Longer string"
# 2021.09.21 17:59:30.516 [E0012,W0000] INFO    AreNotEqual  : Mismatch.
```

4.1.4.2.17 IsClose() and IsCloseW()

Note: The version with W generates a warning instead of an error

It tests the two input values, whether they are in the specified range of each other. If both tolerances are given, the checks will be ORed together.

If the test fails (the difference of the outputs is outside of the specified tolerances), the error counter will be incremented.

```
def IsClose( self
           , aExpected     : Union[int, float]
           , aValue        : Union[int, float]
           , aAbsTolerance : Union[int, float] = 0.0
           , aRelTolerance : Union[int, float] = 0.0
           , aLabel        : Union[None, str]  = None) -> bool
```

Arguments

- **aExpected** : The expected value.
- **aValue** : The value for validation.
- **aAbsTolerance** : The absolute tolerance. (default: 0.0)
- **aRelTolerance** : The relative tolerance. (default: 0.0)

- **aLabel** : See *Common Function Parameters*.

Return Value

Returns 'True' if the two values are in the specified range of each other, otherwise 'False'

Example

```
tester = MBTester();
retVal = tester.IsClose( aExpected = 5, aValue = 5 );
# Log:
# 2022.11.18 15:19:36.202 [E0000,W0000] INFO      Expected : type:<class 'int'> value:5
# 2022.11.18 15:19:36.205 [E0000,W0000] INFO      Value    : type:<class 'int'> value:5
# 2022.11.18 15:19:36.205 [E0000,W0000] INFO      IsClose : The values are close.
retVal = tester.IsClose( aExpected = 5, aValue = 6 );
# Log:
# 2022.11.18 15:19:36.209 [E0000,W0000] INFO      Expected : type:<class 'int'> value:5
# 2022.11.18 15:19:36.213 [E0000,W0000] INFO      Value    : type:<class 'int'> value:6
# 2022.11.18 15:19:36.214 [E0001,W0000] ERROR     IsClose : The values are not close to each other. (5/6/0.0/0.
↪0)
retVal = tester.IsClose( aExpected     = 5
                       , aValue        = 6
                       , aAbsTolerance = 1 );
retVal = tester.IsClose( aExpected     = 5
                       , aValue        = 7
                       , aAbsTolerance = 1 );
retVal = tester.IsClose( aExpected     = 5
                       , aValue        = 7
                       , aAbsTolerance = 1
                       , aRelTolerance = 1 );
retVal = tester.IsClose( aExpected = 5, aValue = True );
# Log:
# 2022.11.18 15:19:36.265 [E0002,W0000] INFO      Expected : type:<class 'int'> value:5
# 2022.11.18 15:19:36.266 [E0002,W0000] INFO      Value    : type:<class 'bool'> value:True
# 2022.11.18 15:19:36.269 [E0003,W0000] ERROR     IsClose : Unaccepted type(s).
retVal = tester.IsClose( aExpected = 5
                       , aValue    = 5
                       , aLabel    = "Additional info" );
# Log:
# 2022.11.18 15:19:36.274 [E0003,W0000] INFO      Expected : type:<class 'int'> value:5
# 2022.11.18 15:19:36.283 [E0003,W0000] INFO      Value    : type:<class 'int'> value:5
# 2022.11.18 15:19:36.286 [E0003,W0000] INFO      IsClose : The values are close. - Additional info
tester.IsClose(aExpected = 5, aValue = 6);
# Log:
# 2022.11.18 15:45:00.087 [E0003,W0000] INFO      Expected : type:<class 'int'> value:5
# 2022.11.18 15:45:00.092 [E0003,W0000] INFO      Value    : type:<class 'int'> value:6
# 2022.11.18 15:45:00.092 [E0004,W0000] ERROR     IsClose : The values are not close to each other. (5/6/0.0/0.
↪0)
```

4.1.5 Class MBPort

A port is used by bus to communicate with a device. There are specific port classes for different buses. All port classes are derived from this class.

This class is for internal use, only the derived classes should be instantiated.

4.1.5.1 Properties

4.1.5.1.1 is_open

A readonly boolean property indicating whether the underlying connection has been successfully opened (True) or not (False).

Is_open is closely related to scope_id:

- When scope_id is 0 then is_open is False.
- When scope_id is a positive non-zero value then is_open is True.

4.2 MPBus

4.2.1 Introduction

This module implements the Belimo MP bus protocol.

4.2.1.1 Initialization

```
from mb_tester.MBTester     import MBTester;
from mb_tester.MPSerialPort import MPSerialPort;
from mb_tester.MPBus        import MPBus;

# An MBTester object is required
tester = MBTester(...);

# An MPSerialPort object is required
mpserial = MPSerialPort(...);

# create an MPBus instance
mpbus = MPBus( aTester = tester, aDefaultPort = mpserial );
```

4.2.1.2 MP Command Parameters

Common parameters used in MP Commands, for example: *Peek*, *Poke*, *SetData*, *GetData*, etc.

- **aCommunicationType** : Override the MP address (communication type) of the used port for the current command. For possible values see *CommunicationTypes*.

4.2.1.3 Timeouts

The time limit to wait for an answer is 800 ms. If a no answer is received within the timeout, retry 3 times by default. The number of retries can be set in the MPSerialPort.

4.2.2 Types

These are the types of configurations related to MPBus.

4.2.2.1 MPBaudRates

Import : 'from mb_tester.MPSerialPort import MPBaudRates'

```
class MPBaudRates(IntEnum):
    B1200   = 1200;
    B9600   = 9600;
    B38400  = 38400;
```

4.2.2.2 CommunicationTypes

Import : 'from mb_tester.MPBus import CommunicationTypes'

```
class CommunicationTypes(IntEnum):
    UNKNOWN   = 0;
    MP1       = 1;
    MP2       = 2;
    MP3       = 3;
    MP4       = 4;
    MP5       = 5;
    MP6       = 6;
    MP7       = 7;
    MP8       = 8;
    MP9       = 9;
    MP10      = 10;
    MP11      = 11;
    MP12      = 12;
    MP13      = 13;
    MP14      = 14;
    MP15      = 15;
    MP16      = 16;
    MASTER    = 17;
    PP        = 18;
    PPX       = 19;
    BROADCAST = 20;
    ONEVENT   = 21;
    TOOLSLAVE = 22;
```

4.2.3 Class MPBus

This class provides MP communication with MP slaves. It implements a selection of the most common MP commands. In addition it enables to send any MP command using the GenerateCommandBytes() and SendRaw() functions.

4.2.3.1 Methods

4.2.3.1.1 Constructor

```
def __init__( self
            , aTester      : MBTester
            , aDefaultPort : MPBusPort ):
```

Arguments

- **aTester** : An MBTester object. It is used for logging results and accessing parameters
- **aDefaultPort** : An MPBusPort object to be used as default. If another port is to be used, it can be set for each command.

4.2.3.1.2 Peek()

Sends the MP_Peek (1) command.

Syntax

```
def Peek( self
        , aAddress           : int
        , aLength            : int
        , aPort              : Union[None, MPBusPort]               = None
        , aCommunicationType : Union[None, str, CommunicationTypes] = None
        , aAcceptErrors      : Union[None, List[int]]               = None
        , aLabel             : Union[None, str]                     = None
        ) -> Tuple[int, List[int]]:
```

Arguments

- **aAddress** : Memory address on device.
- **aLength** : The length of read data from the given address.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*..
- **aLabel** : See *Common Function Parameters*..

Return Value

It returns according to the *Function Return Values*

Example

```
(resultCode, buffer) = mpbus.Peek(aAddress = 32769, aLength = 2, aLabel = 'Peek command');
# buffer = [0x12, 0x34]
# Log
# 2021.09.11 15.42.30.321 [E0000,W0000] OUTPUT  Peek command [0x01800102]
# 2021.09.11 15.42.31.322 [E0000,W0000] INPUT   Peek command [0x1234]
(resultCode, buffer) = mpbus.Peek(aAddress = 0x8003, aLength = 4, aLabel = 'Peek command');
```

(continues on next page)

(continued from previous page)

```
# buffer = [0x00, 0x11, 0x22, 0x33]
# Log
# 2021.09.11 15.42.30.321 [E0000,W0000] OUTPUT  Peek command [0x01800303]
# 2021.09.11 15.42.31.322 [E0000,W0000] INPUT   Peek command [0x00112233]
```

4.2.3.1.3 Poke()

Sends the MP_Poke (2) command.

```
def Poke( self
        , aAddress           : int
        , aBlock             : List[int]
        , aPort              : Union[None, MPBusPort]               = None
        , aCommunicationType : Union[None, str, CommunicationTypes] = None
        , aAcceptErrors      : Union[None, List[int]]               = None
        , aLabel             : Union[None, str]                     = None
        ) -> Tuple[int, List[int]]:
```

Arguments

- **aAddress** : Memory address on the device.
- **aBlock** : The block for writing to the given address.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, buffer) = mpbus.Poke( aAddress = 32769
                                 , aBlock = [0x12, 0x34] );
# Log
# 2021.09.11 15.42.30.700 [E0000,W0000] OUTPUT  Poke [0x0280011234]
# 2021.09.11 15.42.31.835 [E0000,W0000] INPUT   Poke []
(resultCode, buffer) = mpbus.Poke( aAddress = 32769
                                 , aBlock = [0x11, 0x12, 0x13, 0x14]
                                 , aLabel = 'Poke command' );
# Log
# 2021.09.11 15.42.30.700 [E0000,W0000] OUTPUT  Poke command [0x02800111121314]
# 2021.09.11 15.42.31.835 [E0000,W0000] INPUT   Poke command []
```

4.2.3.1.4 GetData()

Sends the MP_Get_Data (111) command.

```
def GetData( self
           , aModelId           : int
           , aAuto              : bool                                 = True
           , aPort              : Union[None, MPBusPort]               = None
           , aCommunicationType : Union[None, str, CommunicationTypes] = None
           , aAcceptErrors      : Union[None, List[int]]               = None
```

(continues on next page)

(continued from previous page)

```
          , aLabel            : Union[None, str]                    = None
          ) -> Tuple[int, List[int]]:
```

Arguments

- **aModelId** : Model identifier on the device.
- **aAuto** : If the value is True, then *GetNextBlock* will be called automatically. (default: True)
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*.

When the call was a success and the length of payload is more than 4 and aAuto parameter is 'False', then additional call(s) of *GetNextBlock* function is/are required. The last byte of the data list contains the amount of remaining bytes. Use it for calculate the count of GetNextBlock calls.

Example

```
(resultCode, buffer) = mpbus.GetData(aModelId = 17, aLabel = 'Get Model: 17' );
# Log
# 2021.09.11 15.42.30.700 [E0000,W0000] OUTPUT  Get Model: 17 [0x386f00111056]
# 2021.09.11 15.42.31.835 [E0000,W0000] INPUT   Get Model: 17 [0x5d0102030402dc87]
(resultCode, buffer) = mpbus.GetData(aModelId = 13, aLabel = 'Get Model: 13' );
# Log
# 2021.09.11 15.42.30.700 [E0000,W0000] OUTPUT  Get Model: 13 [0x386f000d90ca]
# 2021.09.11 15.42.31.835 [E0000,W0000] INPUT   Get Model: 13 [0x2d138840f6]
```

4.2.3.1.5 GetNextBlock()

Sends the MP_Get_NextBlock(13) command.

```
def GetNextBlock( self
                , aBlockNr          : int
                , aPort             : Union[None, MPBusPort]              = None
                , aCommunicationType : Union[None, str, CommunicationTypes] = None
                , aAcceptErrors     : Union[None, List[int]]              = None
                , aLabel            : Union[None, str]                    = None
                ) -> Tuple[int, List[int]]:
```

Arguments

- **aBlockNr** : The identifier of next block. 0 < aBlockNr < 256
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. If length of payload is 7, then this function needs to call again with incremented block identifier.

