ModbusPal

Version 1.4 – (May 2009)

http://modbuspal.sourceforge.net/

Part 1: Understanding ModbusPal

1 What is ModbusPal?

1.1 Overview

ModbusPal is a MODBUS slaves simulator. Its purpose is to offer an easy to use interface with the capabilities to reproduce realistic MODBUS environments.

The core of ModbusPal is written in Java. TCP/IP is supported natively, and the serial communication is supported if RxTx Project is installed on the computer.

Other simulation tools offer built-in functions to “animate” the data of the virtual MODBUS slaves. But there are always situations where predefined functions cannot reproduce the reality.

The approach of ModbusPal is to let the user write external scripts: what predefined functions can’t do, scripts will.

The second key feature of ModbusPal is the “Learn” mode. As ModbusPal receives MODBUS requests from the master, it will dynamically creates the missing resources: slaves, registers and coils are generated on-the-fly.

1.2 MODBUS support

ModbusPal currently doesn’t support all MODBUS functionalities. The following tables tracks the current status of MODBUS support in ModbusPal.

Medium	Supported
TCP/IP	YES
Serial	YES

Data format	Supported
RTU	YES
ASCII	no

Implementation	Supported
MODBUS	YES
J-BUS	YES

Functions	Supported
0x01 Read coils	no

0x02 Read discrete inputs	no
0x03 Read holding registers	YES
0x04 Read input registers	no
0x05 Write single coil	no
0x06 Write single register	no
0x07 Read exception status	no
0x08 Diagnostics	no
0x0B Get comm. Event counter	no
0x0C Get comm. Event log	no
0x0F Write multiple coils	no
0x10 Write multiple registers	YES
0x11 Report slave ID	no
0x14 Read file record	no
0x15 Write file record	no
0x16 Mask write register	no
0x17 Read/write multiple registers	no
0x18 Read FIFO queue	no
0x2B Encapsulated interface transport	no

2 Roadmap

ModbusPal is already fully operational regarding the above supported MODBUS functionalities and the features described in this guide.

Still, there are a number of features that are missing. This paragraph describes the functionalities that are planned for the future releases of ModbusPal.

2.1 MODBUS

- Support of ASCII data format
- Support of functions 0x06 (Write single register), 0x17 (Read/write multiple registers), 0x01 (Read coils), 0x05 (Write single coil) and 0x0F (Write multiple coils)

2.2 Scripts

- Enriched API
- Running ModbusPal in the background
- API Documentation

2.3 Interface

- Addition of Undo/Redo capabilities
- Addition of a Record/Replay feature
- Visualization of the automations as graphs.

Part 2: Defining MODBUS slaves

3 MODBUS slaves in ModbusPal

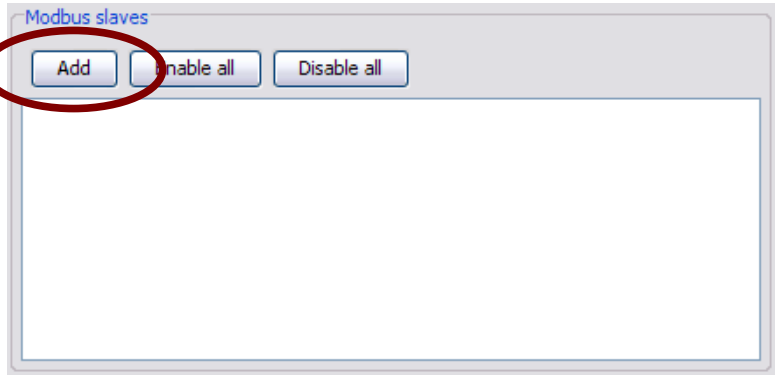
The main characteristic of a MODBUS slave is its identifier, which is used to address the requests to the desired devices. That is the reason why it is the first parameter to define when adding a MODBUS slave in a ModbusPal project.

But it is not sufficient. A MODBUS slave exchanges information with the MODBUS master. This information is organized in registers and coils. Then, once the slave is created with an identifier, its content must be defined.

This section covers the basics of creating a MODBUS slave in ModbusPal.

3.1 Adding a new slave

To add a new slave in the current project, click on the « Add » button in the main window:

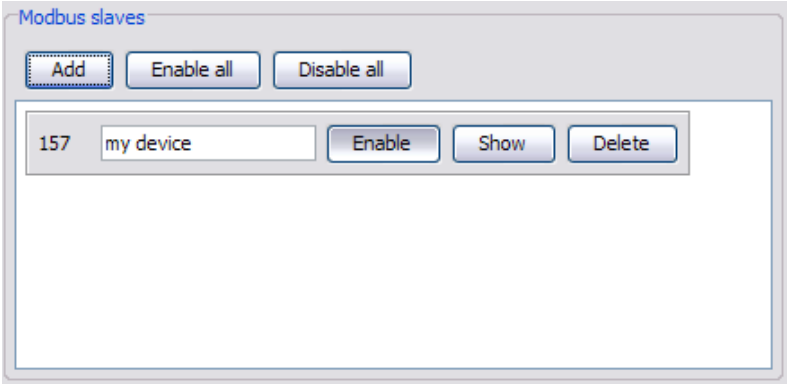


Click on the « Add » button to add a new MODBUS slave

A small dialog appears. Assign an address to the new slave from the list of available addresses and, optionally, customize its name. Specifying a name can help to explicit the role of the new slave in the project.

Assign a MODBUS address to the new slave and customize its name

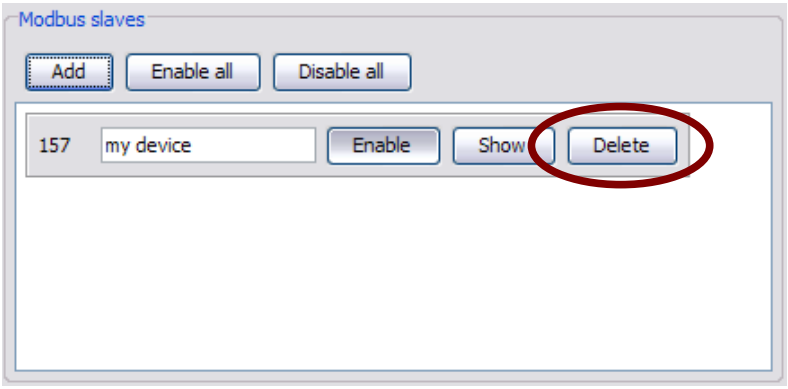
Finally, click on « Add » to validate. The MODBUS slave is inserted into the list of the main window:



The new MODBUS slave is inserted into the list

3.2 Removing a slave

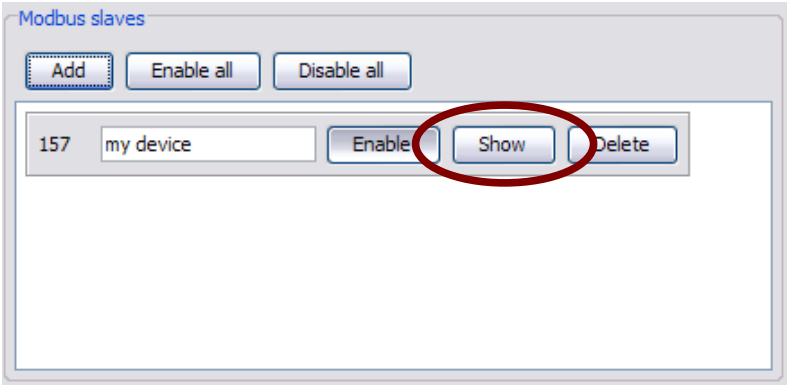
Click on the « Delete » button of one slave in the list to remove it from the current project.



Click on « Delete » to remove a MODBUS slave

3.3 Configuring the MODBUS slave

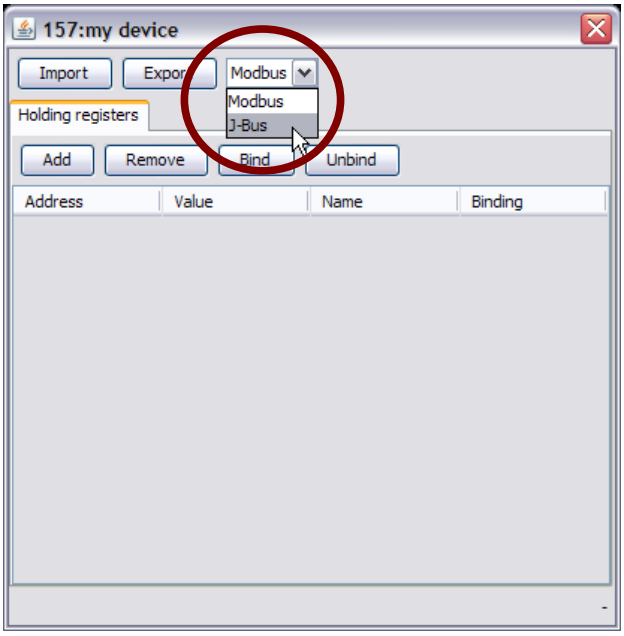
A freshly created MODBUS slave contains no register or coil. They can be defined in the slave's dialog, which becomes visible when the« Show » button is toggled.



Click on «Show» to open the slave's dialog

3.3.1 Choosing the implementation

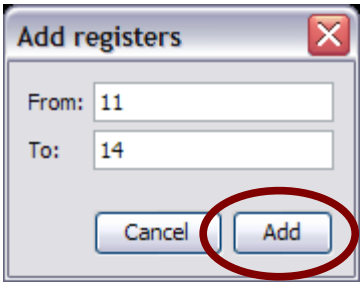
Each slave in the project can have a different MODBUS implementation. Currently supported implementations are « MODBUS » and « J-Bus ».



Choose the MODBUS implementation for the slave

3.3.2 Adding holding registers

Select the « Holding registers » tab and click on the « Add » button in order to define holding registers for a given MODBUS slave. A new dialog will appear, asking for the first and the last address of the range of registers to add into the slave.



