Colway Solutions
Modbus Software. Less Is More

Getting Started Guide for

Modbus Test Automation

Colway Solutions
February 23, 2017

Table of Contents

Modbus Test Automation

Introduction

Mb_Automation allows user to send multiple MODBUS commands at a time to the slave and logs the data into a text file. MB_Automation routines provide both read and write access to one or more modbus slave devices through the script generated using XML file.

Test scripts consist of ASCII text data field separated by Node elements for each function codes. They may be constructed using any XML editor application. A test script entry consists of at least 15 data fields with sub child nodes.

Features of MB_Automation

- Mb_Automator supports around 11 function codes such are,
  i. Read Coils (0x01)
  ii. Read Discrete Input (0x02)
  iii. Read Holding Registers (0x03)
  iv. Read Input Registers (0x04)
  v. Write Single Coil (0x05)
  vi. Write Single Register (0x06)
  vii. Read Exception Status (0x07)
  viii. Write Multiple Coils (0x0F)
  ix. Write Multiple Registers (0x10)
  x. Report Slave ID (0x11)
  xi. Read/Write Multiple Registers (0x17)
- Validation of Read Coils with pre-defined reference data.
- Validation of Write Coils & Registers.
- User friendly XML script.
- Data logging of real time communication statistics like sent requests, responses received, errors etc.

Syntax for automation profile

Sample automation profile (Xml Script file) consists of one parent element called "<MbProfile>" and there are around minimum 15 child elements (child elements may increase or decrease based on user requirement) out of which 11 child elements are used for MODBUS commands and the rest 3 child elements are mandatory to run Mb_Automation. Following are the three mandatory syntax.

- Repeat Count
  Using <RepeatCount> element user can decide the number of times script to run.
  XML format:

```
<RepeatCount>
<NumOfTimes_Script_Run>2</NumOfTimes_Script_Run>
</RepeatCount>
```

- Log File

  User can define the source path to store the log file and can change the log filename.

  XML format:

```
<LOGFILE>
    <Filename>D:/logFile.txt</Filename>
</LOGFILE>
```

- Modbus Mode

  Set Modbus communication mode to “RTU” or “TCP” depending on the communication preference.

  XML format:

```
< ModbusMode>
    <Mode>RTU</Mode>
</ModbusMode>
```

- mbTcpMode

  User can set the communication parameters for TCP communication.

  XML format:

```
<mbTcpMode>
    <DeviceIP>127.0.0.1</DeviceIP>
    <tcpPortNo>502</tcpPortNo>
    <Timeout>1000</Timeout>
</mbTcpMode>
```

- Serial Port

  User can set the communication parameters for the initialization of serial communication.

  XML format:

```
<SERIALPORT>
        <Parity>0</Parity>
        <Stopbits>0</Stopbits>
        <Databits>1</Databits>
        <Baudrate>3</Baudrate>
        <Timeout>1000</Timeout>
        <Portname>COM2</Portname>
</SERIALPORT>
```

XML syntax for MODBUS Commands

IMPORTANT NOTE: The MODBUS commands missing below have been explained in section 'Validation of Coils and Registers'.

- Write Single Coil (0x05)

  Syntax for Write Single Coil:

```
<MbTxn>
        <FC>0x05</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>1</Length>
        <Data>
            <Value>1</Value>
        </Data>
</MbTxn>
```

- Write Single Register (0x06)

  Syntax for Write Single Register:

```
<MbTxn>
        <FC>0x06</FC>
        <DeviceId>1</DeviceId>
        <StartAddr>1</StartAddr>
        <Length>1</Length>
        <Data>
            <Value>0xAABB</Value>
        </Data>
</MbTxn>
```

- Report Slave ID (0x11)

  Syntax for Report Slave ID:

```
<MbTxn>
        <FC>0x11</FC>
        <DeviceID>1</DeviceID>
</MbTxn>
```

- Read Exception Status (0x07)

  Syntax for Read Exception Status:

```
<MbTxn>
        <FC>0x07</FC>
        <DeviceID>1</DeviceID>
</MbTxn>
```

- Read/Write Multiple Registers (0x17)

  Note: Data value can be entered horizontally separated by comma (,) inside a tag "<Value></Value>" also user can create another new tag "<Value></Value>" in the next line (if required) and can continuing entering the data value sequentially as shown below.

