Modbus Remote Manager Installation Instructions for Digi DAL Routers

Change Log

1.0.2	Initial Release

Contents

Introduction

Modbus Remote Manager is a Python-based application that allows users to poll Modbus registers from devices connected to Digi hardware and sends the Modbus data to Digi Remote Manager in a metrics datapoint format. This application also allows users to write values back to existing registers on connected Modbus equipment.

This document will cover how to install the application onto your Digi hardware, and the parameters used for the necessary configuration files used with the application.

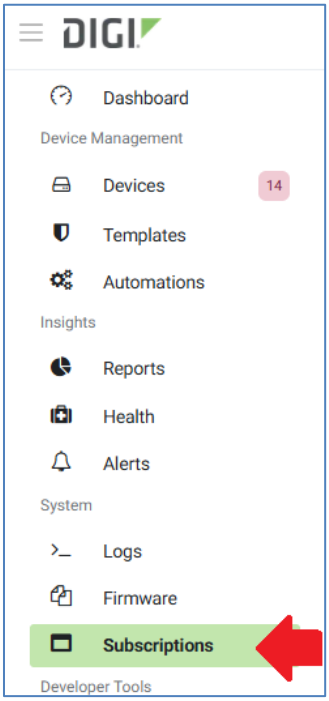
Installation and Auto-Start Instructions

This area will cover how to apply the license to the Digi hardware, how to install the application onto the Digi hardware, and how to configure the device to start the application, all using the Digi Remote Manager interface.

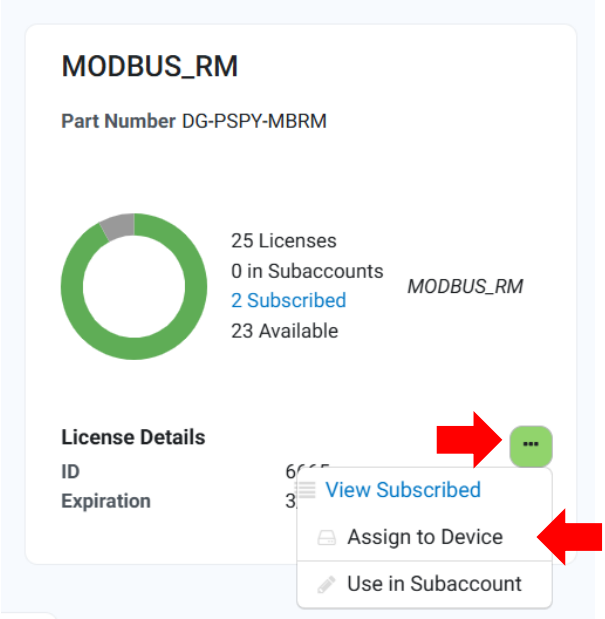
Installation

Apply the application subscription to the target device

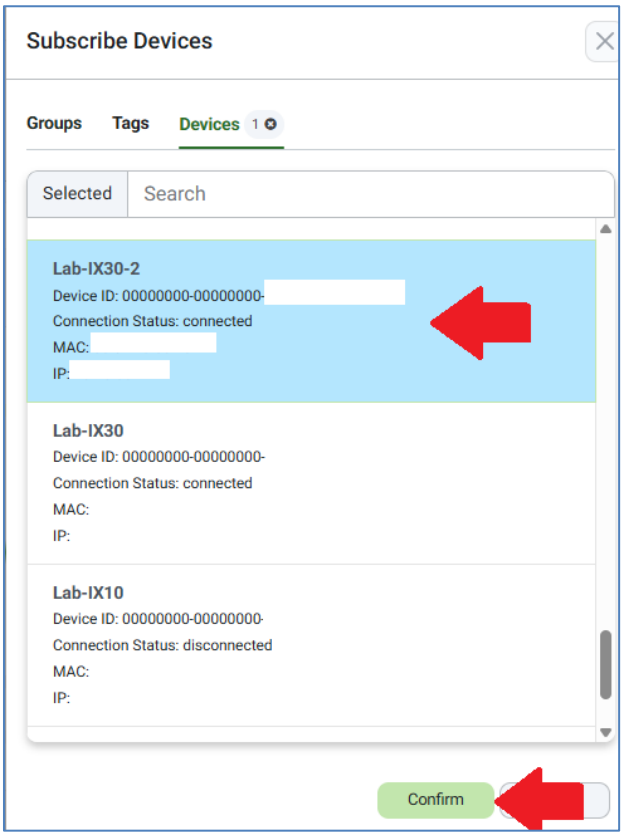
1) Log into Digi Remote Manager (Digi RM) using your credentials. If you do not have a Digi RM account, please purchase one by contacting Digi Sales or your Digi Authorized Reseller.
2) Register your Digi device(s) into Digi RM. For instructions on this process, please see this link.
3) Using the left-hand navigation panel, navigate to **System > Subscriptions**:



4) Locate the **MODBUS_RM** tile on the Subscriptions page, and click the 3 dots to expand the sub-menu. On the sub-menu, click **Assign to Device**:



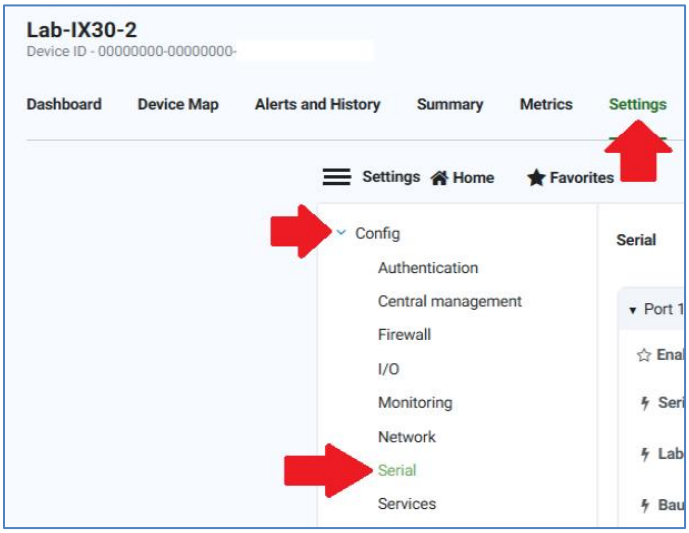
5) On the **Subscribe Devices** window, choose the device(s) the license should be applied to, and then click **Confirm** to complete the process.



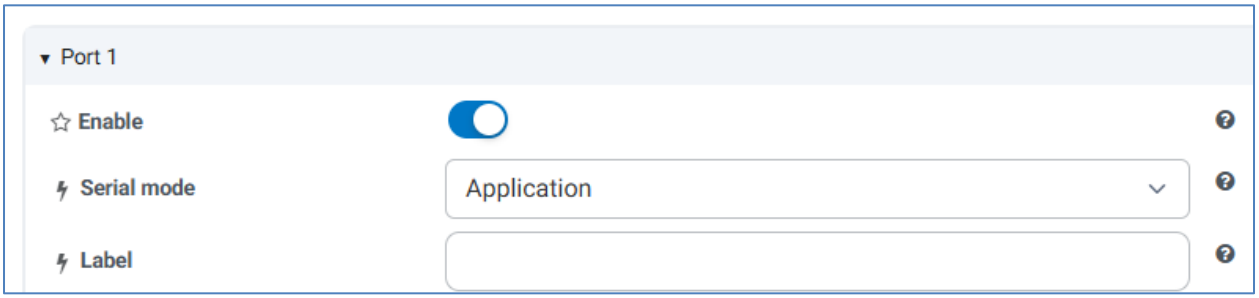
Configuring Modbus RTU Mode

If you need to use Modbus RTU mode in your application, you will need to configure the serial port of the Digi device to be in Application mode.

1) Within Digi RM, navigate to **Device Management > Devices**, and then click on the device that needs to be configured.
2) On the device's page, navigate to **Settings**, expand **Config**, and click on **Serial**:



3) **Expand** Port 1 (or another port if your device has multiple serial ports).
4) If disabled, **enable** the serial port. Change the **Serial mode** to **Application**:



5) Click **Apply** to save the changes.

It is possible to use Templates within Digi RM to help automate pushing configuration settings to the Digi devices. Please see the Templates documentation for more information.

Upload the Configuration Files to the Digi Device

There are 3 configuration files and 1 application file that need to be installed onto the target Digi device. These files are:

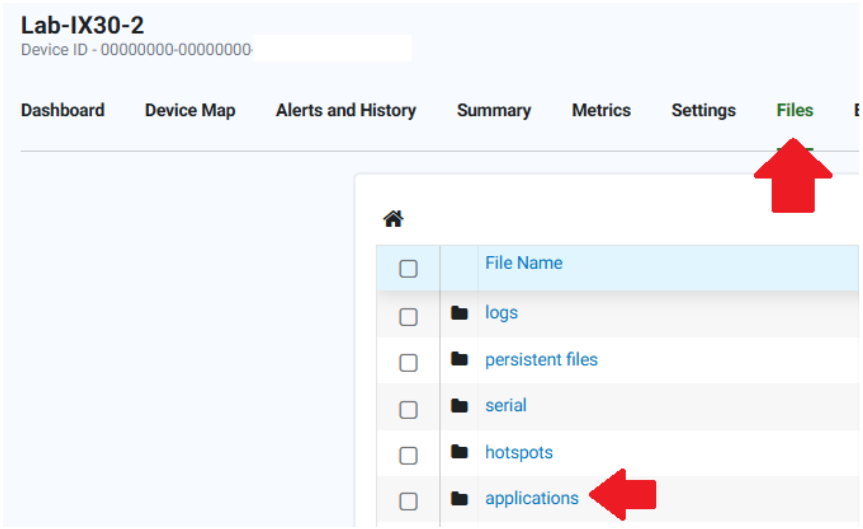
1) mb_rm.zip
2) config.cfg
3) servers.cfg
4) regs.csv

These files will need to be uploaded to the **/etc/config/scripts** location on the target Digi device. This location is known as the **applications** folder in Digi RM.