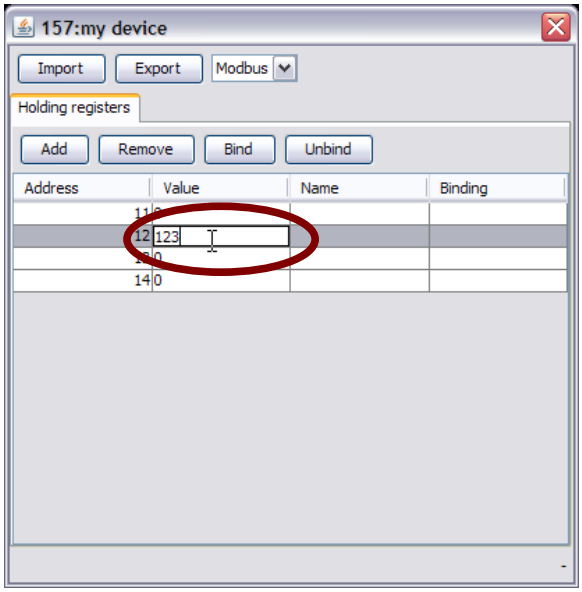
Specify the first and the last address of the holding registers to add into the slave

Note: the registers are created with '0' as a default value.

3.3.3 Editing holding registers

3.3.3.1 Changing the value

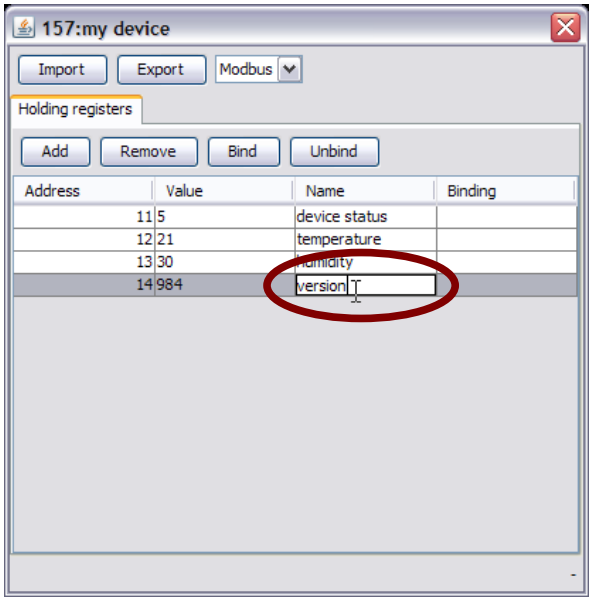
The default value of a holding register is '0'. It can be modified by double-clicking on the cell of the value. A holding register's value is comprised between 0 and 65535.



Double-click on a value in order to modify it

3.3.3.2 Editing the name

Each holding register can have a name, so that its role is clarified for the user. To edit the name, double-click on the corresponding « Name » cell.

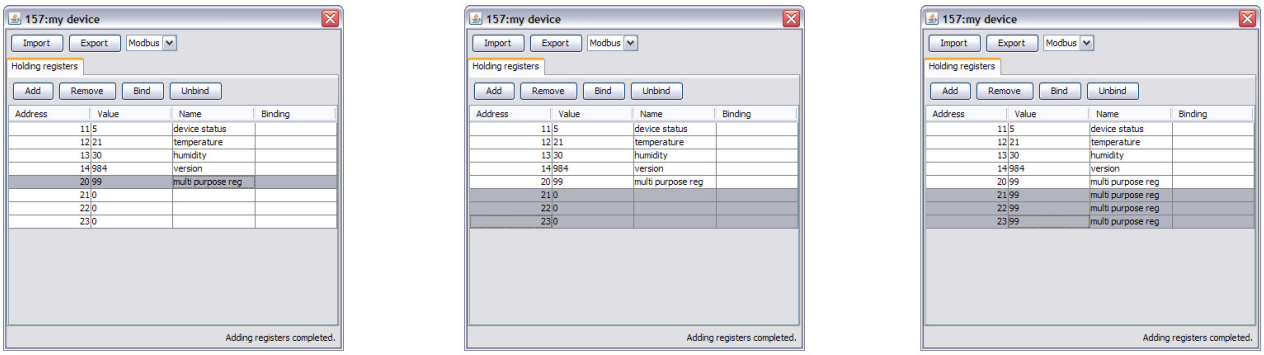


Double-click on the « Name » cell to edit it

3.3.3.3 Copy/Paste

To save time, existing values and names can be copied from one register to another.

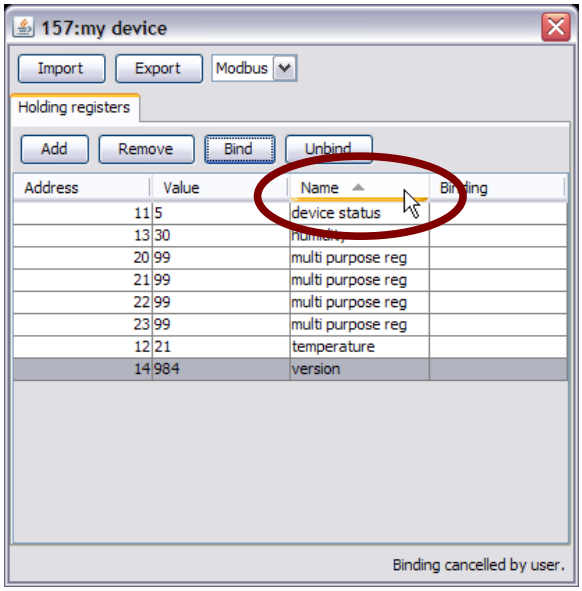
Select the holding registers to copy and press CTRL+C. Then, select the registers to modify, and press CTRL+V.



Use CTRL+C and CTRL+V to duplicate values and names

3.3.4 Sorting holding registers

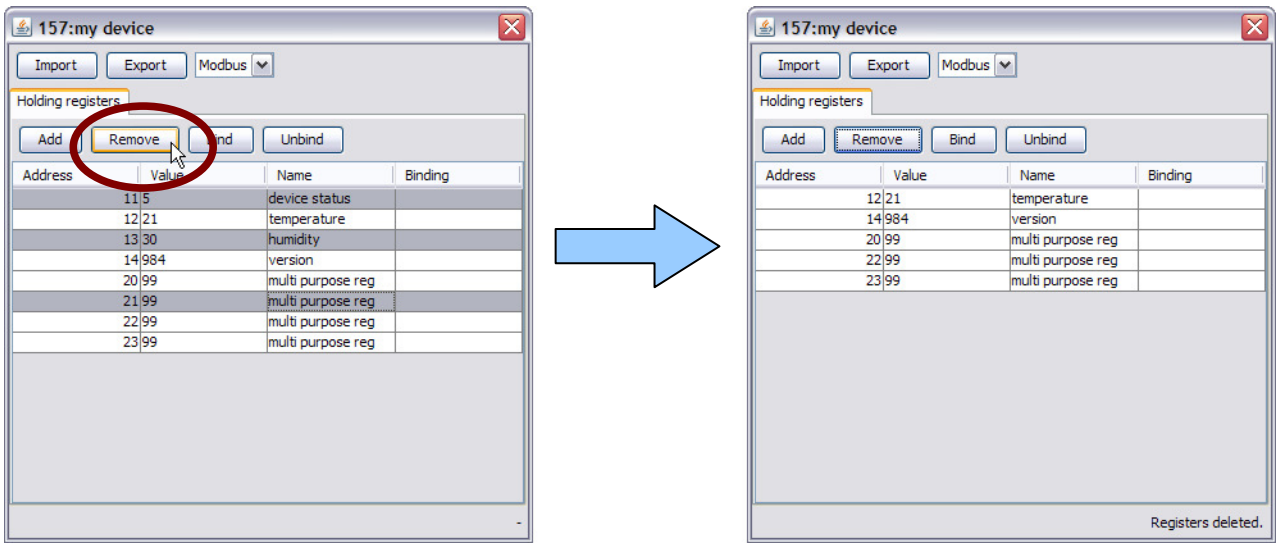
The holding registers can be sorted by address, value or name. Just click on the corresponding table header.



Click on a table header to sort the registers

3.3.5 Removing holding registers

To remove holding registers, select the corresponding rows and then click on the « Delete » button.

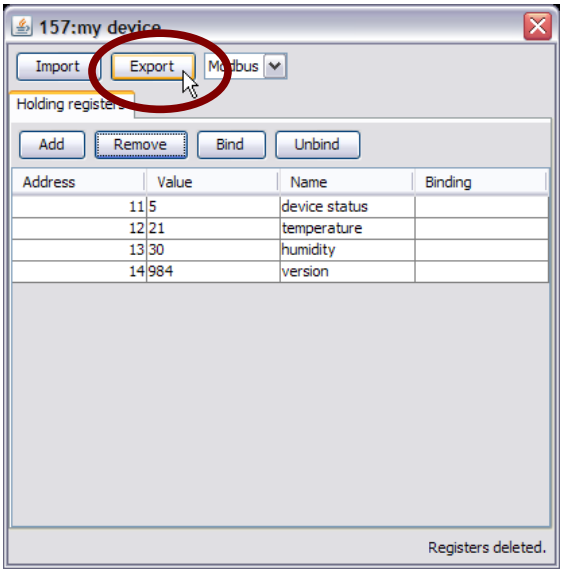


Select registers and click on « Remove » to delete them

3.3.6 Export and import

3.3.6.1 Exporting slave

Once a MODBUS slave is fully configured, it can be exported as template for creating more slaves with the same configuration. To export a slave, click on the « Export » button and choose an export file.



Click on « Export » and save the MODBUS slave as a template to create more slaves

Note: if the slave has bindings defined (see §7.1 for an overview of bindings), they can be exported as well.