Syntax for Read/Write Multiple Registers:

```
<MbTxn>
        <FC>0x17</FC>
        <DeviceID>1</DeviceID>
        <StartAddrReadReg>1</StartAddrReadReg>
        <StartAddrWriteReg>1</StartAddrWriteReg>
        <QtyReadReg>5</QtyReadReg>
        <QtyWriteReg>5</QtyWriteReg>
        <Data>
            <Value>0x1111, 0x2222, 0x3333</Value>
            <Value> 0x4444, 0x5555</Value>
        </Data>
</MbTxn>
```

Validation of Coils and Registers

Important Note: Reference data/write registers or coils values can be entered horizontally separated by comma (,) inside a tag "<Value> </Value>" and also user can create another new tag "<Value></Value>" in the next line (if required) and can continuing entering the reference data value sequentially as shown below.

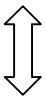
Ex1:

```
<MbTxn>
        <FC>0x01</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>16</Length>
        <CompWrMuCoils>0</CompWrMuCoils>
        <Data>
            <Value>0, 1, 1, 1, 1, 0, 1, 1</Value>
            <Value>1, 0, 1, 1, 0, 1, 1, 1</Value>
        </Data>
</MbTxn>
```

Reference Data

Ex2:

```
<MbTxn>
        <FC>0x10</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>6</Length>
        <Data>
            <Value>0x0FAA, 0xAADD</Value>
            <Value>0xCCAA, 0xFFFF</Value>
        </Data>
</MbTxn>
```



Write Register Values

- Read Coils (0x01)

MB_Automator provides a feature to cross verify the successful transaction of read coils over MODBUS.

Read Coils can be validated by providing a reference data in the XML file. The following shows the syntax for validating the read coils.

During validation of read coils with reference data, make sure that "<CmpWrMulCoils>" tag is assigned to zero (0) as shown below.

XML format for (0x01):

```
<MbTxn>
        <FC>0x01</FC>
        <DeviceID>1</DeviceID>
```

```
        <StartAddr>1</StartAddr>
        <Length>16</Length>
        <CmpWrMulCoils>0</CmpWrMulCoils>
    <Data>
        <Value>0, 1, 1, 1, 1, 0, 1, 1</Value>
        <Value>1, 0, 1, 1, 0, 1, 1, 1</Value>
    </Data>
</MbTxn>
```

Reference Data

Note: If user wants to skip the validation/comparison of read coils. Just delete the entire sub child node (referenced data) with the tag name called "<Data> </Data>".

- Write Multiple Coils (0x0F)

MB_Automator provides a feature to cross verify the successful transaction of write multiple coils over MODBUS.

XML format for (0x0F):

```
< MbTxn>
        <FC>0x0F</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>16</Length>
    <Data>
        <Value>0, 1, 1, 1, 1, 0, 1, 1</Value>
        <Value>1, 0, 1, 1, 0, 1, 1, 1</Value>
    </Data>
</ MbTxn>
```

Write Coils Values

In order to validate the write multiple coils, just call Read Coils (0x01) function code without any reference data for read coils in the XML file immediately after the write multiple coils.

During validation of write multiple coils, make sure that there is no reference data is associated with read coils function code and make sure that "<CmpWrMulCoils>" tag is assigned to one (1) as shown below.

Syntax for Read Coils (0x01) without any reference data for validation of Write Multiple Coils:

```
<MbTxn>
        <FC>0x01</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>16</Length>
        <CmpWrMulCoils>1</CmpWrMulCoils>
</MbTxn>
```

Note1: During validation/comparison of write multiple coils the "StartAddr" and "Length" of "Read Coils (0x01)" should be same as the "Write Multiple Coils". If different value is given then validation result fails.

Note2: If user wants to skip the validation/comparison of write multiple coils. Then assign zero (0) to "<CmpWrMulCoils>" tag in read coils function code.

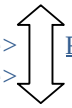
- Read Discrete Input (0x02)

MB_Automator provides a feature to cross verify the successful transaction of read discrete input over MODBUS.

Read Discrete Input can be validated by providing a reference data in the XML file. The following shows the syntax for validating the read discrete input.

XML format for (0x02):

```
<MbTxn>
        <FC>0x02</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>16</Length>
    <Data>
        <Value>0, 1, 1, 1, 1, 0, 1, 1</Value>
        <Value>1, 0, 1, 1, 0, 1, 1, 1</Value>
    </Data>
</MbTxn>
```



Reference Data

Note: If user wants to skip the validation/comparison of read discrete input. Just delete the entire sub child node (referenced data) with the tag name called "<Data> </Data>".