Example

```
# Get the value of model with id 12 (returns with a length of maximum four bytes)
(resultCode, block) = mpbus.GetData(aModelId = 12, aAuto = False);
if (type(block) is list) and (len(block) > 4) :
    blockNr = 1;
    countOfRemainingBytes = block[4];
    block = block[:4];
    while countOfRemainingBytes > 0 :
        # The next block, if it has more data
        (resultCode, data) = mpbus.GetNextBlock(aBlockNr = blockNr);
        if (type(data) is list) :
            countOfRemainingBytes -= len(data);
            block += data;
        else:
            break;
        blockNr += 1;
```

4.2.3.1.6 SetData()

Sends the MP_Set_Data (110) command.

```
def SetData( self
           , aModelId           : int
           , aBuffer            : List[int]
           , aAuto              : bool                                = True
           , aPort              : Union[None, MPBusPort]              = None
           , aCommunicationType : Union[None, str, CommunicationTypes] = None
           , aAcceptErrors      : Union[None, List[int]]              = None
           , aLabel             : Union[None, str]                    = None
           ) -> Tuple[int, List[int]:
```

Arguments

- **aModelId** : Model identifier on the device. (This number is two bytes long unsigned type.)
- **aBuffer** : Value in byte buffer for the model. If the size of the block is more than 4 bytes and the aAuto is set to 'True' (default), then *SetNextBlock* will be called automatically.
- **aAuto** : If the value is 'True' and the size of aBuffer is more than 4 bytes, then *SetNextBlock* will be called automatically. (default: True)
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, buffer) = mpbus.SetData(aModelId = 12, aBuffer = data, aLabel = "Set Model: 12" );
# Log
# 2021.09.11 15.42.30.700 [E0000,W0000] OUTPUT  Set Model: 12 [0x586e000c012cccdb]
# 2021.09.11 15.42.31.835 [E0000,W0000] INPUT   Set Model: 12 [0x0d808d]
(resultCode, buffer) = mpbus.SetData(aModelId = 18, aBuffer = data, aLabel = "Set Model: 18" );
# Log
# 2021.09.11 15.42.30.700 [E0000,W0000] OUTPUT  Set Model: 18 [0x586e0012012cccc5]
# 2021.09.11 15.42.31.835 [E0000,W0000] INPUT   Set Model: 18 [0x0d808d]
```

4.2.3.1.7 SetNextBlock()

Sends the MP_Set_NextBlock (112) command.

```
def SetNextBlock( self
                , aBuffer            : List[int]
                , aBlockNr           : int
                , aPort              : Union[None, MPBusPort]               = None
                , aCommunicationType : Union[None, str, CommunicationTypes] = None
                , aAcceptErrors      : Union[None, List[int]]               = None
                , aLabel             : Union[None, str]                     = None
                ) -> Tuple[int, List[int]]:
```

Arguments

- **aBuffer** : Value in byte buffer for the model. This command sends only the first 7 or less bytes from the buffer.
- **aBlockNr** : Block identifier number in the range [1..256]
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
data = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f ];
# Set the model with more then 4 bytes long data
(resultCode, buffer) = mpbus.SetData(aModelId = 12, aBuffer = data, aAuto = False);
if resultCode == 0 :
    # The start of set is done properly
    (resultCode, buffer) = mpbus.SetNextBlock(aBuffer = tester.ToSubBblock( aBuffer     = data
                                                                          , aOutLength  = 4
                                                                          , aStartIndex = 4 )
                                            , aBlockNr =  1);
    (resultCode, buffer) = mpbus.SetNextBlock( aBuffer = tester.ToSubBblock( aBuffer = data
                                                                           , aOutLength = 4
                                                                           , aStartIndex = 11 )
                                            , aBlockNr = 2);
```

4.2.3.1.8 SendRaw()

This function allows for the user to send a raw MP command via MPBus.

```
def SendRaw( self
           , aBuffer            : List[int]
           , aAnswerExpected    : bool                                 = True
           , aPort              : Union[None, MPBusPort]               = None
           , aCommunicationType : Union[None, str, CommunicationTypes] = None
           , aAcceptErrors      : Union[None, List[int]]               = None
           , aLabel             : Union[None, str]                     = None
           ) -> Tuple[int, List[int]]:
```

Arguments

- **aBuffer** : It contains the whole MP command to be sent out.

- **aAnswerExpected** : Answer is expected for the command or not. If it is 'True' (default), then the command waits for the answer, otherwise it will not.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*. This parameter has no effect, the communication type is determined by the contents of **aBuffer**.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. If the aAnswerExpected attribute is 'False', then the second value is always an empty list.

Example

```
(resultCode, buffer) = mpbus.SendRaw(
                            aBuffer = [0x78, 0x3D, 0x00, 0x00, 0x01, 0xF4, 0x03, 0xE8, 0x32, 0x69]
                          , aAnswerExpected = False
                          , aLabel = "Custom: MP_Set_Min_Mid_Max" );
# Log
# 2021.05.04 11:51:37.141 [E0009,W0000] OUTPUT  Custom: MP_Set_Min_Mid_Max [0x783d000001f403e83269]
# 2021.05.04 11:51:37.238 [E0009,W0000] INPUT   Custom: MP_Set_Min_Mid_Max [0x0d808d]
(resultCode, buffer) = mpbus.SendRaw( aBuffer = [0x18, 0x3B, 0x80, 0xA3]
                                    , aAnswerExpected = True
                                    , aLabel = "Custom: MP_Get_Min_Mid_Max" );
# Log
# 2021.05.04 11:51:37.392 [E0009,W0000] OUTPUT  Custom: MP_Get_Min_Mid_Max [0x183b80a3]
# 2021.05.04 11:51:37.505 [E0009,W0000] INPUT   Custom: MP_Get_Min_Mid_Max [0x6d000001f403e83241]
```

4.2.3.1.9 ColdStart()

Sends the MP_ColdStart (67) command.

```
def ColdStart( self
             , aPort              : Union[None, MPBusPort]               = None
             , aCommunicationType : Union[None, str, CommunicationTypes] = None
             , aAcceptErrors      : Union[None, List[int]]               = None
             , aLabel             : Union[None, str]                     = None
             ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, buffer) = mpbus.ColdStart( aAcceptErrors = [ 1111 ] );
# MP Command not known by slave 11
# Log
# 2021.05.06 14:06:43.022 [E0000,W0000] OUTPUT  ColdStart [0x184380db]
```

(continues on next page)

(continued from previous page)

```
# 2021.05.06 14:06:43.116 [E0000,W0000] INPUT   ColdStart [0x9d0bc056]
# 2021.05.06 14:06:43.117 [E0000,W0000] ERROR   ColdStart (1111) Command not known by the slave
```

4.2.3.1.10 GetFirmware()

Sends the MP_Get_Fimrware (82) command.

```
def GetFirmware( self
               , aPort              : Union[None, MPBusPort]               = None
               , aCommunicationType : Union[None, str, CommunicationTypes] = None
               , aAcceptErrors      : Union[None, List[int]]               = None
               , aLabel             : Union[None, str]                     = None
               ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is the bytes of firmware version.

Example

```
(resultCode, buffer) = mpbus.GetFirmware( aLabel = 'Get Firmware Info' );
# Log
# 2021.05.06 14:06:43.887 [E0000,W0000] OUTPUT  Get Firmware Info [0x285202c0b8]
# 2021.05.06 14:06:43.978 [E0000,W0000] INPUT   Get Firmware Info [0x5d10010100123c63]
```

4.2.3.1.11 SetInfoField()

Sends the MP_Set_Infofield (52) command.

Syntax

```
def SetInfoField( self
                , aBlockNr           : int
                , aBuffer            : List[int]
                , aPort              : Union[None, MPBusPort]               = None
                , aCommunicationType : Union[None, str, CommunicationTypes] = None
                , aAcceptErrors      : Union[None, List[int]]               = None
                , aLabel             : Union[None, str]                     = None
                ) -> Tuple[int, List[int]]:
```

Arguments

- **aBlockNr** : Block identifier [0-7].
- **aBuffer** : Data to be sent out. (Limited to a maximum of 5 bytes, the other bytes will be truncated)
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.

- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, buffer) = mpbus.SetInfoField( aBlockNr = 0
                                         , aBuffer = [ 0x00, 0x22, 0x00, 0x01, 0x11] );
# Log
# 2021.05.06 14:06:45.304 [E0000,W0000] OUTPUT  SetInfoField [0x7834000022000111423c]
# 2021.05.06 14:06:45.400 [E0000,W0000] INPUT   SetInfoField [0x0d808d]
```

4.2.3.1.12 GetInfoField()

Sends the MP_Get_Infofield (53) command.

Syntax

```
def GetInfoField( self
                , aBlockNr           : int
                , aPort              : Union[None, MPBusPort]               = None
                , aCommunicationType : Union[None, str, CommunicationTypes] = None
                , aAcceptErrors      : Union[None, List[int]]               = None
                , aLabel             : Union[None, str]                     = None
                ) -> Tuple[int, List[int]]:
```

Arguments

- **aBlockNr** : The block identifier number.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*.

Example

```
(resultCode, buffer) = mpbus.GetInfoField( aBlockNr = 0, aLabel = 'GetInfofield command' );
# Log
# 2021.05.06 14:06:47.480 [E0000,W0000] OUTPUT  GetInfofield command [0x283500001d]
# 2021.05.06 14:06:47.540 [E0000,W0000] INPUT   GetInfofield command [0x5d0022753011240f]
```

4.2.3.1.13 GenerateCommandBytes()

Creates a new custom command that can be sent later with the *SendRaw* function.

Syntax

```
def GenerateCommandBytes( self
                        , aCommandCode       : int
                        , aParameters        : Union[None, TypeByteBlock]           = None
                        , aPort              : Union[None, MPBusPort]               = None
                        , aCommunicationType : Union[None, str, CommunicationTypes] = None
```

(continues on next page)

(continued from previous page)

```
                        , aAcceptErrors      : Union[None, List[int]]              = None
                        , aLabel             : Union[None, str]                    = None
                        ) -> List[int]:
```

Arguments

- **aCommandCode** : The code of the command to be created.
- **aParameters** : MP parameters.
- **aPort** : See *Common Function Parameters*. This parameter does not have an effect, use the default value.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*. This parameter has no effect.
- **aLabel** : See *Common Function Parameters*.

Return Value

Returns the raw command bytes including the start and parity bytes. If any error occurs during command generation, then it returns an empty list.

Example

```
(result, newCommand) = mpbus.GenerateCommandBytes(aCommandCode = 111, aParameters = [0x00, 0x11]);
# Log:
# 2021.05.04 13:57:23.789 [E0000,W0000] INFO    Command bytes generated [0x386F00111056]
(result, newCommand) = mpbus.GenerateCommandBytes( aCommandCode = 111
                                    , aParameters = [0x00, 0x11]
                                    , aCommunicationType = CommunicationTypes.MP1 );
# Log:
# 2021.05.04 13:57:23.789 [E0000,W0000] INFO    Command bytes generated [0x386F00111056]
```

4.2.3.1.14 GetMPAddress()

Sends the MP_Get_MP_Address (13) command.

```
def GetMPAddress( self
                , aPort              : Union[None, MPBusPort]               = None
                , aCommunicationType : Union[None, str, CommunicationTypes] = None
                , aAcceptErrors      : Union[None, List[int]]               = None
                , aLabel             : Union[None, str]                     = None
                ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*.

Example

```
(resultCode, address) = mpbus.GetMPAddress();
# Log
# 2021.05.15 15:10:05.570 [E0000,W0000] OUTPUT  GetMPAddress [0x180d8095]
# 2021.05.15 15:10:05.633 [E0000,W0000] INPUT   GetMPAddress [0x1d088095]
if len(address) > 0 :
    tester.Log(LogLevels.INFO, "MP address : " + str(address[0]));
# Log
# 2021.05.15 15:10:05.633 [E0000,W0000] INFO    MP address : 0x08
```

4.2.3.1.15 SetMPAddress()

Sends the MP_Set_MP_Address (38) command.