3.3.6.2 Importing as new

Exported slaves can be imported as templates for new MODBUS slaves. Once a MODBUS slave has ben created (refer to §3.1, “Adding a new slave”), open the slave's dialog and click on the « Import » button. A file selection dialog appears in order to let the user choose an import file. *Et voilà!*

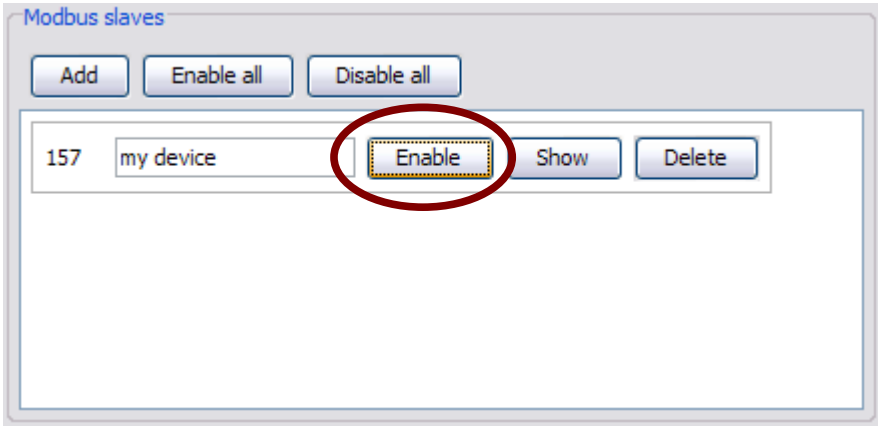
3.3.6.3 Importing and merging

When a slave export file is imported into an already existing slave, the imported data will be merged with the existing. In case of conflicts, a dialog will appear with proposals to keep the existing data or replace it with the new.

3.4 Enabling/Disabling a MODBUS slave

While the project is running, it can be useful to disable MODBUS slaves for testing purpose. Once a slave has been disabled, any incoming request addressed to that slave will end up in a MODBUS exception response.

A MODBUS slave is disabled by toggling the corresponding “Enable” button.



When disabled, a MODBUS slave looks like this

Part 3: Running the project

4 Running the project

When the MODBUS slaves have been defined, the project has to be run. When the project is running, it waits for incoming MODBUS requests, processes them and reply to the master.

As the MODBUS protocol supports serial communication and TCP/IP, so does ModbusPal. Running the project is only a matter of configuring the appropriate COM port or TCP/IP connection.

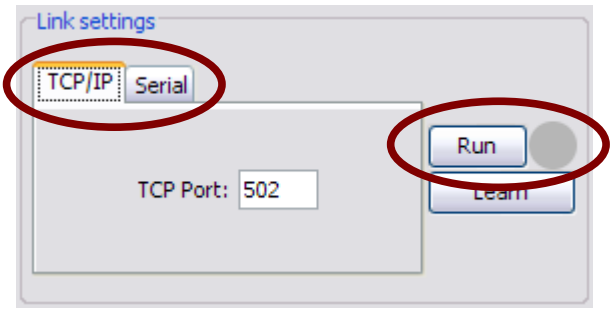
This section will also present the Learn mode, which can very efficiently create a complex project from scratch.

4.1 Built-in links

As MobusPal is a MODBUS slaves simulator, it usually is connected to a real MODBUS master device. The physical connection is either TCP/IP communication or serial link.

Note: serial communication requires that RxTx is installed on the computer.

The desired communication medium is selected by clicking on the corresponding tab in the “Link settings” panel of the main window:



Select “TCP/IP” or “Serial” in order to select the communication medium

Each link has its own specific parameters, with standard default values. When the parameters are set to match those of the real master device, clicking on the “Run” button will make ModbusPal waiting for incoming MODBUS requests.

Each time a MODBUS request is received, the LED-like icon flickers.

To stop the project from running, click on the “Run” button again.

4.1.1 TCP/IP link

Select the « TCP/IP » tab to use TCP/IP communication. The only parameter to set is the TCP port number on which incoming connections will be made. The standard MODBUS port is selected by default (502).

4.1.2 Serial link

Select the “Serial” tab in order to use serial communication. All usual COM settings can be modified, but the standard MODBUS configuration is selected by default.

Note: serial communication requires that RxTx is installed on the computer.

4.1.2.1 RxTx

ModbusPal requires that RxTx is installed on the computer in order to use the serial ports.

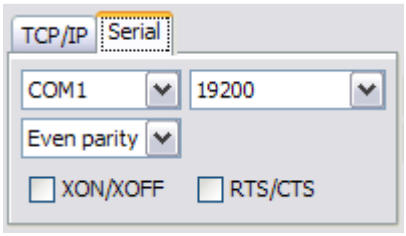
Website: http://www.rxtx.org/

“RXTX is a Java library, using a native implementation (via JNI), providing serial and parallel communication for the Java Development Toolkit (JDK). All deliverables are under the GNU LGPL license. It is based on the specification for Sun's Java Communications API, though while many of the class descriptions are the same the package used it not, since gnu.io is used instead. A certain amount of compatibility is intended with API, though this project should be considered as a fork and therefore compatible in spirit, but not in implementation.”

(quoted from RxTx website)

Refer to §**Erreur ! Source du renvoi introuvable.** (“**Erreur ! Source du renvoi introuvable.**”) for download and installation instructions.

4.1.2.2 Settings



The settings of the serial communication

- A COM port must be selected from the list of available ports. This selected port will be used to receive MODBUS requests and to send the replies.
- A baudrate can be selected from the list of standard baudrates, but any custom value can be entered by editing the text field. 19200 is the default MODBUS baudrate.
- The parity can be modified. “Even parity” is the default MODBUS parity.
- Additionally, XON/XOFF and RTS/CTS flow control methods can be enabled.

4.2 Learn mode

The Learn mode is one of the key differentiators of ModbusPal. The statement is that creating a complete project including many MODBUS slaves is very time consuming (and boring!).

When the Learn mode is activated, ModbusPal will take care of creating any resources that the project is missing.

4.2.1 Example

Imagine that you just start a new project. You only have configured the TCP/IP or serial settings, your MODBUS master is connected to your PC and ModbusPal is running.

At this point, any request of your MODBUS master will end up in errors, because you haven't defined any MODBUS slave yet.

If you activate the Learn mode, each time that ModbusPal receives a request it will:

- Create the appropriate MODBUS slave if it doesn't exist
- Create the required registers/coils if they don't exist
- Reply without error.

It just means that your entire project is automatically created "on-the-fly" by the MODBUS master's requests.

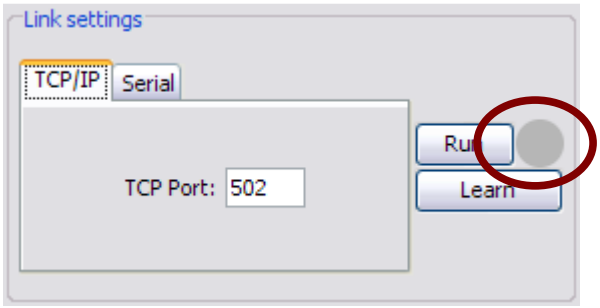
When you're satisfied with the result, you should disable the learn mode.

4.2.2 Notes

- Each register/coil created by the Learn mode gets "0" as a default value, unless the MODBUS request is a "Write…" request. In that case, the register/coil takes the value that is defined in the request.

4.3 The status LED

When the project is running, the status LED indicates that incoming MODBUS requests are received and processed.



The status LED reflects MODBUS activity

- Green flash: a MODBUS request has been received and processed successfully
- Orange flash: a MODBUS request has been received but it resulted in a MODBUS exception response
- Red flash: data has been received, but was not a correct MODBUS request

Part 4: Using automations

5 Automations in ModbusPal

At this point of the document, creating simple projects with ModbusPal has been covered. But such projects are still far from realistic MODBUS slave simulation!

The following sections are dedicated to the mechanisms that can turn simple projects into realistic simulations. The registers and coils defined in the slaves of the project need to become dynamic.

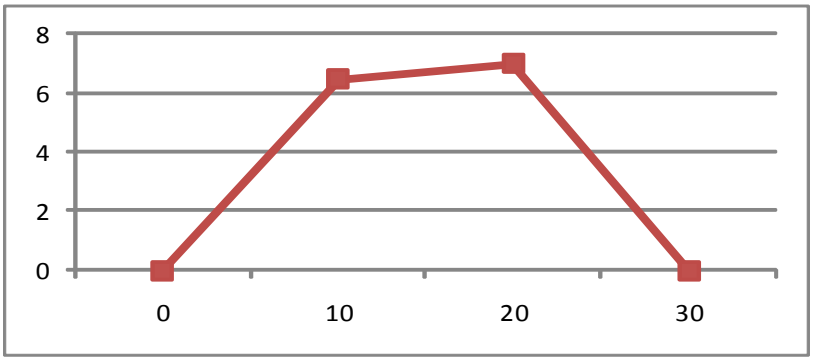
This section describes how to create dynamic values, which are called automations. Then, those automations can be associated with the registers and coils of the MODBUS slaves of the project.

5.1 Overview

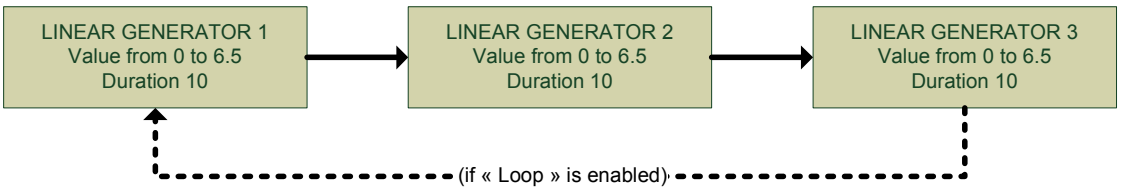
An automation is made of a succession of generators. Each generator is a formulae that produces a dynamic value. A generator is executed for a limited duration, and then the next generator is used to continue producing the dynamic value.

