ModbusPal

Version 1.4 – (May 2009)

<http://modbuspal.sourceforge.net/>

Part 1: Understanding ModbusPal

# What is ModbusPal?

# Roadmap

## Modbus support

### RTU/Ascii

### Function codes

## Scripting capabilities

## HMI enhancements

### Undo/Redo

### Record/Replay

### Visualization

Part 2: Defining Modbus slaves

# Modbus slaves in ModbusPal

The main characteristic of a Modbus slave is its identifier, because it is what helps you address Modbus requests to the desired device. It is why the first thing to do when you add a Modbus slave in your application is to define its identifier.

But it is not enough; you have to define the *content* of the slave, in terms of what data the slave will return when it receives a Modbus request. Once the slave is created, you must define the registers and/or coils that it will hold.

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

## Create a new slave

In order to create a new slave in the current project, you just have to click on the « Add » button:

|  |
| --- |
| Create a new slave by clicking on the « Add » button |

A small dialog appears, a let you assign a slave address to your slave. Addresses that are already assigned are not available for selection. You can also assign a name to your slave, if you like. It may help you identify the role if this slave in your project.

|  |
| --- |
| Assign a Modbus address to your slave, and define its name if you need |

Then 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 |

## Remove a slave

Removing a slave is very simple; you just have to click on the corresponding « Delete » button:

|  |
| --- |
| Click on « Delete » to remove a Modbus slave |

## Configure the Modbus slave

When you create a new Modbus slave in your project, it contains no register or coil. You have to open the slave's dialog in order to define them. If you click on the « Show » button, you will open the slave's dialog:

|  |
| --- |
| Click on «Show» to display the slave's dialog |

### Choose the implementation

You can define the Modbus implementation for each slave. Currently supported implementations are « Modbus » and « J-Bus ».

|  |
| --- |
| Choose the Modbus implementation for the slave |

### Add holding registers

If you want to define holding registers for your Modbus slave, you just have to select the « Holding registers » tab and click on the « Add » button. A new dialog will appear and ask you to specify the first and and the last address of the registers to add into the slave. Of course, you can add as many non contiguous ranges as you want.

|  |  |
| --- | --- |
| Specify the first and the last address of the holding registers you want to add |  |

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

### Edit holding registers

#### Change the value

The default value of a holding register is '0'. You can modify it by double-clicking on the cell of the value you want to modify. A holding register's value is comprised between 0 and 65535.

|  |
| --- |
| Double-click on a value in order to modify it |

#### Edit the name

You can specify a name for each holding registers of your slave, so that its role is explicited. You just have to double-click on the corresponding « Name » cell.

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

#### Copy/Paste

You can quickly duplicate the values and names that you have edited. Select the holding registers you want to copy, and press CTRL+C. Then select the registers where you want to paste the same values and names, and press CTRL+V.

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

### Sort holding registers

You can sort the holding registers by address, value or name. You just have to click on the corresponding table header.

|  |
| --- |
| Click on a table header in order to sort the registers against that column |

### Remove holding registers

You can remove holding registers by selecting the corresponding rows and then clicking on the « Delete » button.

|  |
| --- |
| Select registers and click on « Remove » to delete them |

### Export and import

#### Export slave

When you're finished designing one Modbus slave, you'll probably want to add more slaves with the same configuration; fortunately, you don't have to create them from scratch again. Open the dialog of the slave that you want to use as a template, click on the « Export » button and save the export file.

|  |
| --- |
| Click on « Export » and save your Modbus slave as a template to create more slaves |

Note: if your slave has bindings (see « **Erreur ! Source du renvoi introuvable.** »), you can choose to export the bindings or not.

#### Import as new

You can import an exported slave in order to create brand new Modbus slaves. First, you have to create a new slave as explained in « **Erreur ! Source du renvoi introuvable.** ». Open the slave's dialog; it contains no data. Click on the « Import » button and select the slave template that you want to use. *Et voilà!*

#### Import and merge

You can import a slave template into an already existing one; the imported data will be merged with the existing and, in case of clonficts, you'll be ask whether you want to keep the existing data or replace it by the new. Just click on the « Import » button and select the template that you want to use.

## Enable/Disable a Modbus slave

While you are running your project, you may find useful to disable Modbus slaves for testing purpose. When you disable a slave, any incoming request addressed to that slave will end up in a Modbus exception.

If you want to disable a Modbus slave, you just have to deselect the corresponding “Enable” button.

|  |
| --- |
| When disabled, a Modbus slave looks like this |

Part 3: Running the project

# TCP/IP and serial links

Now that you have created your Modbus slaves, you need to run your project. As the Modbus protocol supports serial communication and TCP/IP, so does ModbusPal. Running your project is only a matter of configuring your COM port or your TCP/IP connection.