```
def SetMPAddress( self
                , aNewAddress        : int
                , aYear              : int
                , aWeek              : int
                , aWeekDay           : int
                , aRunNumber         : int
                , aTestStation       : int
                , aPort              : Union[None, MPBusPort]               = None
                , aCommunicationType : Union[None, str, CommunicationTypes] = None
                , aAcceptErrors      : Union[None, List[int]]               = None
                , aLabel             : Union[None, str]                     = None
                ) -> Tuple[int, List[int]]:
```

Arguments

- **aNewAddress** : New MP Address in the range [1..16]
- **aYear** : Last two digits of year. (e.g: 2016 -> 16)
- **aWeek** : Week of the year.
- **aWeekDay** : Number of day of week.
- **aTestStation** : Identifier of test station.
- **aRunNumber** : The run number for serial.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(error, buffer) = mpbus.SetMPAddress( aNewAddress  = CommunicationTypes.MP8
                                    , aYear        = 11
                                    , aWeek        = 23
                                    , aWeekDay     = 3
                                    , aRunNumber   = 2
                                    , aTestStation = 27);
# Log
# 2021.05.15 15:31:36.228 [E0000,W0000] OUTPUT  SetMPAddress [0x78260b1703021b00267e]
# 2021.05.15 15:31:36.320 [E0000,W0000] INPUT   SetMPAddress [0x0d808d]
```

4.2.3.1.16 GetState()

Sends the MP_Get_State (10) command.

```
def GetState( self
            , aPort              : Union[None, MPBusPort]               = None
            , aCommunicationType : Union[None, str, CommunicationTypes] = None
            , aAcceptErrors      : Union[None, List[int]]               = None
            , aLabel             : Union[None, str]                     = None
            ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*.

Example

```
(resultCode, state) = mpbus.GetState();
# Log
# 2021.05.18 12:07:41.163 [E0000,W0000] OUTPUT  GetState [0x180a0012]
# 2021.05.18 12:07:41.259 [E0000,W0000] INPUT   GetState [0x7d33449856001520c839]
```

4.2.3.1.17 SetForcedControl()

Sends the MP_Set_Forced_Control (14) command.

```
def SetForcedControl( self
                    , aForceOperation    : int
                    , aPort              : Union[None, MPBusPort]               = None
                    , aCommunicationType : Union[None, str, CommunicationTypes] = None
                    , aAcceptErrors      : Union[None, List[int]]               = None
                    , aLabel             : Union[None, str]                     = None
                    ) -> Tuple[int, List[int]]:
```

Arguments

- **aForceOperation** : Identifier of forced operation.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(error, buffer) = mpbus.SetForcedControl(aTestStation = 2);
# Log
# 2021.05.18 12:07:41.038 [E0000,W0000] OUTPUT  SetForcedControl [0x280e02c0e4]
# 2021.05.18 12:07:41.131 [E0000,W0000] INPUT   SetForcedControl [0x0d808d]
```

4.2.3.1.18 GetRelative()

Sends the MP_Get_Relative (41) command.

```
def GetRelative( self
               , aPort              : Union[None, MPBusPort]               = None
               , aCommunicationType : Union[None, str, CommunicationTypes] = None
               , aAcceptErrors      : Union[None, List[int]]               = None
               , aLabel             : Union[None, str]                     = None
               ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The length of the second value is 4 bytes long. The first two bytes contain the actual position and the second two bytes contain the setpoint.

Example

```
(resultCode, data) = mpbus.GetRelative();
# Log
# 2021.05.18 16:02:09.371 [E0000,W0000] OUTPUT  GetRelative [0x182980b1]
# 2021.05.18 16:02:09.437 [E0000,W0000] INPUT   GetRelative [0x4d131417181055]
if len(data) == 4 :
    tester.Log( LogLevels.INFO
              , "Actual    : " + str(tester.ToNumber( aType = ConvertTypes.UINT16
                                                    , aBuffer = data)));
    tester.Log( LogLevels.INFO
              , "Set Point : " + str(tester.ToNumber( aType = ConvertTypes.UINT16
                                                    , aBuffer = data
                                                    , aStartIndex = 2)));
# Log
# 2021.05.18 16:02:09.444 [E0000,W0000] INFO    Actual    : 4884
# 2021.05.18 16:02:09.446 [E0000,W0000] INFO    Set Point : 5912
```

4.2.3.1.19 SetRelative()

Sends the MP_Set_Relative (37) command.

```
def SetRelative( self
               , aValue             : int
               , aPort              : Union[None, MPBusPort]               = None
               , aCommunicationType : Union[None, str, CommunicationTypes] = None
               , aAcceptErrors      : Union[None, List[int]]               = None
               , aLabel             : Union[None, str]                     = None
               ) -> Tuple[int, List[int]]:
```

Arguments

- **aValue** : The value to set.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(error, data) = mpbus.SetRelative(aValue = 4500);
# Log
# 2021.05.18 16:02:09.243 [E0000,W0000] OUTPUT  SetRelative [0x38251194b028]
# 2021.05.18 16:02:09.340 [E0000,W0000] INPUT   SetRelative [0x0d808d]
```

4.2.3.1.20 Login()

Sends the MP_Login (78) command.

```
def Login( self
         , aPassword          : List[int]
         , aPort              : Union[None, MPBusPort]               = None
         , aCommunicationType : Union[None, str, CommunicationTypes] = None
         , aAcceptErrors      : Union[None, List[int]]               = None
         , aLabel             : Union[None, str]                     = None
         ) -> Tuple[int, List[int]]:
```

Arguments

- **aPassword** : 4 bytes long byte list as a password.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list. If the format of password is invalid it returns ERROR_PYT_ARGUMENT_ERROR.

Example

```
(result, data) = mpbus.Login(aPassword = [0xFB, 0x78, 0xFB, 0x79]);
# Log
# 2021.05.18 16:57:20.308 [E0000,W0000] OUTPUT  Login [0x584efb78fb79d4c3]
# 2021.05.18 16:57:20.401 [E0000,W0000] INPUT   Login [0x0d808d]
```

4.2.3.1.21 Logout()

Sends the MP_Logout (60) command.

```
def Logout( self
          , aPort              : Union[None, MPBusPort]              = None
          , aCommunicationType : Union[None, str, CommunicationTypes] = None
          , aAcceptErrors      : Union[None, List[int]]              = None
          , aLabel             : Union[None, str]                    = None
          ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(result, data) = mpbus.Logout();
# Log
# 2021.05.18 16:57:20.432 [E0000,W0000] OUTPUT  Logout [0x183c0024]
# 2021.05.18 16:57:20.527 [E0000,W0000] INPUT   Logout [0x0d808d]
```

4.2.3.1.22 SpecialFunction()

Sends the MP_SpecialFunction (102) command.

```
def SpecialFunction( self
                   , aFunction          : int
                   , aPort              : Union[None, MPBusPort]              = None
                   , aCommunicationType : Union[None, str, CommunicationTypes] = None
                   , aAcceptErrors      : Union[None, List[int]]              = None
                   , aLabel             : Union[None, str]                    = None
                   ) -> Tuple[int, List[int]]:
```

Arguments

- **aFunction** : Identifier number of the function.
- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(result, data) = mpbus.SpecialFunction(aFunction = 1);
# Log
# 2021.05.20 09:36:26.242 [E0000,W0000] OUTPUT  SpecialFunction [0x28660180cf]
# 2021.05.20 09:36:26.336 [E0000,W0000] INPUT   SpecialFunction [0x0d808d]
```

4.2.3.1.23 GetSummary()

Sends the MP_Get_Summary (118) command.

```
def GetSummary( self
              , aPort              : Union[None, MPBusPort]               = None
              , aCommunicationType : Union[None, str, CommunicationTypes] = None
              , aAcceptErrors      : Union[None, List[int]]               = None
              , aLabel             : Union[None, str]                     = None
              ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*.

Example

```
(resultCode, state) = mpbus.GetSummary();
if type(state) is list:
    bits = tester.BytesToBits(state[0]);
    if type(bits) is list:
        if bits[1]:
            tester.Log(LogLevels.INFO, "Data changed");
        else:
            tester.Log(LogLevels.INFO, "No data changed");
```

4.2.3.1.24 GetSeriesNo()

Sends the MP_Get_SeriesNo (50) command.

```
def GetSeriesNo( self
               , aPort              : Union[None, MPBusPort]               = None
               , aCommunicationType : Union[None, str, CommunicationTypes] = None
               , aAcceptErrors      : Union[None, List[int]]               = None
               , aLabel             : Union[None, str]                     = None
               ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aCommunicationType** : See *MP Command Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns according to the *Function Return Values*.

Example

```
(resultCode, seriesNo) = mpbus.GetSeriesNo();
# Log
# 2021.05.20 10:06:54.668 [E0000,W0000] OUTPUT  GetSeriesNo [0x183280aa]
# 2021.05.20 10:06:54.760 [E0000,W0000] INPUT   GetSeriesNo [0x7d565758595a5b5c4c6c]
if type(seriesNo) is list:
    tester.LogBytes(aLogLevel = LogLevels.INFO, aTextPrefix = "Series Number:", aBytes = seriesNo);
# 2021.05.20 10:06:54.780 [E0000,W0000] INFO    Series Number: [0x565758595a5b5c]a5b5c]
```

4.2.4 Class MPSerialPort

Class representation of an MP-Bus serial port, implements *MBPort*.

4.2.4.1 Properties

4.2.4.1.1 communication_type

The communication type used on this port.

Example

```
port.communication_type = CommunicationTypes.MP2;
# Log
# 2021.12.07 13:32:42.056 [E0000,W0000] DEBUG     'port0' - 'MP1' -> 'MP2'
port.communication_type = 33;
# Log
# 2021.12.07 15:32:42.058 [E0001,W0000] ERROR     'port0' - Invalid communication type '33'
```

4.2.4.1.2 retry_count

Number of retries used in MP communication. The property is read-only and is set by the constructor argument.

4.2.4.2 Methods

4.2.4.2.1 Constructor

Syntax

Arguments

- **aTester** : An MBTester object.
- **aName** : Identifier name for the port.
- **aDeviceId** : Serial port identifier string. (example: “COM1”, “COM2”, etc.)
- **aCommunicationType** : See *MP Command Parameters*
- **aRtsLevel** : Set the RTS (Request To Send) level to high (True) or low (False). Simulation of bad serial communication in case of dev board.

- **aDtrLevel** : Set the DTR (Data Terminal Ready) level to high (True) or low (False). Simulation of bad serial communication in case of dev board.
- **aBaudRate** : The baud rate of the serial port. This value will be registered for the "aDeviceId". Changing this value for the device is not possible by creating an another port. It can cause an error. The following baud rates are allowed: 1200, 9600, 38400. The default value is 1200. (See *the constants of usable baudrates*)
- **aRetryCount** : The retry count during the communication. (default: 3)

Example

```
tester = MBTester();
bus1 = MPSerialPort( aTester            = tester
                   , aName              = "Port1"
                   , aDeviceId          = "COM1"
                   , aCommunicationType = CommunicationTypes.MP1
                   , aBaudRate          = MPBaudRates.B38400 );
# Log:
# 2021.02.11 13:57:23.789 INFO    MP Port created ['Port1',0]
bus2 = MPSerialPort( aTester            = tester
                   , aName              = "Port2"
                   , aDeviceId          = "COM1"
                   , aCommunicationType = CommunicationTypes.MP2);
# Log:
# 2021.02.11 13:57:23.790 FATAL_ERR Could not create port 'Port1' due to already defined.
bus3 = MPSerialPort( aTester            = tester
                   , aName              = "Port1"
                   , aDeviceId          = "COM1"
                   , aCommunicationType = CommunicationTypes.MP3 );
# Log:
# 2021.02.11 13:57:23.790 FATAL_ERR Could not create port 'Port1' due to already defined.
bus4 = MPSerialPort( aTester            = tester
                   , aName              = "Port3"
                   , aDeviceId          = "COM1"
                   , aCommunicationType = CommunicationTypes.MP4
                   , aBaudRate          = MPBaudRates.BAUD_RATE_1200 );
# Log:
# 2021.02.11 13:57:23.795 ERROR     Could not initialize serial port because it is already initialized ...
# ...with different baud rate ['COM1',1200/38400]
```

4.3 BNP (Belimo NFC Protocol)

4.3.1 Introdution

To send MP commands over BNP the same MPBus class is used as for the regular MP communication, the difference is that it is initialized with a BNPPort instead of an MP serial port.

4.3.1.1 Initialisation