For example, consider the following requirements:

TIME	VALUE
0	0
10	6,5
20	7
30	0



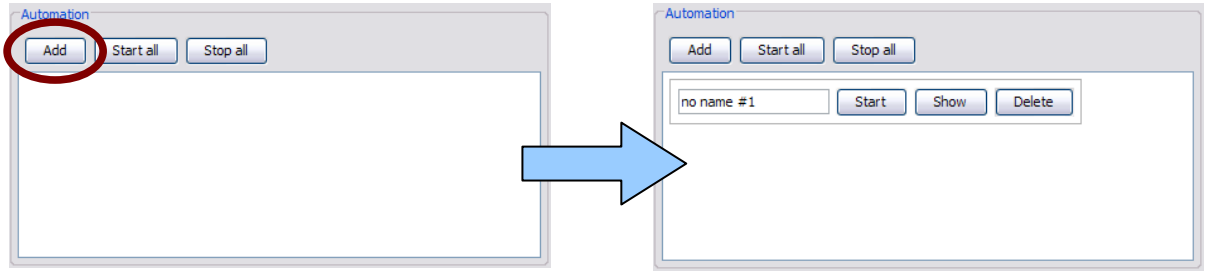
The above graph is made of 3 segments. It can be simulated by using 3 linear generators:



Additionally, the “Loop” option can be enabled so that the automation restarts from the beginning when its end is reached. Otherwise, it would stop.

5.2 Creating an automation

Clicking on « Add » button in the “Automation” tab creates a new automation in the current project:



Create a new automation by clicking on the « Add » button

The new automation is instantly added into the list. It contains no generator, and thus cannot generate a dynamic value.

5.3 Automation’s name

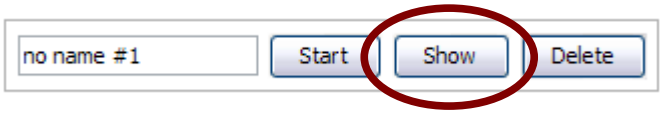
It is very important that each automation in the project has a **unique name**. The name of the automation is what ModbusPal uses to associate automations with the registers/coils of the MODBUS slaves.

The name of the automation can be changed, though. If the new name is identical to the name of another existing automation, an error message will appear.

5.4 Editing an automation

5.4.1 Open the editor

An automation is edited in a dedicated window, called automation editor. The automation editor is opened by clicking on the corresponding “Show” button in the main window:



Click on the “Show” button in order to open the automation editor

5.4.2 Settings

The settings of the automation are located in the “Global settings” and “Controls” panels:

The settings of the automation are located in ”Global settings” and “Controls”

5.4.2.1 Initial value (“Init”)

The “Init” field is a floating point value that specifies the initial value of the

automation.

5.4.2.2 Sampling rate ("Step")

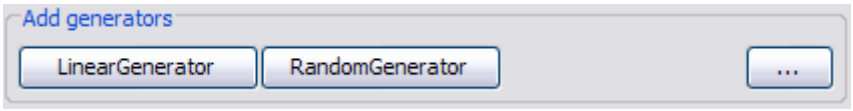
The "Step" is a decimal value in seconds, which defines the duration between two updates of the value generated by the automation. The smaller the "step" is, the higher is the refresh rate of the value.

5.4.2.3 Loop

The "Loop" toggle button determines, when the automation ends, if it should restarts automatically from the beginning.

5.4.3 Add generator

A generator is added into the automation by clicking on the appropriate button in the "Add generators" panel. Each button represents a type of generator.



Add a generator by clicking on its name, or click on "…" to add scripted generators

6 ModbusPal embeds a few built-in generators (refer to §6, "Generators"). They are always available, but may not be sufficient; More generators can be added in the project by using scripts (refer to §8, "Jython

ModbusPal uses Jython as a Python interpreter. Jython must be downloaded and installed on the computer so that ModbusPal scripting capabilities are enabled.

Website: http://www.jython.org/

"Jython implements the Python programming language on the Java(tm) Platform. It consists of a compiler to compile Python source code down to Java bytecodes which can run directly on a JVM, a set of support libraries which are used by the compiled Java bytecodes, and extra support to make it trivial to use Java packages from within Jython."

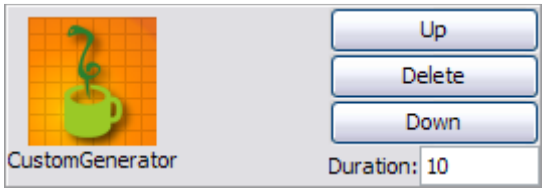
(quoted from Jython Project website)

Refer to §12.2 ("Jython") for download and installation instructions.

Scripted generators").

The new generator is always added at the end of the list.

A generator looks like this:



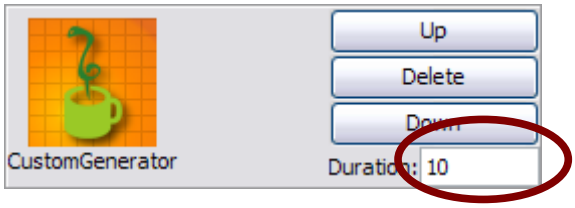
A generator looks like this

Every generator contains an icon with a subtitle on the left, some buttons and settings on the right. Optionally, it may contain additional settings in the middle, which are specific to that type of generator.

6.1.1 Duration of generators

Each generator is executed for a determined amount of time, called “duration”, and then the automation goes on with the next generator in the list.

The duration is configured by typing the desired number of seconds in the “Duration” input box of each generator in the list.



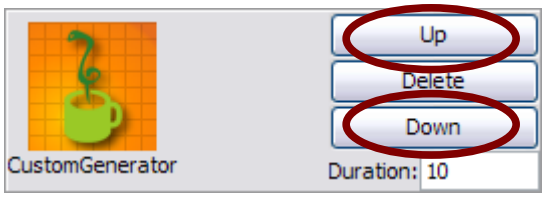
The duration of a generator, in seconds

6.1.2 Changing the order of execution

The order of execution of the generators can be changed by swapping them, using the “Up” and “Down” buttons.

Clicking on “Up” will swap the current generator with the one directly before.

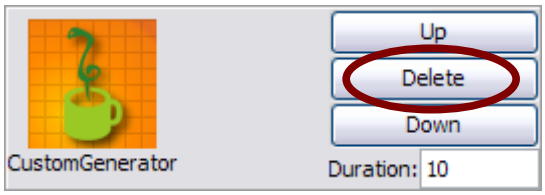
Clinking on “Down” will swap the current generator with the one directly after.



Change the execution order by using the “Up” and “Down” buttons

6.1.3 Deleting a generator

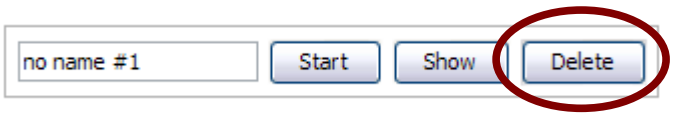
Clicking on the “Delete” button will remove a generator from the current automation.



Click on “Delete” to remove the generator from the automation

6.2 Delete automation

Clicking on the “Delete” button will remove an automation from the current project.



Click on the "Show" button in order to open the automation editor

Note: any association between the removed automation and the MODBUS slaves will also be deleted.

6.3 Binding automation

Bindings are required to associate automations and MODBUS slaves.

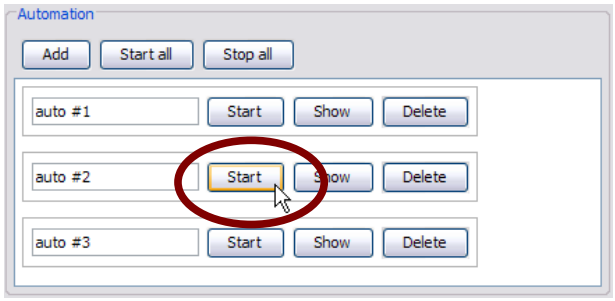
Refer to §8, "Bindings", to learn about bindings and how to create them.

6.4 Executing automations

When a project file is loaded, or when new automations are created, automations are stopped and do not generate dynamic values. They have to be started.

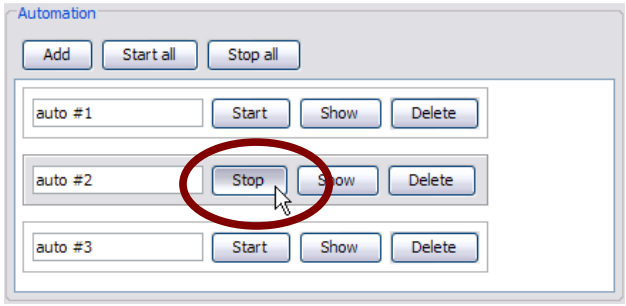
6.4.1 Start/Stop from the main window

An automation can be started or stopped from the main window by toggling the corresponding Start/Stop button in the main window:



Click on "Start" to start the execution of an automation.

When the automation starts, the corresponding entry in the list of the main window is grayed, and the "Start" button becomes a "Stop" button.



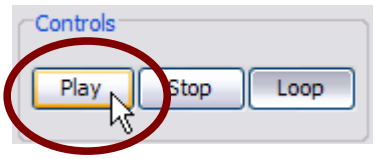
Click on "Stop" to stop the execution of the automation.

6.4.2 Start/Stop/Pause in the automation editor

An automation can be started, paused or stopped from the automation's editor. Click on the corresponding "Show" button in order to display the automation's editor, and

focus on the “Controls” panel.

Click on the “Play” button in order to start the execution of the automation.



Click on “Play” to start the execution of the automation.

While the automation is running, it can be suspended by clicking on “Pause”. As long as the automation is paused, its current state won’t be modified: the current time reference and the current output value are constant.

Click on “Pause” to suspend the automation.

Clicking on “Pause” again will let the automation resume its execution, starting from the state it was in while suspended.