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

## TCP/IP and serial links

MobusPal is a modbus slaves simulator. It means that you’ll probably want to connect a real Modbus master device to your computer. You can use a TCP/IP connection, or serial communication.

Note: serial communication requires that you install RxTx on your computer.

You choose TCP/IP or serial communication by selecting the corresponding tab in the main window:

|  |
| --- |
| Select “TCP/IP” or “Serial” in order to select your communication medium |

When you have configured the communication settings, you have to click on the “Run” button to start listening to incoming Modbus requests. Each time a Modbus request is received, the led-like icon will flicker.

You can stop processing the incoming Modbus requests by clicking on the “Run” button again.

### TCP/IP link

If you want to use TCP/IP communication, you must select the TCP/IP tab and setup the port on which you want to receive incoming connections. The standard Modbus port is selected by default (502).

Then, click on the “Run” button; ModbusPal will start listening to incoming Modbus requests over TCP/IP.

### Serial link

If you want to use serial communication through your COM ports, you must select the serial tab and use the appropriate serial settings. The standard Modbus configuration is selected by default.

Then, click on the “Run” button; ModbusPal will start listening to incoming Modbus requests over the chosen COM port.

Note: serial communication requires that you have RxTx installed.

#### RxTx

ModbusPal requires that RxTx is installed on your PC if you want to use serial communication.

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)

#### Settings

|  |
| --- |
| The settings of the serial communication |

* You can select any of the COM ports that RxTx has detected on your machine.
* You can select one of the predefined baudrates, but you also can edit the input box and put any value that you want. 19200 is the default Modbus baudrate.
* You can select the parity, “Even parity” being the default Modbus parity.
* You can enable Xon/Xoff and/or Rts/Cts flow control.

## Learn mode

The Learn mode is one of the key differentiators of ModbusPal. The statement is that the amount of work required to create your Modbus slaves can be very important, and it’s a very repetitive task.

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

### 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.

### 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.

## The status LED

When the project is running, the status LED can tell you what’s happening between your computer and your real devices.

|  |
| --- |
| The status LED reflects Modubs activity |

* Green flash: a modbus request has been received and processed successfully

Part 4: Using automations

# Automations in ModbusPal

When you have reached this part, you already know how to efficiently create projects with ModbusPal. And you have noticed that you are far from realistic Modbus slave simulation!

Now you surely need to make your project come to life. Your registers and coils need to be dynamic, and that’s where automations come in handy.

Automations are dynamic values, created by a combination of mathematical functions. Once you have created an automation, you can associate it with any number of registers/coils.

## 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.

Imagine, for example, that you want to create the following automation:

The above automation is made of 3 segments. It means that you can create it by using 3 linear generators.



In addition, you can activate the “Loop” option, so that when the end of the automation is reached, it automatically starts again from the beginning.

## Create an automation

In order to create a new automation in the current project, you just have to click on the « Add » button:

|  |  |
| --- | --- |
|  |  |
| 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. You won’t be able to run it.

## Automation’s name

It is very important that each automation of your project has a **unique name**. It is because the name of the automation is what ModbusPal uses to associates automations with the registers/coils of your slaves.

You can change the name of your automation, but you will get an error message if the new name is identical to the name of another existing automation.

## Edit an automation

### Open the editor

In order to edit an automation, you have to open the associated automation editor; just click on the “Show” button of the desired automation.

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

### Settings

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

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

#### Initial value (“Init”)

The “Init” field is a floating point value where you specify the initial value of the automation.

#### 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.

#### Loop

The “Loop” switch determines, when the automation ends, if it should restarts automatically from the beginning.

### Add generator

You add generator into the automation by simply clicking on the appropriate button in the “Add generators” panel.

|  |
| --- |
| Add a generator by clicking on its name, or click on “…” to add scripted generators |

ModbusPal embeds a list of predefined generators (see §6, “Generators“). They are always available, but may not be sufficient; you can add your own generators by using scripts (see §8, “Scripted generators”).

When you add a generator, it is 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 the selected generator.

### 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.

You configure the duration by typing the desired number of seconds in the “Duration” input box of each generator in the automation editor.

|  |
| --- |
| Here you can set the duration of a generator |

### Change order of execution

When you add a generator, it is always inserted at the end of the list. You can change the order of the generators 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 |

### Delete a generator

You just have to click on the “Delete” button in order to remove a generator from the current automation.

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

## Delete automation

You delete an automation by simply clicking on the corresponding “Delete” button in the main window.

|  |
| --- |
| 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.