```
from mb_tester.MBTester import MBTester, LogLevels;
from mb_tester.NFCPort  import USBFeigReaderPort, GetFirstTag;
from mb_tester.NFCTag   import NXP_NT3H2211_Tag;
from mb_tester.BNP      import BNPPort;
from mb_tester.MPBus    import MPBus;

tester    = MBTester();
nfc_port  = USBFeigReaderPort( aTester = tester
                             , aName = "Reader" );
```

(continues on next page)

(continued from previous page)

```
nfc_port.InitReader();

tags = nfc_port.GetDetectedTagList();
if (len(tags) == 0):
    tester.FatalError("No tags were detected.");

tag = BNP_NXP_NT3H2211_Tag(
    aTester = tester
    , aTagId = tags[0]['tag_id']
    , aProtocolVersion = ProtocolVersion.BNPV3 );

bnp_port  = BNPPort( aTester  = tester
                   , aName    = "device1"
                   , aNfcPort = nfc_port
                   , aTag     = tag );
mpbus     = MPBus( aTester      = tester
                 , aDefaultPort = bnp_port );

# test code starts from here
address = mpbus.GetMPAddress();
```

4.3.1.2 Warnings

- For BNPV1-3: The current version does not support the "answer delayed" functionality of the BNP protocol. This is currently also not supported in the SWAP implementations.

4.3.2 Types

These are the types of configurations related to BNP.

4.3.2.1 ProtocolVersion

Import : 'from mb_tester.BNPPort import ProtocolVersion'

```
class ProtocolVersion(IntEnum):
"""The Belimo NFC Protocol Versions"""

    BNPV1 : int = 1;
    """Unsupported"""
    BNPV2 : int = 2;
    """Unsupported"""
    BNPV3 : int = 3;
    """Supported"""
```

4.3.3 Class BNPPort

Class representation of a BNP port, implements *MBPort*.

4.3.3.1 Properties

4.3.3.1.1 reader

The used reader port.

```
@property
def reader(self) -> NFCReaderPort:
```

4.3.3.1.2 tag

The used tag.

```
@property
def tag(self) -> NFCTagBase:
```

4.3.3.1.3 communication_type

The used communication type.

```
@property
def communication_type(self) -> CommunicationTypes:
```

4.3.3.2 Methods

4.3.3.2.1 Constructor

```
def __init__( self
            , aTester             : MBTester
            , aNfcPort            : NFCReaderPort
            , aTag                : NFCTagBase
            , aCommunicationTypes : CommunicationTypes = CommunicationTypes.PP ):
```

Arguments

- **aTester** : The MB Tester object.
- **aNfcPort** : The used port for the NFC reader device.
- **aTag** : The tag of the device.
- **aCommunicationType** : The used communication type. (default: 'CommunicationTypes.PP')

Example

```
## create port with the exact protocol version
bnp = BNPPort( aTester          = tester
             , aNfcPort         = nfc_port
             , aTag             = tag );
```

4.3.4 Tags

These classes are based on the *NFC Tags*.

4.3.4.1 Available classes for tags

Class	Type of Tag
BNP_NXP_Tag	ISO14443A - Generic NXP Tag
BNP_NXP_NT3H2111_Tag	ISO14443A - NXP NT3H2111
BNP_NXP_NT3H2211_Tag	ISO14443A - NXP NT3H2211

4.3.4.2 Properties

It inherits properties from the *NFC Tags* .

4.3.4.2.1 protocol_version

The used protocol version. (read only)

```
@property
def protocol_version( self ) -> int:
```

4.3.4.3 Methods

4.3.4.3.1 Constructor

```
def __init__( self
            , aTester          : MBTester
            , aTagId           : str
            , aProtocolVersion : ProtocolVersion
            , aTag             : Union[None, NXPGenericTag] ):
```

Arguments

- **aTester** : An [MBTester](../MBTester/Class_MBTester.md#class mbtester) object.
- **aTagId** : The hexadecimal IDD identifier of the label. This string is exactly the same as the ID in the return value of the *GetDetectedTagList()* function.
- **aProtocolVersion** : The used protocol version during the BNP communication.
- **aTag** : The NFC tag object, if it is already created for simple NFC communication. (This argument is only available in the BNP_NXP_Tag constructor.)

Example

```
from MBTester import MBTester, LogLevels;
from NFCPort  import USBFeigReaderPort;
from BNPTag   import BNP_NXP_NT3H2211_Tag;
from BNPPort  import ProtocolVersion;

tester = MBTester();
reader = USBFeigReaderPort( aTester = tester, aDeviceId = None, aName = "USBReader");
reader.InitReader();
```

(continues on next page)

(continued from previous page)

```
tags   = port.GetDetectedTagList();
if (len(tags) == 0):
    tester.Fatalerror("No tag were detected.");

tag    = BNP_NXP_NT3H2211_Tag( aTester          = tester
                             , aTagId           = tags[0]['tag_id']
                             , aProtocolVersion = ProtocolVersion.BNPV3 );
```

4.4 NFC

4.4.1 Introduction

Implements the communication with NFC tags.

4.4.1.1 Hardware support

Supported readers:

- Feig reader

Supported devices:

- NXP chips

4.4.1.2 NFC related function parameters

These parameters are only used in communication functions, for example: *ReadTag*, *WriteTag*, etc.

- **aTag** : The targeted NFC tag. This will override the default NFC tag for the current function.

4.4.1.3 Initialization

```
from mb_tester.MBTester import MBTester;
from mb_tester.NFC      import NFC;
from mb_tester.NFCPort  import USBFeigReaderPort;
from mb_tester.NFCTag   import ISO14443ATag;

# An MBTester object is required
tester = MBTester(...);

# An NFC port is optional
port = USBFeigReaderPort(...);

# An NFC tag is optional
tag = ISO14443ATag(...);

# Create an NFC instance
nfc = NFC( aTester = tester, aDefaultPort = port, aDefaultTag = tag )
```

4.4.1.4 Tag handling

If no default tag is set for the NFC class and not tag set for the command the first available tag in the field is used. The reader may return any tag in the field as the first tag and it may change between each communication. If you expect to have several tags in the field it is best to explicitely select one for the communication.

4.4.2 Types

These are the types of configurations related to NFC.

4.4.3 Global

4.4.3.1 GetReaderList

This function queries the list of available readers.

Note: It does not work for serial FEIG readers because they do not have an identifier. The serial FEIG readers are identified by their COM ports.

This function is usable even without creating a port.

```
def GetReaderList( aTester        : MBTester
                 , aUnusedReaders : bool             = True
                 , aLabel         : Union[None, str] = None ) -> Union[None, List[str]]:
```

Arguments

- **aTester** : The MBTester object to be used.
- **aUnusedReaders** : If 'True', the function only returns unused readers, if 'False', it returns unused and used readers too.
- **aLabel** : See *Common Function Parameters*.

Return value

Returns 'None' if no readers are detected. If it can detect readers, it returns the following struct:

```
{
    <Reader Type 1> : [ <Reader id 1>, <Reader id 1>, ..., <Reader id n> ],
    <Reader Type 2> : [...],
    ...
    <Reader Type n> : [...]
}

Example:
{
    READER_TYPE_FEIG_USB : [ "487778044", "487778047", ... ],
    ...
}
```

Example

```
from mb_tester.NFC import GetReaderList;

tester  = MBTester( ... ;)
readers = GetReaderList( aTester = tester );
# Log:
# 2021.07.14 11:02:00.000 [E0000,W0000] INFO    READER_TYPE_FEIG_USB : [ "463668459", "463668470" ]
```

(continues on next page)

(continued from previous page)

```
# readers
# { READER_TYPE_FEIG_USB : [ "463668459", "463668470" ] }
```

4.4.3.2 GetFirstTag

Create a tag object based on the first available tag id, and the given tag object type.

```
def GetFirstTag( aTester     : MBTester
               , aReaderPort : NFCReaderPort
               , aTagObjectType
               , aLabel      : Union[None, str] = None ):
```

Arguments

- **aTester** : The MBTester object to be used.
- **aReaderPort** : The used reader port.
- **aTagObjectType** : The type of the tag handler object.
- **aLabel** : See *Common Function Parameters*.

Return value

Returns 'None' if no tags are detected, otherwise returns an object based on the given type (aTagObjectType) for tag handling.

Example

```
from mb_tester.MBTester import MBTester;
from mb_tester.NFC      import USBFeigReaderPort, GetFirstTag;
from mb_tester.NFCTag   import NXP_NT3H2211_Tag;

tester = MBTester();
port   = USBFeigReaderPort( aTester = tester );

# tag found
tag    = GetFirstTag( aTester        = tester
                    , aReaderPort    = port
                    , aTagObjectType = NXP_NT3H2211_Tag );
# Log:
# 2022.07.16 16:30:00.000 [E0000,W0000] INFO      Tag created [(...)]

tag    = GetFirstTag( aTester        = tester
                    , aReaderPort    = port
                    , aTagObjectType = NXP_NT3H2211_Tag );
# Log:
# 2022.07.16 16:30:00.001 [E0000,W0000] INFO      Tag not found!
```

4.4.4 Class NFC

4.4.4.1 Properties

4.4.4.1.1 default_port

The default port for the NFC communication. If the value is 'None', then the aPort attribute has to be added for all commands.

```
@property
def default_port(self):
```

Example

```
tester = MBTester();
nfc = NFC(tester);
nfc.default_port = USBfeigReaderPort( aTester = tester );
nfc.default_port = None;
```

4.4.4.1.2 default_tag

The default tag for the NFC communication. If this is not set and not specified as an attribute in the commands, the first tag available in the field will be used.

```
@property
def default_tag(self):
```

Example

```
tester = MBTester();
nfc = NFC(tester);
nfc.default_tag = NXPGenericTag( aTester = tester, aTagId = "1234");
nfc.default_tag = None;
```

4.4.4.2 Methods

4.4.4.2.1 Constructor

```
def __init__( self
            , aTester      : MBTester
            , aDefaultPort : Union[None, NFCReaderPort] = None
            , aDefaultTag  : Union[None, NFCTagBase]    = None ):
```

Arguments

- **aTester:** The MBTester object to be used.
- **aDefaultPort:** The NFCReaderPort object to be used by default. If the value is 'None', then the aPort attribute has to be added for all commands. It can be added later via the *default_port* property.
- **aDefaultTag:** The NFCTagBase object to be accessed by default. If this is not set and not specified as an attribute in the commands, the first tag available in the field will be used. It can be added later via the *default_tag* property.

Example

```
from mb_tester.MBTester import MBTester;
from mb_tester.NFC      import NFC;
from mb_tester.NFCPort  import USBFeigReaderPort;
from mb_tester.NFCTag   import NXP_NT3H2111_Tag;

tester = MBTester();
port   = USBFeigReaderPort( aTester = tester
                          , aName = "Port1"
                          , aIsDefault = True );
tag    = NXP_NT3H2111_Tag( aTester = tester
                         , aTagId  = "1234" );
nfc    = NFC( aTester      = tester
```

(continues on next page)

(continued from previous page)

```
            , aDefaultPort = port
            , aDefaultTag  = tag );
```

4.4.4.2.2 ReadTag()

This function reads and returns the memory of the tag.