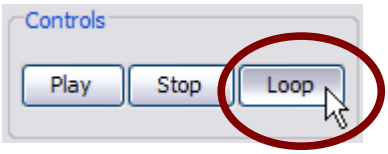
Finally, the execution of the automation can be by clicking on the “Stop” button.

Click on “Stop” to stop the execution of the automation.

6.4.3 Loop

When an automation starts, it executes each generator one by one during the specified amount of time. When the execution of the last generator is finished, the automation ends, and the generated value is no more updated, unless the “Loop” option is enabled.

If the “Loop” option is enabled, then the execution of the automation continues infinitely by restarting the whole generators sequence.



Click on ”Loop” in order to enable or disable the loop option.

Note 1: “Loop” is enabled by default.

Note 2: the internal counter that keeps track of elapsed time is not resetted when the sequence restarts.

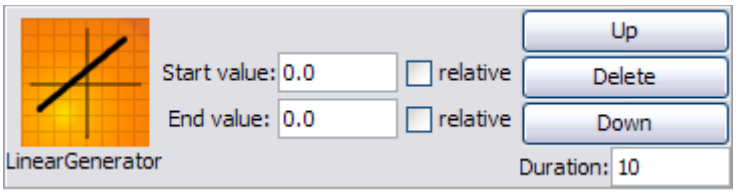
7 Generators

This section describes how to use the built-in generators and get the best out of

them.

7.1 Linear generator

The Linear generator acts like a mathematical linear function of the "$f(x) = ax + b$" kind. The first and the final values must be specified so that the "a" and "b" parameters are computed automatically.



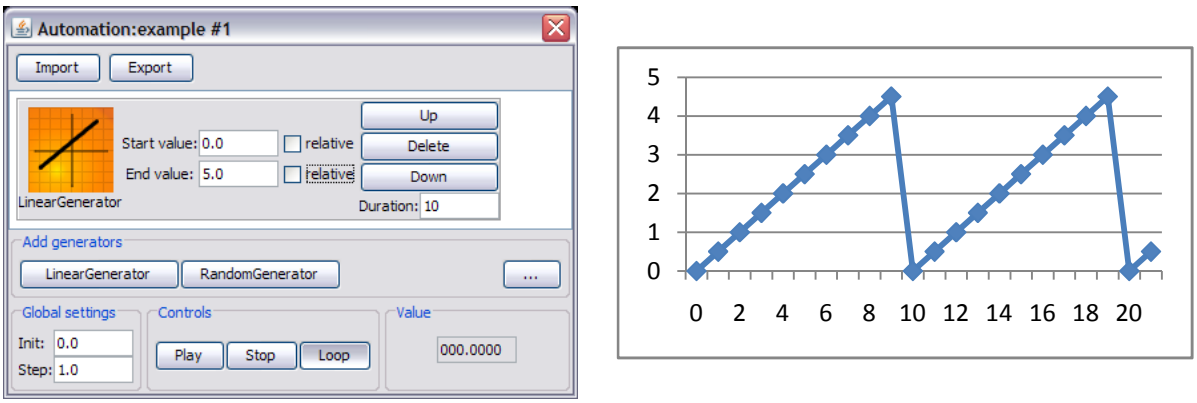
A large variety of results can be obtained by enabling or disabling the "relative" options:

- When "relative" is enabled for the "Start" value, the actual start value is the sum of the automations' value at the moment of the starting of the generator and the value in the "Start value" input box.
- When "relative" is enabled for the "End" value, the actual end value is the sum of the generator's start value and the value in the "End value" input box.

Refer to the following examples in order to get a better understanding of the "relative" option.

7.1.1 Example #1: absolute values

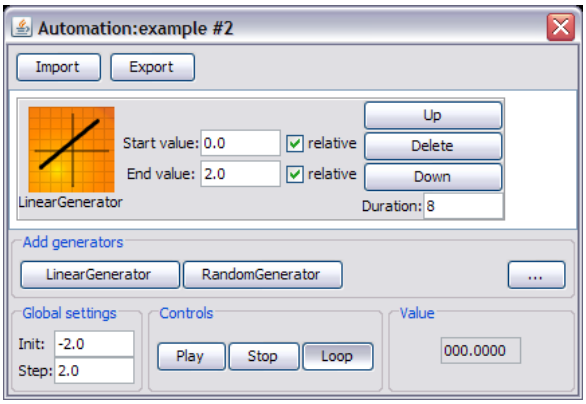
Create an automation with a single "Linear" generator inside. Use the same settings as in the example below, and the resulting dynamic value is a segment repeating itself over and over again.

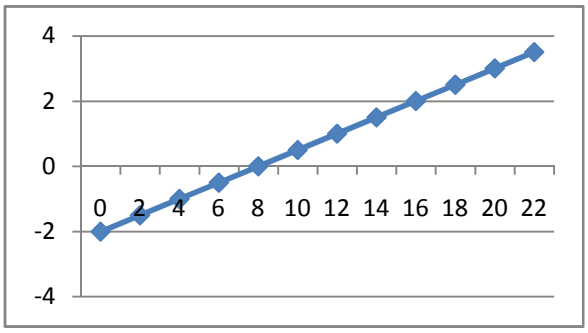


Create repeated data segment with Linear generator

7.1.2 Example #2: relative values (continuous)

Create an automation with a single "Linear" generator, and use relative starting and ending values in order to create an ever-increasing dynamic value.



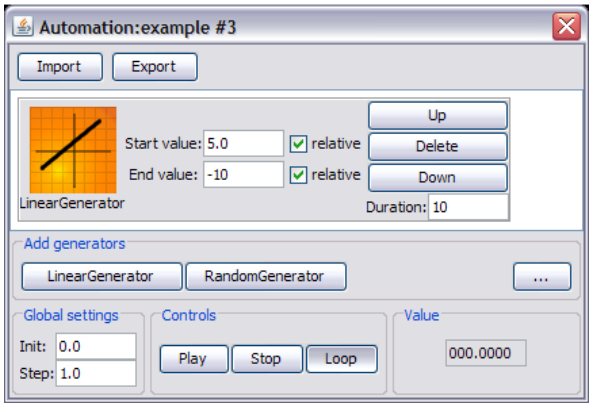


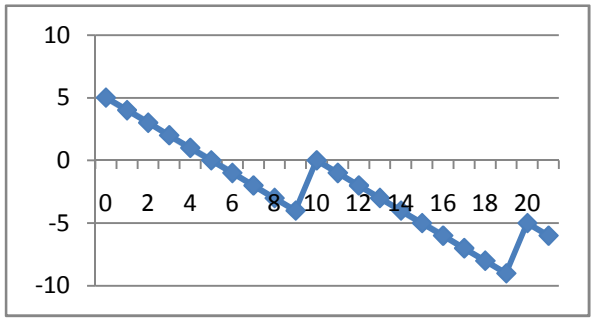
Create an ever-increasing value with the Linear generator

In this example, the initial value of the automation has been set to *-2*. Also, the "step" has been changed to *2*, which means that the value is updated every 2 seconds (instead of 1 second by default).

7.1.3 Example #3: relative values (segmented)

By changing the relative start value and using *5* instead of 0, a mix of the two above examples is obtained.

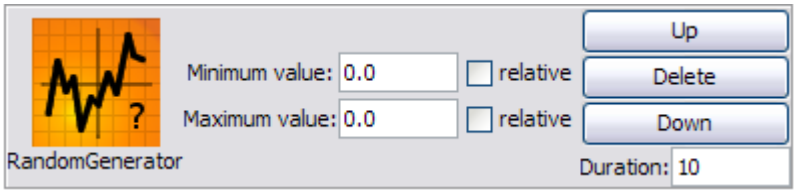




A mix of the two previous examples.

7.2 Random generator

The Random generator creates random values in the interval defined by the specified minimum and maximum values.



The minimum and maximum values are either absolute or relative.

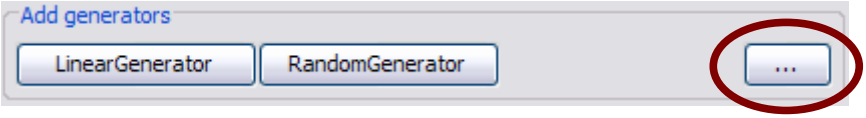
If relative, the actual minimum value is obtained by summing the current automation's value (at the moment when this generator starts) and the value specified in the "Minimum value" input box.

If relative, the actual maximum value is obtained by summing the current automation's value (at the moment when this generator starts) and the value specified in the "Maximum value" input box.

7.3 Custom generators

The project may require very specific generators in order to mimic a real-world device. The approach of ModbusPal is to let the user create its own generators thanks to scripts.

To add scripted generators, summon the script manager dialog by clicking on the "…" button.



In order to add scripted generators, summon the script manager dialog by clicking on "…"

8 Refer to chapter §9, "Jython

ModbusPal uses Jython as a Python interpreter. Jython must be downloaded and installed on the computer so that ModbusPal scripting capabilities are enabled.

Website: http://www.jython.org/

"Jython implements the Python programming language on the Java(tm) Platform. It consists of a compiler to compile Python source code down to Java bytecodes which can run directly on a JVM, a set of support libraries which are used by the compiled Java bytecodes, and extra support to make it trivial to use Java packages from within Jython."

(quoted from Jython Project website)

Refer to §12.2 ("Jython") for download and installation instructions.

Scripted generators" for more information.