## Bind automation

You need to create bindings if you want to use your automations and your Modbus slaves altogether.

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

Refer §0 ”Soon enough, you’ll need to have very particular generators in order to match your exact needs. The choice of ModbusPal is to let you create your own generators thanks to scripts.

If you want to add scripted generators, you have to summon the script manager dialog by clicking on the “…” button.

|  |
| --- |
| In order to add scripted generators, summon the script manager dialog by clicking on “…” |

See chapter §8, “Scripted generators” for more information.

Bindings” in order to learn about bindings and how to create them.

## Execute automation

When you load a project file or when you have just created a new automation, you have to start it. An automation has to be executed in order to generate dynamic values.

### Start/Stop from the main window

An automation can be started or stopped from the main window. You just have to click on the corresponding start/stop button.

|  |
| --- |
| Click on “Start” in order 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” in order to stop the execution of the automation. |

### Start/Stop/Pause from 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.

You just have to click on the “Play” button in order to start the execution of the automation.

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

While the automation is running, you can put it into a suspended state by clicking on pause. As long as the automation is paused, its current state won’t be modified: the current generator and the current generated value are constant.

|  |
| --- |
| Click on “Pause” in order 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, you can stop the execution of the automation by clicking on the “Stop” button.

|  |
| --- |
| Click on “Stop” in order to stop the execution of the automation. |

### Loop

When you start the automation, 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 you enable the “Loop” option.

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: “Loop” is enabled by default.

# Generators

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

## Linear generator

The Linear generator acts like a mathematical linear function of the “” kind. You have to specify the first and the final value so that the “a” and “b” parameters are computed automatically.

|  |
| --- |
|  |

You can get a large variety of results by enabling or disabling the “relative” options:

* When “relative” is enabled for the “Start” value, then 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, then the actual end value is the sum of the generator’s start value and the value in the “End value” input box.

Please refer to the following examples in order to get a better understanding of the “relative” option.

### 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 | |

### 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 | |

You can notice that, 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).

### Example #3: relative values (segmented)

By changing the relative start value and using *5* instead of 0, you will get mix of the two above examples.

|  |  |
| --- | --- |
|  |  |
| A mix of the two previous examples. | |

## 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.

## Custom generators

Soon enough, you’ll need to have very particular generators in order to match your exact needs. The choice of ModbusPal is to let you create your own generators thanks to scripts.

If you want to add scripted generators, you have to summon the script manager dialog by clicking on the “…” button.

|  |
| --- |
| In order to add scripted generators, summon the script manager dialog by clicking on “…” |

See chapter §8, “Scripted generators” for more information.

# Bindings

## Overview

You need to create bindings if you want to use your automations and your Modbus slaves altogether.

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

If you have already been using a real Modbus slave before, then you are familiar with the fact that a value can be mapped to the registers/coils in various kinds of ways.

For example, a value may be formatted as 32-bits signed integer, with the MSW in one register and the LSW in another. Defining bindings in ModbusPal helps you mapping the output value of your automations into the registers/coils of your Modbus slaves.

|  |
| --- |
| The bindings define how to associate the automations and the Modbus slaves registers/coils |

## Create a binding

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

First, you have to choose the slave for which you want to create bindings and display its dialog (see §3.3, ”Configure the Modbus slave”). Then, choose the tab corresponding to the registers/coils you want to bind.

### Single selection

Select the line corresponding to the register/coil that you want 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 you must choose the automation to bind to the register/coil, and the data format.

|  |
| --- |
| binding_editor.png  Choose the automation on the left and the data format on the right |

For most data formats, you’ll have to explicitly define the order of the formatted value that you’re interested in.

For example, if you select the 32-bit signed integer data format for a holding register, you have to choose between the low-order word (least significant 16-bit word) and the high-order word (most significant 16-bit word).

When you’re done with your selection, click on “OK” to validate.

### Multiple selection

Hold the “Control” key of your keyboard and select the lines corresponding to the registers/coil that you want 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, you’ll notice that the “Order” selection is disabled. You must select the automation you want to bind, and the data format, but the order will be automatically set when you click on “OK”.

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 you select three registers/coils, the first will get “order=0”, the second will get “order=1” and the last one will get “order=2”.

## Change a binding

You can easily modify an existing binging by selecting the corresponding line and click on the “Binding” button. Then, make a new automation/data format selection, and apply it by pressing the “OK” button. Changing a binding works with single or multiple line selection.

## Remove a binding

If you want to remove one or several bindings, simply select the appropriate registers/coils and then click on the “Unbind” button.

## Predefined bindings

ModbusPal comes with a built-in set of bindings. Most of the time, they will suffice to do what you need to do. Otherwise, there is the possibly to make your own bindings thanks to Python scripts.