- Read Holding Registers (0x03)

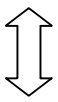
MB_Automator provides a feature to cross verify the successful transaction of Read Holding Registers over MODBUS.

Read Holding Registers can be validated by providing a reference data in the XML file. The following shows the syntax for validating the Read Holding registers.

During validation of read holding registers with reference data, make sure that "<CmpWrMulReg>" tag is assigned to zero (0) as shown below.

XML format for (0x03):

```
<MbTxn>
        <FC>0x03</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>10</Length>
        <CmpWrMulReg>0</CmpWrMulReg>
    <Data>
        <Value>0x0FAA, 0xAADD, 0xBBCC</Value>
        <Value>0xCCAA, 0xDDDD, 0xFFFF</Value>
    </Data>
</MbTxn>
```



Reference Data

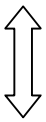
Note: If user wants to skip the validation/comparison of Read Holding register. Just delete the entire sub child node (referenced data) with the tag name called "<Data> </Data>".

- Write Multiple Registers (0x10)

MB_Automator provides a feature to cross verify the successful transaction of write multiple registers over MODBUS.

XML format for (0x10):

```
<MbTxn>
        <FC>0x10</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>3</Length>
    <Data>
        <Value>0x0FAA, 0xAADD, 0xBBCC</Value>
    </Data>
</MbTxn>
```



Write Register Values

In order to validate the write multiple registers, there is no necessity to provide reference data. To validate, just call Read Holding Registers (0x03) function code immediately after the write multiple registers without any reference data for Read Holding Registers in the XML file.

During validation of write multiple registers, make sure that there is no reference data is associated with read holding register function code and make sure that "<CmpWrMulReg>" tag is assigned to one (1) as shown below.

Syntax for Read Holding Register (0x03) without any reference data for validation of Write Multiple Registers:

```
<MbTxn>
        <FC>0x03</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>3</Length>
        <CmpWrMulReg>1</CmpWrMulReg>
</MbTxn>
```

Note1: During validation/comparison the "StartAddr" and "Length" of "Read holding registers (0x03)" should be same as the "write multiple registers (0x10)". If different value is given then validation result fails.

Note2: If user wants to skip the validation/comparison of write multiple registers. Then assign zero (0) to "<CmpWrMulReg>" tag in read holding registers function code.

- Read Input Registers (0x04)

MB_Automator provides a feature to cross verify the successful transaction of Read Input registers over MODBUS.

Read Input Registers can be validated by providing a reference data in the XML file. The following shows the syntax for validating the Read Input registers.

XML format for (0x04):

```
< MbTxn>
        <FC>0x04</FC>
        <DeviceID>1</DeviceID>
        <StartAddr>1</StartAddr>
        <Length>5</Length>
    <Data>
        <Value>0x0FAA, 0xAADD, 0xBBCC</Value>
        <Value>0xCCAA, 0xFFFF</Value>
    </Data>
</ MbTxn>
```

Reference Data

Note: If user wants to skip the validation/comparison of Read Input registers. Just delete the entire sub child node (referenced data) with the tag name called "<Data> </Data>".

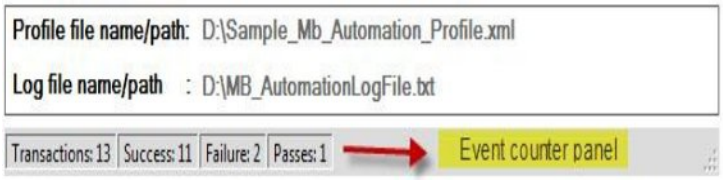
How to execute Modbus Test Automation?

1. Click on Modbus Test Automation toolbar button.
2. Load the sample automation profile supplied with the installer (can be found in the GUI installation folder. Copy into some local folder)
   Note: Ensure for proper communication settings in the sample automation profile. One of the most common causes for termination of automation test could be because of improper communication settings in the automation profile.
3. Click on 'Run Automation Profile' button.
4. On successful automation test. Following pop up window will be displayed.

In case of automation test failed a pop up error message will be displayed.

Additional information on Modbus Test Automation

- On successful automation test. Sample automation profile name/path & log file name/path information are available. As one shown below.



- Event Counter Panel

This panel displays a set of event counters to indicate status of communication of automation test with the Modbus device. A brief description of these counters follows:

a) Transaction: The total number of Modbus commands executed.
b) Success: The number of Modbus commands executed successfully.
c) Failure: The number command failed to execute (could be because of Read/write timeout, unsupported function code and so on).
d) Passes: Number of times automation profile executed.