9 Bindings

9.1 Overview

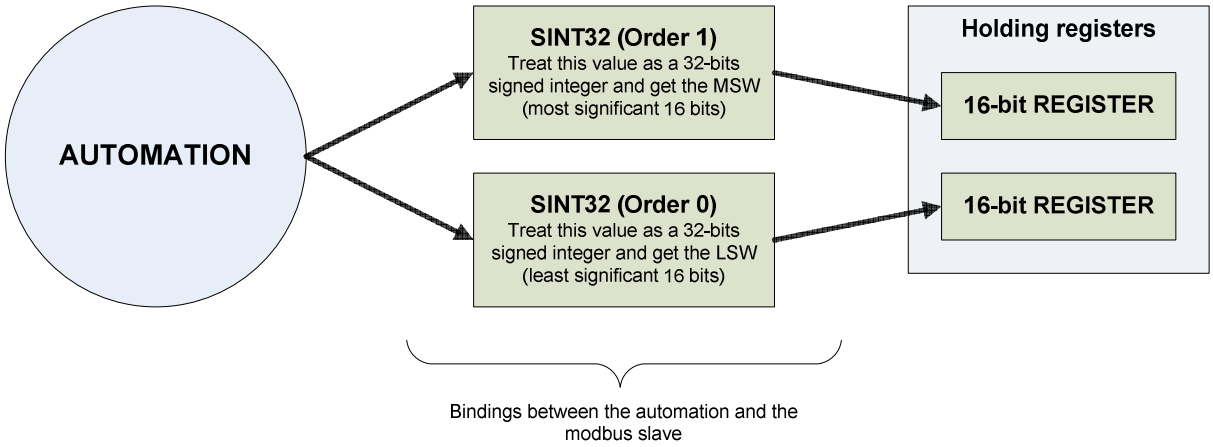
Bindings have to be created in order to use automations in the MODBUS slaves of the project.

A binding is an association between a register/coil and an automation.

The data stored in a real-world MODBUS slave is very often formatted as a combination of bytes/words in order to fit into the 16-bit registers.

For example, a value may be stored internally as 32-bits signed integer, with the most significant word (MSW) in one 16-bit register and the least significant word (LSW) in another one.

The goal of a binding in ModbusPal is precisely to create such mappings between the output value of the automations and the registers/coils of the MODBUS slaves.



The bindings define how to associate the automations and the MODBUS slaves registers/coils

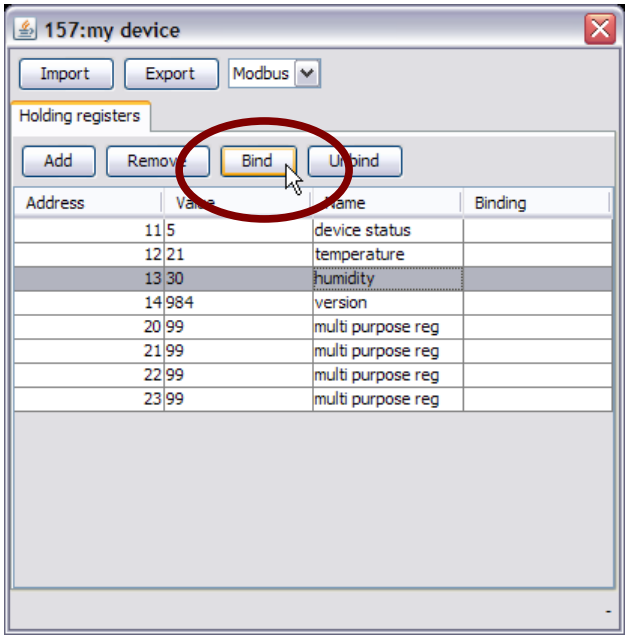
9.2 Creating a binding

The bindings are created by selecting the registers and coils directly in the dialog of a MODBUS slave

First, choose the slave for which the bindings has to be created and display its dialog (refer to §3.3, ”Configuring the MODBUS slave”). Then, choose the tab corresponding to the registers/coils that must be bound.

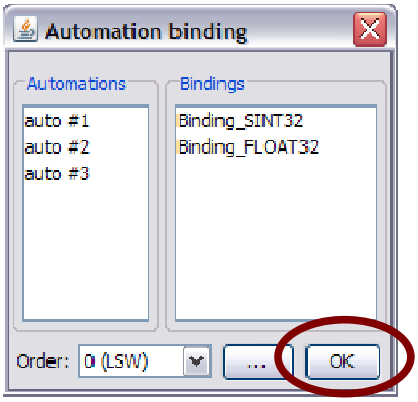
9.2.1 Single selection

Select the line corresponding to the register/coil to bind and click on the “Bind” button.



Create a binding by selecting one register/coil and clicking on “Bind”

A new dialog appears, in which the automation to bind to that register/coil must be selected from the list, along with the desired data format.



Choose the automation on the left and the data format on the right

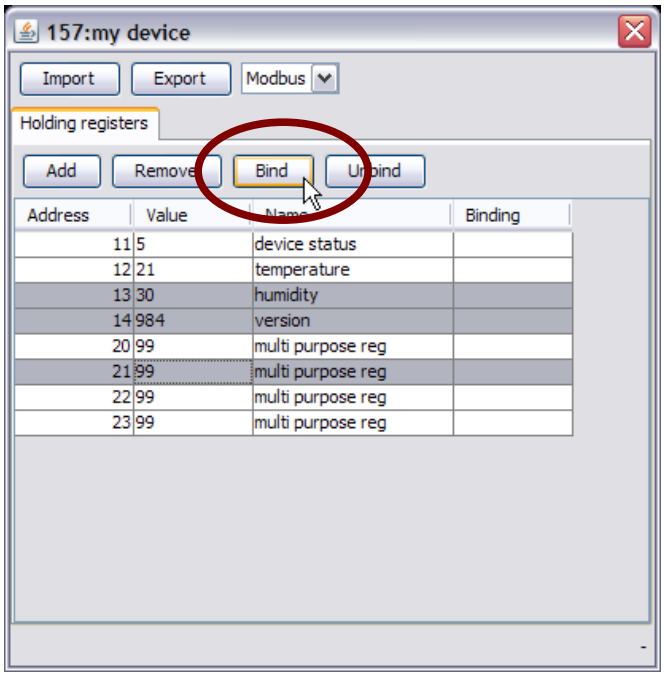
For most data formats, an order has to be explicitly defined.

For example, a 32-bit value fits into two 16-bits registers: one register for the LSW, and another one for the MSW. When a binding is defined between a register and an automation formatted as a 32-bit value, the order defines if the register will hold the LSW (order 0) or the MSW (order 1).

When all the selections are done, click on “OK” to validate.

9.2.2 Multiple selection

Hold the “Control” key of the keyboard and select the lines corresponding to the registers/coil to bind and click on the “Bind” button.



Create a binding by selecting multiple registers/coils and clicking on “Bind”

When the “Automation Binding” dialog appears, the “Order” selection is disabled. The automation to bind to those registers/coils must be selected from the list, along with the desired data format. Click on “OK” to validate the selection.

The order is automatically set starting from 0 for the first selected line, and then is increased by one for the following lines.

For example, if three registers/coils are selected, the first will get “order=0”, the second will get “order=1” and the last will get “order=2”.

9.3 Changing a binding

An existing binging can be easily modified by selecting the corresponding line and clicking on the “Binding” button. Then, a new automation/data format selection can be made, and applied by pressing the “OK” button. Changing a binding works with single or multiple line selection.

9.4 Removing a binding

To remove one or several bindings, select the appropriate registers/coils and then click on the “Unbind” button.

9.5 Predefined bindings

ModbusPal comes with a few built-in bindings. They will suffice most of the time. Otherwise, it is possible to make custom bindings thanks to Python scripts.

The following paragraphs describe the built-in bindings.

9.5.1 SINT32

The automation’s value is formatted as a 32-bit signed integer value. The least significant 16-bit word (LSW) is obtained with “order=0”. The most significant 16-bit word is obtained with “order=1”.

9.5.2 FLOAT32

The automation’s value is formatted as a 32-bit IEEE floating point number. The least significant 16-bit word (LSW) is obtained with “order=0”. The most significant 16-bit word is obtained with “order=1”.

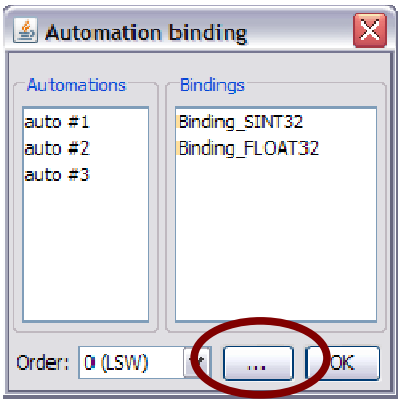
9.5.3 Summary

Name	Type	Signed	Size
SINT32	Integer	yes	32-bits
FLOAT32	IEEE float	yes	32-bits

9.6 Custom bindings

Customized bindings can be created by writing a Python script and adding it to the list of scripted bindings.

When the “binding dialog” appears, click on “…” to summon the script manager.



Click on “…” to add scripted bindings

Refer to §12, “Scripted bindings” for details on how to write scripted bindings.

Part 5: Using Python scripts

10 Jython

ModbusPal uses Jython as a Python interpreter. Jython must be downloaded and installed on the computer so that ModbusPal scripting capabilities are enabled.

Website: http://www.jython.org/

"Jython implements the Python programming language on the Java(tm) Platform. It consists of a compiler to compile Python source code down to Java bytecodes which can run directly on a JVM, a set of support libraries which are used by the compiled Java bytecodes, and extra support to make it trivial to use Java packages from within Jython."

(quoted from Jython Project website)

Refer to §12.2 ("Jython") for download and installation instructions.

11 Scripted generators

11.1 Overview

Writing a scripted generator consists of subclassing the *PythonGenerator* Java class in a Python class. The magic of interpreting the Python class is made by the Jython interpreter.

Important: Make sure that the name of the Python class matches the name of the file containing the script. For example, the script that defines a "CustomGenerator" class must be named "CustomGenerator.py".

A scripted generator should only contain a class definition, this class being a subclass of *PythonGenerator.*

11.2 Minimalist generator script

The *PythonGenerator* class has no abstract member, so there is no obligation to implement any of its method.

The following example only overrides the "getValue" method, which is responsible of returning the dynamic value created by the generator.

The returned value is "time" itself, which means that the "getValue" method is equivalent to the mathematical function $f(x) = x$.