### SINT32

The automation’s value is formatted as a 32-bit signed integer value. You can get the lower 16- bits with “order=0”, and the higher 16-bits with “order=1”.

### FLOAT32

The automation’s value is formatted as a 32-bit IEEE floating point number. You can get the lower 16- bits with “order=0”, and the higher 16-bits with “order=1”.

### Summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Type** | **Signed** | **Size** | **Min. value** | **Max. value** |
| SINT32 | Integer | yes | 32-bits |  |  |
| FLOAT32 | IEEE float | yes | 32-bits |  |  |

## Custom bindings

If you need to create your own customized bindings, you can do so by writing a Python script and add it into the list of scripted bindings.

When the “binding dialog” appears, you just have to click on “…” to display the appropriate tab of the script manager.

|  |
| --- |
| binding_editor.png  Click on “…” if you need to add scripted bindings |

See §9, “Scripted bindings” for details on how to write scripted bindings.

Part 5: Using Python scripts

# Scripted generators

## 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 Python interpreter (Jython).

**Important**: Make sure that the name of the Python class that you create matches the name of the file containing the script. For example, if you create a “CustomGenerator” class that is a subclass of *PythonGenerator*, then you must ensure that your script file is named “CustomGenerator.py”.

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

## 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 .

|  |
| --- |
| *Content of “MinimalistGenerator.py”* |
| from modbuspal.script import PythonGenerator  class MinimalistGenerator(PythonGenerator):  def getValue(self,time):  return time; | |

## 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 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); | |

# Scripted bindings

## 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 Python interpreter (Jython).

**Important**: Make sure that the name of the Python class that you create matches the name of the file containing the script. For example, if you create a “CustomBinding” class that is a subclass of *PythonBinding*, then you must ensure that your script file is named “CustomBinding.py”.

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

## Requirements

When writing a scripted binding, there are a few requirements that you must bear in mind.

### getSize()

You must override the “getSize()” method, otherwise your 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).

### getRegister()

You must override the “getRegister()” method, otherwise your 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 rank.

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

* if “rank” is equal to 0, it returns the least significant 16-bit word of the 32-bit signed integer.
* If “rank” 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:

* “rank” is an integer value defining the rank of the register to return. Its interpretation depends on how the binding is implemented. The granularity of the rank is 16 bits.
* “value” is a double precision floating point value, which is the current value of the automation. You have to cast it into the data format that the binding represents, and then extract the correct 16-bit part of this data, depending on the value of “rank”.

### getCoil()

You may override this method, 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:

* “rank” is an integer value defining the rank of the coil to return. The granularity of the rank is 1 bit.
* “value” is a double precision floating point value, which is the current value of the automation. You have to cast it into the data format that the binding represents, and then extract the correct bit of this data, depending on the value of “rank”.

## SINT32 binding script

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

|  |
| --- |
| *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; | |

## 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; | |

# Startup and on-demand scripts

## 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. You can use them to initialize your project, for example for creating a custom GUI.

On-demand scripts can be executed as many times as you want, but only if you explicitly trigger them.

## Startup scripts

### Add startup scripts

You add startup scripts by using the Script manager. You just have to click on the “Script” button located in the main window to summon it.

|  |
| --- |
| 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. |

You are then invited to select a script file from your computer. If the script file is valid, it is added to the list.

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

### Execution of startup scripts

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

But you can also execute them 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. |

### Remove a startup script

You can remove a startup script from the project by clicking the corresponding delete button in the Script manager.

|  |
| --- |
| Click in the “Delete” button if you want to delete the corresponding startup script. |

### 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 you click on it. If you add this script to your project as a startup script, then this dialog will appear each time that you load your project file.

|  |
| --- |
| *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(); | |

## On-demand scripts

### Add on-demand scripts

You add on-demand scripts by using the Script manager. You just have to click on the “Script” button located in the main window to summon it.

|  |
| --- |
| Click on the “Script” button in the main window in order 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 in order to add a new on-demand script. |

You are then invited to select a script file from your computer. If the script file is valid, it is added to the list.

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

### Execution of on-demand scripts

On-demand scripts are only executed if you click on the corresponding “Execute” button.

|  |
| --- |
| Click in the “Execute” button each time you want to execute the on-demand script. |

### Remove an on-demand script

You can remove an on-demand script from the project by clicking the corresponding “Delete” button in the Script manager.

|  |
| --- |
| Click in the “Delete” button if you want to delete the corresponding on-demand script. |

Part 6: ModbusPal API

TODO

Part 7: Installing third-party libraries

# RxTx

Xxxx

# Jython

Xxxx