```
def ReadTag( self
           , aBlockAddress : int
           , aLength       : int
           , aPort         : Union[None, NFCReaderPort] = None
           , aTag          : Union[None, NFCTagBase]    = None
           , aAcceptErrors : Union[None, List[int]]     = None
           , aLabel        : Union[None, str]           = None
           ) -> Tuple[int, List[int]]:
```

Arguments

- **aBlockAddress:** The first data block's address is in memory of the tag from where the reading starts.
- **aLength:** The count of bytes to read.
- **aPort:** See *Common Function Parameters*.
- **aTag:** See *NFC related function parameters*.
- **aAcceptErrors:** See *Common Function Parameters*.
- **aLabel:** See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
tester           = MBTester()
port             = USBFeigReaderPort(aTester = tester, aName = "FEIG Reader 1", aDeviceId = "487778044");
port.InitReader();
port.default_tag = NXPGenericTag( aTester = tester, aTagId = "008055CCDA347504");
nfc              = Nfc(aTester = tester, aDefaultPort = port);

(result, buffer) = nfc.ReadTag(aBlockAddress = 0x13, aLength = 5);
# Log:
# 2021.08.08 06:00:00.001 [E0000,W0000] INPUT   ReadTag [0x0123456789]
if result == 0:
   # buffer contains data
else :
   # buffer is empty
```

4.4.4.2.3 WriteTag()

This function writes to the memory of the NFC tag.

Please take extra care to ensure that the input data size is a multiple of the NFC Tag block size. It is recommended that you first read the blocks you want to modify using the ReadTag function, and then modify and write them back using this function.

If the size of the input data is not as expected, then it causes error.

```
def WriteTag( self
            , aBlockAddress : int
            , aData         : List[int]
            , aPort         : Union[None, NFCReaderPort] = None
            , aTag          : Union[None, NFCTagBase]    = None
            , aAcceptErrors : Union[None, List[int]]     = None
            , aLabel        : Union[None, str]           = None
            ) -> Tuple[int, List[int]]:
```

Arguments

- **aBlockAddress:** The address of the first data block in memory of the NFC tag to start the writing from.
- **aData** : The data to write.
- **aPort** : See *Common Function Parameters*.
- **aTag** : See *NFC related function parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
tester           = MBtester();
port             = USBFeigReaderPort(aTester = tester, aName = "FEIG Reader 1", aDeviceId = "487778044");
port.InitReader();
port.default_tag = NXPGenericTag(aTester = tester, aTagId = "008055CCDA347504");
nfc              = NFC(aTester = tester, aDefaultPort = port);

(result, data) = nfc.WriteTag(aBlockAddress = 0x13, aData = [0x03, 0x05, 0x07, 0x08]);
# Log:
# 2021.08.08 06:00:00.001 [E0000,W0000] OUTPUT  WriteTag [0x03050708]
if result == 0:
   # the write is done
else :
   # the write failed
```

4.4.4.2.4 SendRawReaderCommand()

Sends a raw command to the reader.

```
def SendRawReaderCommand( self
                        , aRequest        : list
                        , aTag            : Union[None, NFCTagBase]    = None
                        , aPort           : Union[None, NFCReaderPort] = None
                        , aAcceptErrors   : Union[None, List[int]]     = None
                        , aLabel          : Union[None, str]           = None
                        ) -> Tuple[int, List[int]]:
```

Arguments

- **aRequest** : The parameters of the request. This byte list has to contain at least 1 byte.
- **aPort** : See *Common Function Parameters*.
- **aTag** : See *NFC related function parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.

- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(result, buffer) = nfc.SendRawReaderCommand(
    aRequest = [0xbd, 0x01, 0x00, 0xff, 0x10, 0x0d, 0xC2, 0xFF]
);
# Log:
# 2022.10.26 14:26:02.715 [E0004,W0000] INPUT     SendRawReaderCommand [0xbd0100ff100dc2ff]
# 2022.10.26 14:26:02.717 [E0004,W0000] OUTPUT    SendRawReaderCommand [0xbd020a]
# 2022.10.26 14:26:02.718 [E0004,W0000] ERROR     SendRawReaderCommand (2151) Status byte (0x02) Data False
# returns:
#   result      : 0
#   buffer      : [ 0xbd, 0x02, 0x0a ]

(result, buffer) = nfc.SendRawReaderCommand(
    aRequest        = [0xbd, 0x01, 0x00, 0xff, 0x10, 0x0d, 0xC2, 0xFF, 0x01, 0x00, 0x00, 0x00]
);
# Log:
# 2022.10.26 14:26:02.722 [E0004,W0000] INPUT     SendRawReaderCommand [0xbd0100ff100d01000000]
# 2022.10.26 14:26:02.743 [E0004,W0000] OUTPUT    SendRawReaderCommand [0xbd01]
# 2022.10.26 14:26:02.744 [E0004,W0000] ERROR     SendRawReaderCommand (2150) Status byte (0x01) No Transponder
# returns:
#   result      : 0
#   buffer      : [ 0xbd, 0x01 ]
```

4.4.4.2.5 SendTransparentCommand()

Sends a ISO14443A transparent command (0xBD).

```
def SendTransparentCommand( self
                          , aRequest        : list
                          , aResponseLength : int
                          , aPort           : Union[None, NFCReaderPort] = None
                          , aTag            : Union[None, NFCTagBase]    = None
                          , aAcceptErrors   : Union[None, List[int]]     = None
                          , aLabel          : Union[None, str]           = None
                          ) -> Tuple[int, List[int]]:
```

Arguments

- **aRequest** : Complete transponder request without SOF and EOF.
- **aResponseLength** : If length is set to “0” the Reader will send the command but not wait for any response. If length is not equal to “0” the Reader will send the command and return the response data of the Transponder without SOF and EOF
- **aPort** : See *Common Function Parameters*.
- **aTag** : See *NFC related function parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(result, buffer) = nfc.SendRawNFCCommand( aRequest = [0xC2, 0xFF], aResponseLength = 0xff );
# Log:
# 2022.10.26 14:26:02.715 [E0004,W0000] INPUT     SendTransparentCommand [0xbd0100ff100dc2ff]
# 2022.10.26 14:26:02.717 [E0004,W0000] OUTPUT    SendTransparentCommand [0xbd020a]
# 2022.10.26 14:26:02.718 [E0004,W0000] ERROR     SendTransparentCommand (2151) Status byte (0x02) Data False
# returns:
#   result      : 2151
#   buffer      : [ 0x0a ]

(result, buffer) = nfc.SendTransparentCommand( aRequest        = [0x01, 0x00, 0x00, 0x00]
                                             , aResponseLength = 0xff );
# Log:
# 2022.10.26 14:26:02.722 [E0004,W0000] INPUT     SendTransparentCommand [0xbd0100ff100d01000000]
# 2022.10.26 14:26:02.743 [E0004,W0000] OUTPUT    SendTransparentCommand [0xbd01]
# 2022.10.26 14:26:02.744 [E0004,W0000] ERROR     SendTransparentCommand (2150) Status byte (0x01) No␣
↪Transponder
# returns:
#   result      : 2150
#   buffer      : [ ]
```

4.4.4.2.6 ReadSRAM()

This function is avaliable in case of NXP Chips for passthrough communication. The function reads the session registers (ECh) and checks the set of the following bits before reading the SRAM:

- NC_REG.PTHRU_ON_OFF (1)
- NS_REG.RF_LOCKED (1)
- NS_REG.SRAM_RF_READY (1)

There is a timeout of 1s, if the registers do not have the correct value, the function fails.

```
def ReadSRAM( self
            , aPort           : Union[None, NFCReaderPort] = None
            , aTag            : Union[None, NFCTagBase]    = None
            , aAcceptErrors   : Union[None, List[int]]     = None
            , aLabel          : Union[None, str]           = None
            ) -> Tuple[int, List[int]]:
```

Arguments

- **aPort** : See *Common Function Parameters*.
- **aTag** : See *NFC related function parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(result, data) = nfc.ReadSRAM();
# 2022.10.26 14:44:37.655 [E0000,W0000] INFO      Checking of the Session Registers... (CSR)
# 2022.10.26 14:44:37.656 [E0000,W0000] INPUT     ReadSRAM [0x7d000014808012100]
# ...
# 2022.10.26 14:44:37.695 [E0000,W0000] INPUT     ReadSRAM [0x0000000000000000000000000000000000000000...
# ...0000000000000000000000000000000000000000000000000000000000000000c037d0000000000000007212000000000]
# returns:
#    result : 0
```

(continues on next page)

(continued from previous page)

```
#    data : [ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
#             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
#             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
#             0x0c, 0x03, 0x7d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x72, 0x12, 0x00, 0x00, 0x00, 0x00 ]
```

4.4.4.2.7 WriteSRAM()

This function is avaliable in case of NXP Chips. The function reads the session registers (ECh) and checks the set of the following bits before writing the SRAM:

- NC_REG.PTHRU_ON_OFF (1)
- NS_REG.RF_LOCKED (1)
- NS_REG.SRAM_RF_READY (0)

There is a timeout of 1s, if the registers do not have the correct value, the function fails.

```
def WriteSRAM( self
             , aData           : List[int]
             , aPort           : Union[None, NFCReaderPort] = None
             , aTag            : Union[None, NFCTagBase]    = None
             , aAcceptErrors   : Union[None, List[int]]     = None
             , aLabel          : Union[None, str]           = None
             ) -> Tuple[int, List[int]]:
```

Arguments

- **aData** : The data to write. It must be exactly 64 bytes long.
- **aPort** : See *Common Function Parameters*.
- **aTag** : See *NFC related function parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
raw_get_series_no = [
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x06, 0x01, 0x11, 0x32, 0xcd, 0xdc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
]
(result, data) = nfc.WriteSRAM( aData = raw_get_series_no );
# 2022.10.26 14:40:56.169 [E0000,W0000] INFO      Checking of the Session Registers... (CSR)
# 2022.10.26 14:40:56.170 [E0000,W0000] INPUT     WriteSRAM [0x7d000148080121000]
# ...
# 2022.10.26 14:40:56.196 [E0000,W0000] OUTPUT    WriteSRAM [0x0000000000000000000000000000000000000000...
# ...000000000000000000000000000000000000000000000000000000000000000000000000000000006011132cddc0000000000000000000000]
# returns:
#    result : 1
#    data   : [ ]
```

4.4.5 Reader Ports

Each port is an interface to a reader. All ports have some common defined functions and properties.

4.4.5.1 Available classes for readers

Class	Type of Reader
SerialFeigReaderPort	Serial FEIG Reader
USBFeigReaderPort	USB FEIG Reader

4.4.5.1.1 Methods

Destructor

Closes the communication.

GetDetectedTagList()

Get the list of available tags in the NFC field. A connected and initialized reader is required.

```
def GetDetectedTagList( self
                      , aAcceptErrors : Union[None, List[int]] = None
                      , aLabel        : Union[None, str]       = None ) -> List[str]:
```

Arguments

- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return Value

It returns an empty list if no tags were detected, or a list of the identifiers of the detected tags.

Example

```
port = USBFeigReaderPort(aName = "FEIG Reader 1", aDeviceId = "487778044");
port.InitReader();
tags = port.GetDetectedTagList();
# Log:
# 2021.08.25 18:16:53.456 [E0000,W0000] INFO    1. found tag: {'tag_id': '008055CCDA347504', ...
# ...'tag_type': 'ISO 14443 Type A', 'tag_data': 'NXP Semiconductors'}
# 2021.08.25 18:16:53.457 [E0000,W0000] INFO    2. found tag: {'tag_id': '00805706729C9F04', ...
# ...'tag_type': 'ISO 14443 Type A', 'tag_data': 'NXP Semiconductors'}
# tags
# { {'tag_id': '008055CCDA347504', 'tag_type': 'ISO 14443 Type A', 'tag_data': 'NXP Semiconductors'}
# , {'tag_id': '00805706729C9F04', 'tag_type': 'ISO 14443 Type A', 'tag_data': 'NXP Semiconductors'} }
```

InitReader()

This function initializes the reader for usage. It turns on the RF field of the reader.

This function is not possible to use before the creation of the port.