Content of "MinimalistGenerator.py"

```
from modbuspal.script import PythonGenerator
```

```
class MinimalistGenerator(PythonGenerator):

  def getValue(self,time):
    return time;
```

11.3 Advanced generator script

The following script illustrates all the aspects of a fully customized generator.

- It defines a new icon for display in the automation editor
- It creates a control panel for display in the automation editor, letting the user customize some parameters
- It generates a dynamic value of the $f(x) = ax + b$ kind, where 'a' is a user-defined value and 'b' is the initial value of the generator.
- It saves some parameters with XML formatting into the project file
- It loads those parameters from the project file

Content of "AdvancedGenerator.py"

```
from modbuspal.script import PythonGenerator
from modbuspal.toolkit import NumericTextField
from modbuspal.toolkit import XMLTools
from java.awt import *
from javax.swing import *

class AdvancedGenerator(PythonGenerator):

  # Init function:
  # - set generator icon
  # - create the control panel
  def init(self):

    self.setIcon("./CustomGenerator.png");
    self.createCtrlPane();


  # This function will create a control panel using Java Swing components.
  # The control panel will appear in the middle of the generator panel,
  # in the automation editor.
  def createCtrlPane(self):

    self.ctrlPane = JPanel();
    self.ctrlPane.setLayout( FlowLayout() );

    self.ctrlPane.add( JLabel("A=") );
    self.aTextField = NumericTextField(1.0);
    self.ctrlPane.add( self.aTextField );


  # Override the getControlPanel function so that the
  # control panel created in the init function is returned
  def getControlPanel(self):

    return self.ctrlPane;


  # Return the generated value, f(x)=ax+b
  # where a is defined by the user (in the control panel)
  # and b is the initial value of the generator (that is the
  # current value of the automation when the generator starts).
  def getValue(self,x):
```

```
    a = float( self.aTextField.getDouble() );
    b = self.getInitialValue();
    return a*x+b;


  # Save the parameters of this generator with XML formatting into
  # the provided output stream.
  def saveSettings(self, out):

    out.write("<a value=\""+ self.aTextField.getText() +"\" />\r\n");


  # Load the parameters of this generator from the provided DOM structure.
  def loadSettings(self,nodes):

    node = XMLTools.getNode(nodes,"a");
    if not (node is None) :
      value = XMLTools.getAttribute("value",node);
      self.aTextField.setText(value);
```

12 Scripted bindings

12.1 Overview

Writing a scripted binding consists of subclassing the *PythonBinding* Java class in a Python class. The magic of interpreting the Python class is made by the Jython interpreter.

Important: Make sure that the name of the Python class matches the name of the file containing the script. For example, the script that defines the “CustomBinding” class must be named “CustomBinding.py”.

A scripted binding should only contain a class definition, this class being a subclass of *PythonBinding.*

12.2 Requirements

When writing a scripted binding, there are a few requirements that must be followed.

12.2.1 getSize()

The “getSize()” method must be overridden, otherwise the scripted binding won’t work.

The “getSize()” method returns, in bits, the total length of the formatted data created by the binding. For example, the built-in “SINT32” binding has a size of 32 bits, thus its “getSize()” method returns 32.

With this information, ModbusPal is able to determine how many coils or registers this binding can hold. The “SINT32” binding can hold 32 coils and 2 registers (a register being a 16-bit integer).

12.2.2 getRegister()

The “getRegister()” method must be overridden, otherwise the scripted binding will always return 0.

The “getRegister()” method returns a 16-bit unsigned integer value which depends on the data format of the binding and the specified order.

For example, the built-in “SINT32” binding is implemented as follow:

- if “order” is equal to 0, it returns the least significant 16-bit word of the 32-bit signed integer.
- If “order” is equal to 1, it returns the most significant 16-bit word of the same 32-bit signed integer.

The “getRegister()” method has two input arguments:

- “order” is an integer value defining the order of the register to return. Its interpretation depends on how the binding is implemented. The granularity of the order is 16 bits.
- “value” is a double precision floating point value, which is the current value of the automation. I must be cast into the data format that the binding represents, then the correct 16-bit part of this data must be extracted (depending on the value of “order”).

12.2.3 getCoil()

This method can be overridden, but in most cases its default implementation should suffice.

The default implementation of “getCoil()” is as follow:

- Retrieve the 16-bit register in which the desired coil is located. This is done by dividing the provided rank by 16. For example, the coil of rank 25 will be located in register of rank 25/16=1 (Euclidian division). The “getRegister()” method is used to retrieve it.
- Extract the correct bit in that register. The rank of the coil within the register is obtained by the remainder of the above division. For example, the coil of rank 25 will be located in the bit number 25%16=9 of the register of rank 1.

The “getCoil()” method has two input arguments:

- “order” is an integer value defining the order of the coil to return. The granularity of the order is 1 bit.
- “value” is a double precision floating point value, which is the current value of the automation. The data must be cast into the format that the binding represents, then the correct bit of this data is extracted (depending on the value of “order”).

12.3 SINT32 binding script

The SINT32 binding already exists as a built-in binding. This script is just an example of what a simple binding script looks like.

Content of “BasicBinding.py”

```
from modbuspal.script import PythonBinding
```

```
class BasicBinding(PythonBinding):

  # This binding uses a 32-bit signed integer formatting,
  # then the size is 32 bits.
  def getSize(self):
    return 32;

  # Override the getRegister() method so that it returns either
  # the least or the most significant 16-bit word of the
  # 32-bit integer.
  def getRegister(self,rank,value):

    # Cast value as an int
    value_as_a_32bit_int = int(value);

    # If rank is 0, extract the least significant 16-bit word
    if rank==0:
      value = value_as_a_32bit_int & 0xFFFF;
      return value;

    # If rank is 1, extract the most significant 16-bit word.
    elif rank==1:
      value = (value_as_a_32bit_int>>16) & 0xFFFF;
      return value;

    # It should never happen but, just in case,
    # treat the higher ranks.
    else:

      # If the 32-bit value is positive, then higher 16-bit words are 0x0000
      if value_as_a_32bit_int >= 0:
        return 0x0000;

      # If the 32-bit value is negative, then higher 16-bit words are 0xFFFF
      else:
        return 0xFFFF;
```

12.4 Advanced binding script

The following example is real-case binding script. It may help you understand why creating your own binding script can be so important for your project.

Imagine you want to simulate some MODBUS device. The manufacturer provides the following description of its “date/time” registers:

Register	High byte	Low byte
24	Minute	Second
25	Day	Hour
26	Year	Month

How do you simulate those registers with ModbusPal? We propose the following solution.

You can very easily generate your date and time in the POSIX time format. The POSIX time is a 32-bit unsigned integer which counts the number of seconds elapsed since 1st January 1970. This is a very common way of representing dates in the world of computers. All it takes to simulate it is an automation with a Linear generator.

Then, you can create a scripted binding in order to associate registers 24 to 26 with the desired information.

The binding will have a size of 3*16 = 48 bits, and the “getRegister()” method will behave as follow:

- For rank==0, return reg = minute x 256 + second
- For rank==1, return reg = day x 256 + hour
- For rank==2, return reg = year x 256 + month

Content of “AdvancedBinding.py”

```
from modbuspal.script import PythonBinding
from java.util import Calendar

class AdvancedBinding(PythonBinding):

  def getSize(self):
    return 3*16;

  # Assuming that the provided "value" is a Unix timestamp (32-bit integer
  # value representing the number of seconds since 1st January 1970),
  # this binding will transfom that timestamp to the following formatting:
  # - register #0 will contain minutes in the high byte, and seconds in the low byte
  # - register #1 will contain days in the high byte, and hours in the low byte
  # - register #2 will contain years in the high byte, and months in the low byte
  def getRegister(self,rank,value):

    unix_timestamp = long(value);
    cal = Calendar.getInstance();
    cal.setTimeInMillis( unix_timestamp * 1000 );

    if rank==0 :
      second = cal.get( Calendar.SECOND );
      minute = cal.get( Calendar.MINUTE );
      return minute * 256 + second;

    elif rank==1 :
      hour = cal.get( Calendar.HOUR_OF_DAY );
      day = cal.get( Calendar.DAY_OF_MONTH ) ;
      return day * 256 + hour;

    elif rank==2 :
      month = cal.get( Calendar.MONTH ) ;
      year = cal.get( Calendar.YEAR) % 100;
      return year * 256 + month;

    else:
      return 0;
```

13 Startup and on-demand scripts

13.1 Overview

Startup scripts and on-demand scripts are a bit different from scripted generators and bindings.

While scripted generators and bindings are only class definitions, startup scripts and on-demand scripts are procedures.

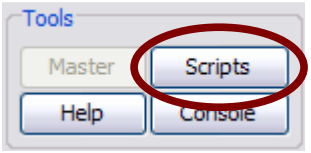
Startup scripts are executed automatically when the project is opened. They can be used to initialize the project (creating a custom GUI, for example).

On-demand scripts can be executed as often as desired, but each time they have to be triggered by the user.

13.2 Startup scripts

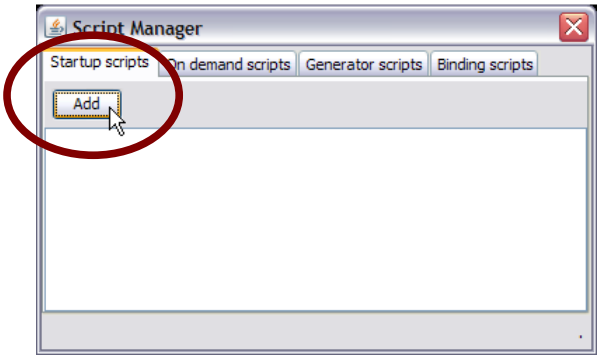
13.2.1 Adding startup scripts

Startup scripts are added by using the Script manager. Click on the “Script” button located in the main window to summon the Script manager’s window.



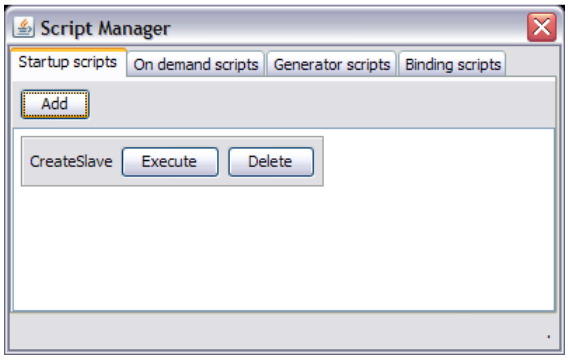
Click on the “Script” button in the main window in order to display the Script manager.

Then, in the Script manager window, select the “Startup scripts” tab and click on the “Add” button.



Click on the “Add” button in the “Startup scripts” tab in order to add a new startup script.

A file chooser dialog appears. Select a script file from your computer and validate. If the script file is valid, it is added in the list.

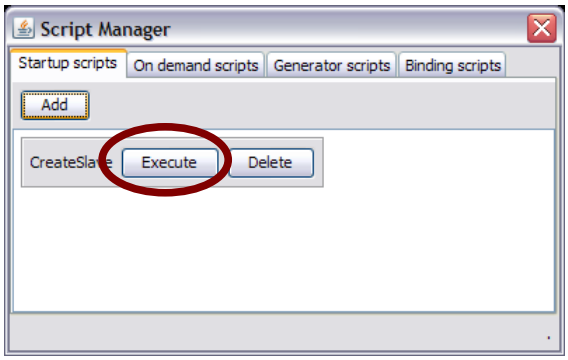


If the script file is valid, it is added in the list of startup scripts.

13.2.2 Execution of startup scripts

Startup scripts are executed automatically when the project file is loaded.

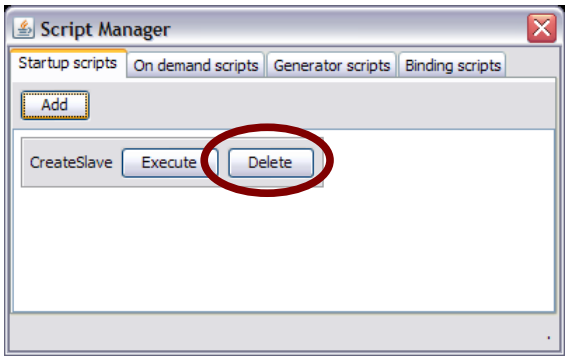
But they also can be executed later by clicking on the corresponding “Execute” button in the Script manager.



Click in the "Execute" button if you want to execute the startup script again.

13.2.3 Removing a startup script

A startup script can be removed from the project by clicking on the corresponding "Delete" button in the Script manager.



Click in the "Delete" button to delete the corresponding startup script.

13.2.4 Example

The following example is a startup script that displays a small dialog saying "Hello world". It contains an "OK" button, which hides the window when clicked. If this script is added in the project as a startup script, then this dialog will appear each time the project file is loaded.

Content of "HelloWorldStartup.py"

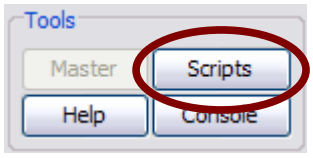
```
from javax.swing import *
from java.awt import *

class HelloWorldFrame(JFrame):

  def buttonPushed(self,event):

    self.setVisible(False);

  def __init__(self):

    self.setTitle("Hello world");
    self.setSize(300, 100);
    self.setLayout(BorderLayout());
    self.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    # create the Label
    self.label = JLabel('Hello, world !');
    self.add(self.label, BorderLayout.CENTER);
```

```
    # create the button
    self.button = JButton('OK',actionPerformed=self.buttonPushed);
    self.add(self.button, BorderLayout.SOUTH);


# Create the Hello world frame:
frame = HelloWorldFrame();

# Make it visible:
frame.setVisible(True);

# Make it the top window:
frame.toFront();
```

13.3 On-demand scripts

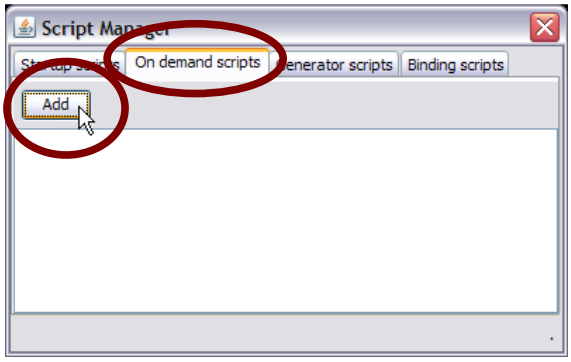
13.3.1 Adding on-demand scripts

On-demand scripts are added by using the Script manager. Click on the “Script” button located in the main window to summon the Script manager’s window.



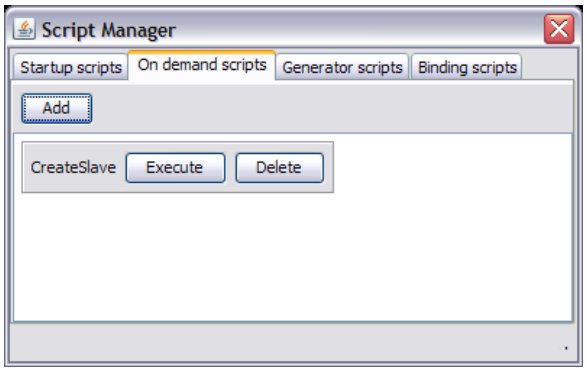
Click on the “Script” button in the main window to display the Script manager.

Then, in the Script manager window, select the “On-demand scripts” tab and click on the “Add” button.



Click on the “Add” button in the “On-demand scripts” tab to add a new on-demand script.

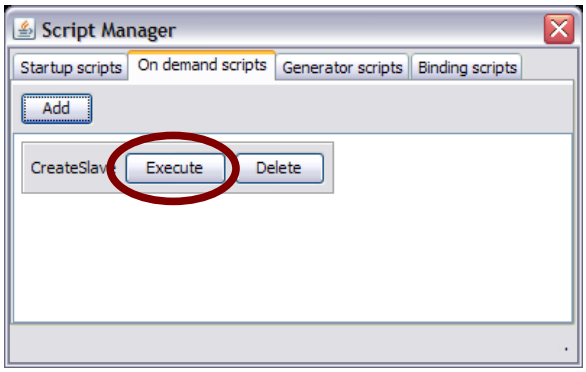
A file chooser dialog appears. Select a script file from your computer and validate. If the script file is valid, it is added in the list.



If the script file is valid, it is added in the list of on-demand scripts.

13.3.2 Execution of on-demand scripts

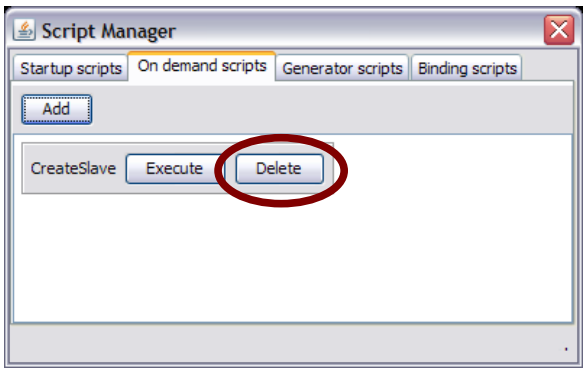
On-demand scripts are only executed if triggered by clicking on the corresponding "Execute" button.



Click on the "Execute" button to execute the on-demand script.

13.3.3 Removing an on-demand script

An on-demand script can be removed from the project by clicking on the corresponding "Delete" button in the Script manager.



Click in the "Delete" button to delete the corresponding on-demand script.

13.3.4 Example

The following example is an on-demand script that disables all MODBUS slaves defined in the current project.

Content of "DisableAllSlavesOnDemand.py"

```
from modbuspal.main import ModbusPal
from modbuspal.main import ModbusConst

# Try each possible MODBUS slave addresses
for addr in range(ModbusConst.FIRST_MODBUS_SLAVE, ModbusConst.LAST_MODBUS_SLAVE+1):

  # Get the MODBUS slave object corresponding to this address
  slave = ModbusPal.getModbusSlave(addr);

  # Verify that there actually IS a MODBUS slave with this address
  if not(slave is None):

    # Disable the slave:
    slave.setEnabled(False);
```

Part 6: ModbusPal API

TODO

Part 7: Installing third-party libraries

14 Third party libraries

14.1 RxTx

ModbusPal requires that RxTx is installed on the computer in order to enable serial communication.

"RXTX is a Java library, using a native implementation (via JNI), providing serial and parallel communication for the Java Development Toolkit (JDK). All deliverables are under the GNU LGPL license. It is based on the specification for Sun's Java Communications API, though while many of the class descriptions are the same the package used it not, since gnu.io is used instead. A certain amount of compatibility is intended with API, though this project should be considered as a fork and therefore compatible in spirit, but not in implementation."

(quoted from RxTx website)

The files and the installation instructions can be found on RxTX Project's web site:

http://www.rxtx.org/

If the installation is correct, then ModbusPal detects the presence of RxTx automatically and enables the serial communication controls. Otherwise, a warning message will appear on startup, indicating that RxTx has not been detected, and the serial communication is disabled.

14.2 Jython

ModbusPal requires that Jython is installed on the computer in order to enable usage of Python scripts in the projects.

"Jython implements the Python programming language on the Java(tm) Platform. It consists of a compiler to compile Python source code down to Java bytecodes which can run directly on a JVM, a set of support libraries which are used by the compiled Java bytecodes, and extra support to make it trivial to use Java packages from within Jython."

(quoted from Jython Project website)

The files and the installation instructions can be found on Jython Project's web site:

http://www.jython.org/

If the installation is correct, the ModbusPal detects the presence of Jython automatically and enables the support of Python scripts. Otherwise, a warning message will appear on startup, indicating that Jython has not been detected, and the scripts are disabled.

END OF THE DOCUMENT