```
def InitReader( self
              , aAcceptErrors : Union[None, List[int]] = None
              , aLabel        : Union[None, str]       = None ) -> bool:
```

Arguments

- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

Returns 'False' in case of error, otherwise 'True'.

Example

```
port = USBFeigReader(aName = "FEIG Reader 1", aDeviceId = "487778044");
port.InitReader();
# Log:
# 2021.06.14 10:17:00.000 [E0000,W0000] INFO    Reader inited : (487778044)
port.InitReader();
# Log:
# 2021.06.14 10:17:00.024 [E0001,W0000] ERROR   <dll-error> InitReader : Init reader failed!...
# ... (2036) Error in Module FEDM: Reader object is already connected with a communication port (-137)
```

DeinitReader()

Releases the reader. It turns off the RF field of the reader.

A reader has to be initialized by *InitReader()* before calling this function.

```
def DeinitReader( self
                , aAcceptErrors : Union[None, List[int]] = None
                , aLabel        : Union[None, str]       = None ) -> bool:
```

Arguments

- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

Returns 'False' in case of error, otherwise 'True'.

Example

```
port = USBFeigReader(aName = "FEIG Reader 1", aDeviceId = "487778044");
port.InitReader();
# Log:
# 2021.06.14 10:17:00.000 [E0000,W0000] INFO    Reader inited : (487778044)
port.DeinitReader();
# Log:
# 2021.06.14 10:17:01.026 [E0000,W0000] INFO    Reader exited (487778044)
```

ReinitReader()

Reinitializes a reader by calling *DeinitReader()* and *InitReader()* with a configurable delay in between.

```
def ReinitReader( self
                , aDelayMs      : int
                , aAcceptErrors : Union[None, List[int]] = None
                , aLabel        : Union[None, str]       = None ) -> bool:
```

Arguments

- **aDelayMS** : The delay in milliseconds between exiting and initializing the reader.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

Returns 'False' in case of error, otherwise 'True'.

Example

```
port = USBFeigReader(aName = "FEIG Reader 1", aDeviceId = "487778044");
port.InitReader();
# Log:
# 2021.06.14 10:17:00.000 [E0000,W0000] INFO    Reader inited : (487778044)
port.ReinitReader(aDelayMs = 500);
# Log:
# 2021.06.14 10:17:00.040 [E0000,W0000] INFO    Reader exited (487778044)
# 2021.06.14 10:17:00.540 [E0000,W0000] INFO    Reader inited (487778044)
```

4.4.5.2 Class USBFeigReaderPort

Class representation of a Feig reader port over USB communication, implements *MBPort*.

4.4.5.2.1 Methods

Constructor

Opens the communication.

```
def __init__( self,
            , aTester    : MBTester
            , aName      : Union[None, str]
            , aDeviceId  : str
            , aIsDefault : bool ):
```

Arguments

- **aTester** : The MBTester object to be used.
- **aName** : A string that will be used as the identifier of the port.
- **aDeviceId** : The device identifier of the reader.
- **aIsDefault** : If 'True', this will be set as the default port, if 'False', it will not be.

Example

```
tester = MBTester();
port1 = USBFeigPort(tester, aName = "FEIG Reader 1");
# 2022.05.11 09:57:23.789 [E0000,W0000] INFO    NFC Port created ['FEIG Reader 1',0]
port2 = USBFeigPort(tester, aName = "FEIG Reader 1", aDeviceId = "463668459", aIsDefault = True );
# Log:
# 2022.05.11 09:57:23.790 [E0000,W0000] FATAL_ERR Could not create default port 'FEIG Reader 1' ...
# ...due to already defined

# If there is no reader connected and the device id is not given:
port3 = USBFeigPort(tester, aName = "FEIG Reader 2");
# Log:
# 2021.08.30 18:40:30.792 [E0000,W0000] FATAL_ERR Could not init 'FEIG Reader 2' port!

del port1
# Log:
# 2022.05.11 10:06:23.078 [E0000,W0000] INFO    NFC Port deleted ['FEIG Reader 1']
```

4.4.5.3 Class SerialFeigReaderPort

Class representation of a Feig reader port over serial communication, implements *MBPort*.

4.4.5.3.1 Methods

Constructor

Opens the communication.

```
def __init__( self,
            , aTester    : MBTester
            , aName      : str
            , aDeviceId  : str
            , aIsDefault : bool ):
```

Arguments

- **aTester** : The MBTester object to be used.
- **aName** : A string that will be used as the identifier of the port.
- **aDeviceId** : Mandatory. The name of the serial COM port. (example: “COM1”)
- **aIsDefault** : If ‘True’, this will be set as the default port, if ‘False’, it will not be.

4.4.6 Tags

These classes are the descriptors of the connected tags.

4.4.6.1 Available classes for tags

Class	Type of Tag	Default Block Size
ISO14443ATag	ISO14443A	None
NXPGenericTag	ISO14443A	None
NXP_NT3H2111_Tag	ISO14443A	4
NXP_NT3H2211_Tag	ISO14443A	4

4.4.6.2 Generic tags

4.4.6.2.1 Properties

tag_id

All classes have this read-only property to get the TAG identifier.

```
@property
def tag_id( self ) -> str:
```

block_size

The block size of the tag.

```
@property
def block_size( self ) -> Union[None, int]:
```

4.4.6.2.2 Methods

Constructor

```
def __init__( self
            , aTester    : MBTester
            , aTagId     : str
            , aBlockSize : Union[None, int] = None ):
```

Arguments

- **aTester** : An [MBTester](../MBTester/Class_MBTester.md#class mbtester) object.
- **aTagId** : The hexadecimal IDD identifier of the label. This string is exactly the same as the ID in the return value of the *GetDetectedTagList()* function.
- **aBlockSize** : The block size used. If 'None', it is identified by reading a block the first time the port is used.

Example

```
from MBTester import MBTester, LogLevels, ConvertTypes;
from NFC      import NFC, NFCDllReturns, GetFirstTag;
from NFCPort  import USBFeigReaderPort;
from NFCTag   import NXP_NT3H2211_Tag;

tester   = MBTester();
nfc_port = USBFeigReaderPort(tester, "FEIG Reader 1", None);
```

(continues on next page)

(continued from previous page)

```
retVal   = nfc_port.InitReader();
if retVal:
    tester.Log(LogLevels.INFO, "Init reader: FEIG Reader 1 was successful.");
else:
    tester.FatalError("Init reader: FEIG Reader 1 was unsuccessful.");

# Get the available tags if it is needed
tags = nfc_port.GetDetectedTagList();
if (len(tags) == 0):
    tester.FatalError("No tag were detected.");

# Create tag directly from a selected tag list entry
nfc_tag = NXP_NT3H2211_Tag(tester, tags[0]['tag_id']);
```

4.4.6.3 NXP_NT3H2211_Tag

4.4.6.3.1 Methods

SectorSelect

Select the NXP sector.

```
def SectorSelect( self
                , aSectorId       : [int, SectorIdentifiers]
                , aPort           : NFCReaderPort
                , aLabel          : Union[None, str] = None ) -> Tuple[int, List[int]]:
```

Arguments

- **aSectorId** : The sector identifier. Accepted values a
- **aPort** : The port identifier.
- **aLabel** : The used label in the log entries.

Return value

It returns according to the *Function Return Values*. The second value is always an empty list.

4.5 Modbus

4.5.1 Introduction

Implements Modbus communication.

4.5.1.1 Modbus related function parameters

These parameters are only used in communication functions, for example: *ReadCoils()*, *WriteCoil()*, *WriteCoils()*, etc.

- **aSlaveAddress:** The address of the slave. The value has to be between 0 and 247. This will override the default slave address for once in the function call.

4.5.2 Types

These are the types of configurations related to Modbus.

4.5.2.1 ModbusBaudRates

These are the available baud rates.

import : 'from mb_tester.Modbus import ModbusBaudRates'

```
class ModbusBaudRates(IntEnum):
    """The supported baud rates."""
    B1200   = 1200;
    B1800   = 1800;
    B2400   = 2400;
    B4800   = 4800;
    B9600   = 9600;
    B19200  = 19200;
    B28800  = 28800;
    B38400  = 38400;
    B57600  = 57600;
    B76800  = 76800;
    B115200 = 115200;
    B230400 = 230400;
    B460800 = 460800;
    B576000 = 576000;
    B921600 = 921600;
```

4.5.2.2 ModbusParities

These are the available parity settings.

import : 'from mb_tester.Modbus import ModbusParities'

```
class ModbusParities(IntEnum):
    """The available parities"""
    NONE = 0;
    ODD  = 1;
    EVEN = 2;
```

4.5.2.3 ModbusTransmissionTypes

These are the available transmission settings.

import : 'from mb_tester.Modbus import ModbusTransmissionTypes'

```
class ModbusTransmissionTypes(IntEnum):
    """The available transmissin types"""
    RTU   = 0;
    ASCII = 1;
```

4.5.2.4 ModbusDiagnosticsSubCommands

import : 'from mb_tester.Modbus import ModbusDiagnosticsSubCommands'

```
class ModbusDiagnosticsSubCommands(IntEnum):
    """Diagnostics sub commands"""
    DIAG_SUB_RETURN_QUERY_DATA                      = 0x00;
    DIAG_SUB_RESTART_COMMUNICATIONS_OPTION          = 0x01;
    DIAG_SUB_RETURN_DIAGNOSTICS_REGISTER            = 0x02;
    DIAG_SUB_FORCE_LISTEN_ONLY_MODE                 = 0x04;
    DIAG_SUB_CLEAR_COUNTERS_AND_DIAGNOSTIC_REGISTERS = 0x0a;
    DIAG_SUB_RETURN_BUS_MESSAGE_COUNT               = 0x0b;
    DIAG_SUB_BUS_COMMUNICATION_ERROR_COUNT          = 0x0c;
    DIAG_SUB_RETURN_BUS_EXCEPTION_COUNT             = 0x0d;
    DIAG_SUB_RETURN_SLAVE_MESSAGE_COUNT             = 0x0e;
    DIAG_SUB_RETURN_SLAVE_NO_RESPONSE_COUNT         = 0x0f;
    DIAG_SUB_RETURN_SLAVE_BUSY_COUNT                = 0x11;
```

4.5.2.5 ModbusProtocolConstants

import : 'from mb_tester.Modbus import ModbusProtocolConstants'

```
class ModbusProtocolConstants:
    """Constants for command generation"""
    ASCII_PROTOCOL_START_BYTES  = [ 0x3a ];
    ASCII_PROTOCOL_END_BYTES    = [ 0x0d, 0x0a ];
    TCP_PROTOCOL_TRANSACTION_ID = 0;
    TCP_PROTOCOL_PROTOCOL_ID    = 0;
```

4.5.2.6 ModbusFunctionCodes

import : 'from mb_tester.Modbus import ModbusFunctionCodes'

```
class ModbusFunctionCodes(IntEnum):
    # Function codes
    FC_READ_COIL_STATUS         = 0x01;
    FC_READ_INPUT_STATUS        = 0x02;
    FC_READ_HOLDING_REGISTERS   = 0x03;
    FC_READ_INPUT_REGISTERS     = 0x04;
    FC_WRITE_SINGLE_COIL        = 0x05;
    FC_WRITE_SINGLE_REGISTER    = 0x06;
    FC_DIAGNOSTICS              = 0x08;
    FC_WRITE_MULTIPLE_COILS     = 0x0f;
    FC_WRITE_MULTIPLE_REGISTERS = 0x10;
    FC_MASKED_WRITE_REGISTER    = 0x16;
    FC_READ_WRITE_REGISTERS     = 0x17;
    FC_READ_FIFO_QUEUE          = 0x18;
```

4.5.3 Class Modbus

4.5.3.1 Initialization

Using a serial port

```
from mb_tester.MBTester          import LogLevels, ConvertTypes, MBTester;
from mb_tester.Modbus            import ModbusTransmissionTypes;
from mb_tester.ModbusSerialPort import ModbusBaudRates, ModbusParities, ModbusSerialPort;
from mb_tester.Modbus            import Modbus;

# A MBTester object is required
tester = MBTester(...);

# A ModbusSerialPort object is required
port = ModbusSerialPort(...);

# Create a Modbus instance
modbus = Modbus( aTester = tester, aDefaultPort = port );
```

Using a TCP port

```
from mb_tester.MBTester import LogLevels, ConvertTypes, MBTester;
from mb_tester.Modbus import ModbusTransmissionTypes;
from mb_tester.ModbusTCPPort import ModbusTCPPort;
from mb_tester.Modbus import Modbus;

# An MBTester object is required
tester = MBTester(...);

# A ModbusTCPPort object is required
port = ModbusTCPPort(...);

# Create a Modbus instance
modbus = Modbus( aTester = tester, aDefaultPort = port );
```

4.5.3.2 Methods

4.5.3.2.1 Constructor

```
def __init__( self
            , aTester      : MBTester
            , aDefaultPort : Union[None, ModbusPort] = None ):
```

Arguments

- **aTester** : An MBTester object.
- **aDefaultPort** : An MPBusPort object to be used by default.

4.5.3.2.2 GenerateRawPackage()

Creates a raw command package to be sent later by *SendRaw()*.

```
def GenerateRawPackage( self
                      , aFunctionCode  : int
                      , aDataBytes     : List[int]
                      , aSlaveAddress  : Union[None, int]        = None
                      , aErrorCheck    : Union[None, List[int]]  = None
                      , aStartBytes    : Union[None, List[int]]  = None
                      , aEndBytes      : Union[None, List[int]]  = None
                      , aTransactionId : Union[None, int]        = None
                      , aProtocolId    : Union[None, int]        = None
                      , aLength        : Union[None, int]        = None
                      , aLabel         : Union[None, str]        = None
                      , aPort          : Union[None, ModbusPort] = None
                      ) -> List[int]:
```

Arguments

- **aFunctionCode** : The function code of the Modbus command. The value has to be between 0 and 255. Other values cause errors. (See *ModbusFunctionCodes*)
- **aDataBytes** : The data bytes of the Modbus command.
- **aSlaveAddress** : The address of the slave. The value has to be between 1 and 247. Other values cause errors.
- **aErrorCheck** : The CRC check. If it is 'None', the function generates the CRC check. If it is not 'None', then it has to be exactly two bytes in list type. Other values cause errors. (default: None)
- **aStartBytes** : The start byte(s). Used only in ASCII transmission type. (default: *ModbusProtocolConstants.ASCII_PROTOCOL_START_BYTES*).
- **aEndBytes** : The end byte(s). Used only in ASCII transmission type. (default: *ModbusProtocolConstants.ASCII_PROTOCOL_END_BYTES*)
- **aTransactionId** : Transaction identifier. (default: *ModbusProtocolConstants.TCP_PROTOCOL_TRANSACTION_ID*)
- **aProtocolId** : Protocol identifier. (default: *ModbusProtocolConstants.TCP_PROTOCOL_PROTOCOL_ID*)
- **aLength** : The value for the TCP package length field. Only applicable to TCP communication. If 'None', the the TCP package length field value will be automatically calculated based on the input parameters of this function. If not 'None', the TCP package length field value will be the same as this parameter without checking for errors. If an invalid value is given, the TCP package data field will be generated correctly with the right length based on the input parameters, but the TCP package length field value will be invalid. This can be used to test invalid packages.
- **aPort** : See *Common Function Parameters*. This parameter is not relevant because the package is not sent.
- **aLabel** : See *Common Function Parameters*.

Return Value

Returns an empty list in case of error, otherwise the raw Modbus command package.

Example

```
newCommand = modbus.GenerateRawPackage( aFunctionCode  = ModbusFunctionCodes.FREAD_INPUT_REGISTERS
                                      , aDataBytes     = [0x00, 0x00, 0x00, 0x01] );
# Log:
# 2021.07.04 13:57:23.789 [E0000,W0000] INFO    GenerateRawPackage [0x01040000000131ca]
newCommand = mpbus.GenerateRawPackage( aFunctionCode  = ModbusFunctionCodes.FC_READ_INPUT_REGISTERS
                                     , aDataBytes     = [0x00, 0x00, 0x00, 0x01]
```

(continues on next page)

(continued from previous page)

```
                                    , aStartBytes    = [0x10,0x10] );
# Log:
# 2021.07.04 13:57:23.789 [E0000,W0000] INFO    GenerateRawPackage [0x10100104000000001fa0d0a]
```

4.5.3.2.3 SendRaw()

Sends a raw Modbus command package. Can be used with *GenerateRawPackage()*.

```
def SendRaw( self
           , aBuffer         : List[int]
           , aAnswerExpected : bool                    = True
           , aPort           : Union[None, ModbusPort] = None
           , aAcceptErrors   : Union[None, List[int]]  = None
           , aLabel          : Union[None, str]        = None
           ) -> Tuple[int, List[int]]:
```

Arguments

- **aBuffer** : The raw command including the device address, data and error check.
- **aAnswerExpected** : If 'True' (default), the function waits for answer. If 'False', it does not.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*. If the aAnswerExpected attribute is 'False', then the second value is always an empty list.

Example

```
tester = MBTester();
port   = ModbusSerialPort(tester
                          , aName             = "Modbus Port 1"
                          , aDeviceId         = "COM25"
                          , aSlaveAddress     = 1
                          , aBaudRate         = ModbusBaudRates.B19200
                          , aDataBits         = 8
                          , aStopBits         = 1
                          , aParity           = ModbusParities.NONE
                          , aTransmissionType = ModbusTransmissionTypes.RTU
                          , aIsDefault        = True);
modbus = Modbus(aTester = tester, aDefaultPort = port);

rawPackage = modbus.GenerateRawPackage( aFunctionCode = ModbusFunctionCodes.FC_WRITE_SINGLE_REGISTER
                                      , aDataBytes    = [0x00, 0x02, 0x33, 0x44] );
# Log:
# 2022.09.14 16:23:23.604 [E0000,W0001] INFO      GenerateRawPackage [0x0106000233443cc9]

modbus.SendRaw( aBuffer = rawPackage, aAnswerExpected = False);
# Log:
# 2022.09.14 16:23:23.605 [E0000,W0001] OUTPUT    SendRaw [0x0106000233443cc9]
# 2022.09.14 16:23:23.652 [E0000,W0001] INPUT     SendRaw [0x0106000233443cc9]

modbus.SendRaw( aBuffer = rawPackage
              , aAnswerExpected = False
              , aLabel = "Unique label");
# Log:
# 2022.09.14 16:23:23.605 [E0000,W0001] OUTPUT    Unique label [0x0106000233443cc9]
```

(continues on next page)

(continued from previous page)

```
# 2022.09.14 16:23:23.652 [E0000,W0001] INPUT     Unique label [0x0106000233443cc9]

rawPackage = modbus.GenerateRawPackage( aFunctionCode = ModbusFunctionCodes.FC_READ_HOLDING_REGISTERS
                                      , aDataBytes    = [0x00, 0x01, 0x00, 0x03] );

(retVal, buffer) = modbus.SendRaw( rawPackage
                                 , aAnswerExpected = True
                                 , aLabel = "Error expected");
# Log:
# 2022.09.14 16:36:08.368 [E0000,W0001] OUTPUT    Error expected [0x010300010003540b]
# 2022.09.14 16:36:08.405 [E0000,W0001] INPUT     Error expected [0x0103060000334400006e24]
# 2022.09.14 16:36:08.406 [E0000,W0001] INFO      Error expected - ...
# ...Received data:  [0x0103060000334400006e24]

tester.LogBytes(LogLevels.INFO, "Retval", buffer);
# Log:
# 2022.09.14 16:36:08.409 [E0000,W0001] INFO      Retval [0x0103060000334400006e24]
```

4.5.3.2.4 Diagnostics()

Sends the Diagnostic (0x08) Modbus command with the selected sub command.

```
def Diagnostics( self
               , aSubCommand     : int
               , aSubCommandData : List[int]
               , aSlaveAddress   : int                      = 1
               , aPort           : Union[None, ModbusPort] = None
               , aAcceptErrors   : Union[None, List[int]]  = None
               , aLabel          : Union[None, str]        = None
               ) -> Tuple[int, List[int]]:
```

Arguments

- **aSubCommand** : The sub command of the diagnostication. The constants of available sub commands are presented by *ModbusDiagnosticsSubCommands* type.
- **aSubCommandData** : The data block of the sub command.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(resultCode, buffer)  = modbus.Diagnostics(
      aSubCommand     = ModbusDiagnosticsSubCommands.DIAG_SUB_RETURN_QUERY_DATA
    , aSubCommandData = [0x00, 0x00]
);
# Log:
# 2022.10.22 17:45:23.739 [E0000,W0000] OUTPUT    Diagnostics [0x0108000000000e00b]
# 2022.10.22 17:45:23.740 [E0000,W0000] INPUT     Diagnostics [0x01880187c0]

(resultCode, buffer)  = modbus.Diagnostics(
      aSubCommand     = ModbusDiagnosticsSubCommands.DIAG_SUB_RESTART_COMMUNICATIONS_OPTION
    , aSubCommandData = [0xff, 0x00]
```

(continues on next page)

(continued from previous page)

```
    , aLabel          = "Unique label"
);
# Log:
# 2022.10.22 17:45:23.739 [E0000,W0000] OUTPUT    Unique label [0x01080001ff00f03b]
# 2022.10.22 17:45:23.740 [E0000,W0000] INPUT     Unique label [0x01880187c0]
```

4.5.3.2.5 ReadDiscreteInputs()

Sends the Read input status (0x02) Modbus command.

```
def ReadDiscreteInputs( self
                      , aStartAddress  : int
                      , aCountOfInputs : int
                      , aSlaveAddress  : int                     = 1
                      , aPort          : Union[None, ModbusPort] = None
                      , aAcceptErrors  : Union[None, List[int]]  = None
                      , aLabel         : Union[None, str]        = None
                      ) -> Tuple[int, List[int]]:
```

Arguments

- **aStartAddress** : The address of the first register to start the reading from.
- **aCountOfInputs** : The count of discrete inputs to read. The value has to be between 1 and 2000.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(resultCode, buffer) = modbus.ReadDiscreteinputs( aStartAddress = 0x01
                                                , aCountOfInputs = 14 );
# Log:
# 2022.10.21 17:45:23.750 [E0000,W0000] OUTPUT    ReadDiscreteInputs [0x01020001000ea80e]
# 2022.10.21 17:45:23.751 [E0000,W0000] INPUT     ReadDiscreteInputs [0x0182018160]
```

4.5.3.2.6 ReadHoldingRegisters()

Sends the Read holding registers (0x03) Modbus command.

```
def ReadHoldingRegisters( self
                        , aStartAddress : int
                        , aReadCount    : int
                        , aSlaveAddress : int                     = 1
                        , aPort         : Union[None, ModbusPort] = None
                        , aAcceptErrors : Union[None, List[int]]  = None
                        , aLabel        : Union[None, str]        = None
                        ) -> Tuple[int, List[int]]:
```

Arguments

- **aStartAddress** : The address of the first register to start the reading from.

- **ReadCount** : The count of registers to read.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(resultCode, buffer) = modbus.ReadHoldingRegisters(aStartAddress = 0x01, aReadCount = 5);
# Log:
# 2022.10.21 17:45:23.750 [E0000,W0000] OUTPUT    ReadHoldingRegisters [0x0103000100005d409]
# 2022.10.21 17:45:23.751 [E0000,W0000] INPUT     ReadHoldingRegisters [0x01030a0164016501660000000012a2]
```

4.5.3.2.7 ReadCoils()

Sends the Read Coils (0x01) Modbus command.

```
def ReadCoils( self
             , aStartAddress : int
             , aReadCount    : int
             , aSlaveAddress : int                     = 1
             , aPort         : Union[None, ModbusPort] = None
             , aAcceptErrors : Union[None, List[int]]  = None
             , aLabel        : Union[None, str]        = None
             ) -> Tuple[int, List[int]]:
```

Arguments

- **aStartAddress** : The address of the first coil to start the reading from.
- **aReadCount** : The count of coils to read.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(resultCode, buffer) = modbus.ReadCoils(aStartAddress = 0x01, aReadCount = 10);
# Log:
# 2021.08.07 13:02:24.200 [E0000,W0000] OUTPUT    ReadCoils [0x01010001000aedcd]
# 2021.08.07 13:02:24.201 [E0000,W0000] INPUT     ReadCoils [0x0181018190]
```

4.5.3.2.8 WriteCoil()

Sends the Write Coil (0x05) Modbus command.

```
def WriteCoil( self
             , aAddress      : int
             , aState        : int
             , aSlaveAddress : int                      = 1
             , aPort         : Union[None, ModbusPort] = None
             , aAcceptErrors : Union[None, List[int]]  = None
             , aLabel        : Union[None, str]        = None
             ) -> Tuple[int, List[int]]:
```

Arguments

- **aAddress** : The address of the coil to write.
- **aState** : The new state for the coil.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, data) = modbus.WriteCoil(aAddress = 0x01, aState = 0xff);
# Log:
# 2021.08.07 13:02:24.202 [E0000,W0000] OUTPUT    WriteCoil [0x01050001ff00ddfa]
# 2021.08.07 13:02:24.203 [E0000,W0000] INPUT     WriteCoil [0x0185018350]
```

4.5.3.2.9 WriteCoils()

Sends the Write coils (0x0f) Modbus command.

```
def WriteCoils( self
              , aStartAddress    : int
              , aQuantityOfCoils : int
              , aForceData       : List[int]
              , aSlaveAddress    : int                      = 1
              , aPort            : Union[None, ModbusPort] = None
              , aAcceptErrors    : Union[None, List[int]]  = None
              , aLabel           : Union[None, str]        = None
              ) -> Tuple[int, List[int]]:
```

Arguments

- **aStartAddress** : The address of the first coil to start the writing at.
- **aQuantityOfCoils** : The quantity of coils to write.
- **aForceData** : The new states of coils.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.

- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, data )= modbus.WriteCoils( aStartAddress    = 0x02
                                      , aQuantityOfCoils = 0x0b
                                      , aForceData       = [0xd1, 0x05]);
# Log:
# 2021.08.07 13:02:24.204 [E0000,W0000] OUTPUT    WriteCoils [0x010f0002000b02d1057975]
# 2021.08.07 13:02:24.205 [E0000,W0000] INPUT     WriteCoils [0x018f0185f0]
```

4.5.3.2.10 ReadInputRegisters()

This function sends out Read input registers (0x04) Modbus command with the count of the registers for reading.

```
def ReadInputRegisters( self
                      , aStartAddress : int
                      , aReadCount    : int
                      , aSlaveAddress : int                      = 1
                      , aPort         : Union[None, ModbusPort] = None
                      , aAcceptErrors : Union[None, List[int]]  = None
                      , aLabel        : Union[None, str]        = None
                      ) -> Tuple[int, List[int]]:
```

Arguments

- **aStartAddress** : The address of the first register to start the reading from.
- **aReadCount** : The count of registers to read.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(resultCode, buffer) = modbus.ReadInputRegisters(aStartAddress = 0x01, aReadCount = 4);
# Log:
# 2022.10.21 17:45:23.750 [E0000,W0000] OUTPUT    ReadInputRegisters [0x0104000010004a009]
# 2022.10.21 17:45:23.751 [E0000,W0000] INPUT     ReadInputRegisters [0x01840182c0]
```

4.5.3.2.11 ReadWriteRegisters()

This function sends out Read/Write multiple registers (0x17) Modbus command.

```
def ReadWriteRegisters( self
                      , aReadStartAddress  : int
                      , aReadCount         : int
                      , aWriteStartAddress : int
                      , aDataBytes         : List[int]
                      , aSlaveAddress      : int                      = 1
                      , aPort              : Union[None, ModbusPort] = None
                      , aAcceptErrors      : Union[None, List[int]]  = None
                      , aLabel             : Union[None, str]        = None
                      ) -> Tuple[int, List[int]]:
```

Arguments

- **aReadStartAddress** : The address of the first register to start the reading from.
- **aReadCount** : The count of registers to read.
- **aWriteStartAddress** : The address of the first register to start the writing at.
- **aDataBytes** : The data to write.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*.

Example

```
(resultCode, buffer) = modbus.ReadWriteRegisters( aReadStartAddress  = 0x01
                                                , aReadCount         = 3
                                                , aWriteStartAddress = 0x04
                                                , aData              = [0x00, 0x02, 0x05, 0x06]);
# Log:
# 2022.10.21 17:45:23.750 [E0000,W0000] OUTPUT    ReadWriteRegisters [0x0117000100030004000408...
# ...00000002000500065 0e2]
# 2022.10.21 17:45:23.751 [E0000,W0000] INPUT     ReadWriteRegisters [0x011706016401650166c1ca]
```

4.5.3.2.12 WriteRegister()

Sends the Write single register (0x06) Modbus command.

```
def WriteRegister( self
                 , aAddress      : int
                 , aValue        : List[int]
                 , aSlaveAddress : int                      = 1
                 , aPort         : Union[None, ModbusPort] = None
                 , aAcceptErrors : Union[None, List[int]]  = None
                 , aLabel        : Union[None, str]        = None
                 ) -> Tuple[int, List[int]]:
```

Arguments

- **aAddress** : The address of the register to write.

- **aValue** : The new value for the register.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, data) = modbus.WriteRegister(aAddress = 0x00, aValue = 1500);
# Log:
# 2021.08.07 13:02:24.202 [E0000,W0000] OUTPUT    WriteRegister [0x0106000105dcdac3]
# 2021.08.07 13:02:24.203 [E0000,W0000] INPUT     WriteRegister [0x0106000105dcdac3]
```

4.5.3.2.13 WriteRegisters()

Sends the Write multiple registers (0x10) Modbus command.

```
def WriteRegisters( self
                  , aStartAddress : int
                  , aDataBytes    : List[int]
                  , aSlaveAddress : int                      = 1
                  , aPort         : Union[None, ModbusPort] = None
                  , aAcceptErrors : Union[None, List[int]]  = None
                  , aLabel        : Union[None, str]        = None
                  ) -> Tuple[int, List[int]]:
```

Arguments

- **aStartAddress** : The address of the first register to start the writing at.
- **aData** : The new values for the registers.
- **aSlaveAddress** : See *Modbus related function parameters*.
- **aPort** : See *Common Function Parameters*.
- **aAcceptErrors** : See *Common Function Parameters*.
- **aLabel** : See *Common Function Parameters*.

Return value

It returns according to the *Function Return Values*. The second value is always an empty list.

Example

```
(resultCode, data) = modbus.WriteRegisters( aStartAddress  = 0x0a
                                          , aData          = [0x10, 0x20, 0x30]);
# Log:
# 2021.08.07 13:02:24.204 [E0000,W0000] OUTPUT    WriteRegisters [0x0110000a000306001000200003006bd]
# 2021.08.07 13:02:24.205 [E0000,W0000] INPUT     WriteRegisters [0x019002cdc1]
```

4.5.4 Class ModbusSerialPort

Class representation of Modbus communication over a serial port, implements *MBPort*.

4.5.4.1 Properties

4.5.4.1.1 slave_address

The default slave address. If the slave address parameter is not given in a function call, this default slave address will be used.

```
@property
def slave_address( self ) -> int:
```

Example

```
port.slave_address = 1;
```

4.5.4.2 Methods

4.5.4.2.1 Constructor

Initializes a port by its name, sets and validates the device id and optionally sets it as the default port. It opens the port automatically.

The constructor checks and prints warnings in case of the following inconsistent settings:

- the parity none and stop bits 1 ("No parity requires 2 stop bits. Current settings are not recommended!")
- the value of 'data bits' is 8 and the transmission type is ASCII ("The ASCII transmission type requires 7 data bits. Current settings are not recommended!")
- the value of 'data bits' is 7 and the transmission type is RTU ("The RTU transmission type requires 8 data bits. Current settings are not recommended!")

```
def __init__( self
            , aTester            : MBTester
            , aName              : str
            , aDeviceId          : str
            , aSlaveAddress     : int
            , aBaudRate          : ModbusBaudRates         = ModbusBaudRates.B19200
            , aDataBits          : int                     = 8
            , aStopBits          : int                     = 2
            , aParity            : ModbusParities          = ModbusParities.EVEN
            , aTransmissionType  : ModbusTransmissionTypes = ModbusTransmissionTypes.RTU
            , aIsDefault         : bool                    = False ):
```

Arguments

- **aTester:** An MBTester object.
- **aName:** Identifier name for the port.
- **aDeviceId:** Serial port identifier string. (example: "COM1", "COM2", etc.)
- **aSlaveAddress:** See *Modbus related function parameters*.
- **aBaudRate:** The baud rate of the serial port. (default: *ModbusBaudRates.B19200*)

- **aDataBits:** The count of data bits for the serial communication. The default is the 8 bits. The possible values are 7 or 8. Other values cause errors. (default: 8)
- **aStopBits:** The count of stop bits for the serial communication. The possible values are 1 or 2. Other values cause errors. (default: 2)
- **aParity:** The parity bit setting for the serial communication. The possible values: *ModbusParities*. Other values cause errors. (default: *ModbusParities.EVEN*)
- **aTransmissionType:** The type of transmission. The possible values: *ModbusTransmissionTypes*. Other values cause errors. (default: *ModbusTransmissionTypes.RTU*)
- **aIsDefault:** If 'True', this will be set as the default port, if 'False', it will not be.

Example

```
from mb_tester import MBTester;
from mb_modbus import MBSerialPort;

tester = MBTester();
port = MBSerialPort( aTester    = tester
                   , aName      = "Port1"
                   , aDeviceId  = "COM1"
                   , aIsDefault = True   );
# Log:
# 2022.02.11 13:57:23.789 INFO    Modbus Port created ['Port1', 0, SERIAL, RTU]
```

4.5.4.2.2 Destructor

Closes the port properly.

4.5.5 Class ModbusTCPPort

Class representation of Modbus communication over a TCP port, implements *MBPort*.

4.5.5.1 Properties

4.5.5.1.1 slave_address

The default slave address. If the slave address parameter is not given in a function call, this default slave address will be used.

```
@property
def slave_address( self ) -> int:
```

Example

```
port.slave_address = 1;
```

4.5.5.2 Methods

4.5.5.2.1 Constructor

Initializes a port by its name, sets and validates the device id and optionally sets it as the default port. It opens the port automatically.

```
def __init__( self
            , aTester            : MBTester
            , aName              : str
            , aDeviceId          : str
            , aSlaveAddress     : int
            , aTransmissionType  : ModbusTransmissionTypes = ModbusTransmissionTypes.RTU
            , aIsDefault         : bool                    = False ):
```

Arguments

- **aTester:** An MBTester object.
- **aName:** Identifier name for the port.
- **aDeviceId:** Network connection identifier (e.g: "192.168.0.5:502", "192.168.0.5"). The default TCP port will be 502 if it is not given in the string.
- **aSlaveAddress:** See *Modbus related function parameters*.
- **aTransmissionType:** The type of transmission. The possible values: *ModbusTransmissionTypes*. Other values cause errors. (default: *ModbusTransmissionTypes.RTU*)
- **aIsDefault:** If 'True', this will be set as the default port, if 'False', it will not be.

Example

```
from mb_tester import MBTester;
from mb_modbus import ModbusTCPPort;

tester = MBTester();
port = ModbusTCPPort( tester
                    , aName = "Port1"
                    , aDeviceId = "192.168.10.100"
                    , aIsDefault = True );
# Log:
# 2022.02.11 13:57:23.789 INFO    Modbus TCP Port created ['Port1',0,'192.168.10.100:502']
```

4.5.5.2.2 Destructor

Closes the port properly.

CHAPTER

FIVE

NOTES