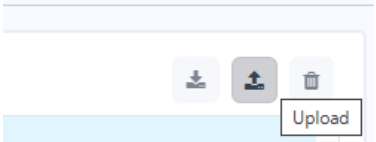
NOTE: Before uploading the config.cfg, servers.cfg, or regs.csv files to the Digi device, ensure that these files have been configured in the desired manner for your use case. Sections further below in this manual describe how to configure these files if more information is needed.

To upload the files to the Digi device:

1) Within Digi RM, navigate to **Device Management > Devices**, and then click on the device that needs to have the files.
2) On the device's page, navigate to **Files**, and then click on **applications**:



3) Click on the **Upload** button, and choose the files to upload to the Digi device:



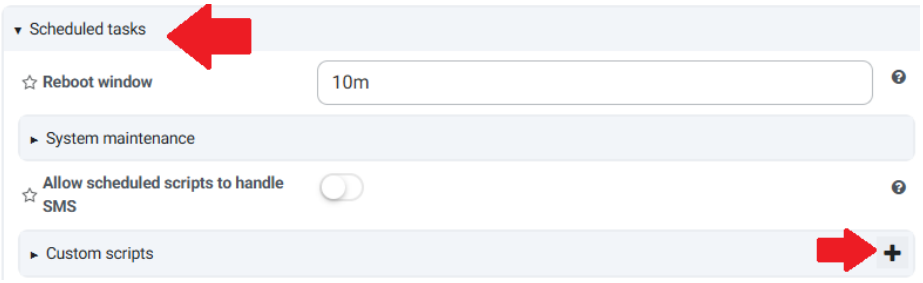
NOTE: As newer versions of the application are released, a new version of the mb_rm.zip file will need to be uploaded to the device. The same will be true if newer versions of the configuration files are also needed.

It is possible to use Templates within Digi RM to help automate pushing files to the Digi devices, along with configuration settings. Please see the Templates documentation for more information.

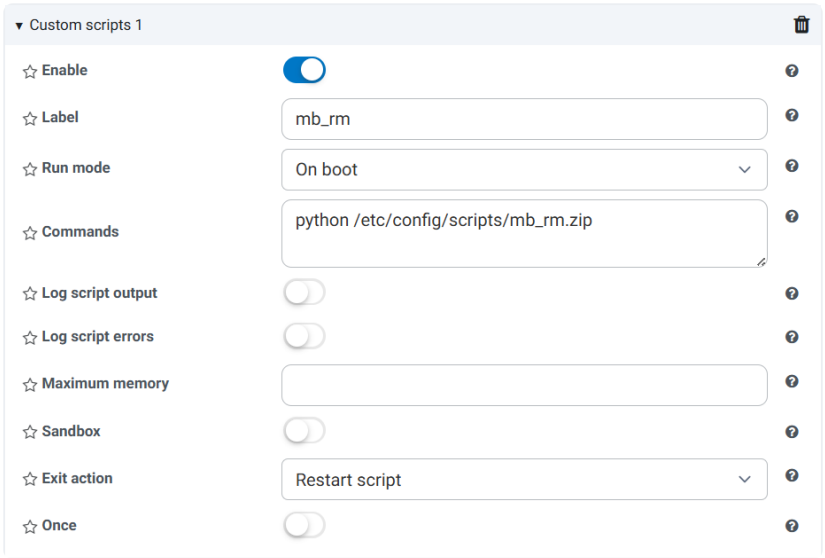
Setting the Application to Auto-Start

To ensure that the application starts when the Digi device is powered up initially, rebooted, or power cycled, these steps should be followed to enable the auto-start:

1) Within Digi RM, navigate to **Device Management > Devices**, and then click on the device that needs to be configured.
2) On the device's page, navigate to **Settings**, expand **Config**, and click on **System.**
3) Expand **Scheduled tasks** and then **click** on the **+ symbol** on the right-side of Custom scripts:



4) Once the new Custom script window opens, the settings will need to be configured to match what is shown below. Use the following screenshot as guidance for steps 5-12:



5) Toggle the **Enable** slider to enable the custom script.
6) Add a **Label** to identify the script.
7) Set the **Run mode** to **On boot**.
8) Type into the Commands field the following path to start the script:
   python /etc/config/scripts/mb_rm.zip
   NOTE: Additional arguments can be used on the command. See the next section for options.
9) Set the **Sandbox** slider to **disabled**.
10) Set the **Exit action** to **Restart script**.
11) Set the **Once** slider to **disabled**.
12) **(Optional)** If logging of the script output or errors is necessary, set the sliders to **Enabled** for **Log script output** and/or **Log script errors** respectively. This data will be sent to the Digi device's system log when enabled.
13) Click **Apply** to save and apply the changes.

It is possible to use Templates within Digi RM to help automate pushing configuration settings to the Digi devices. Please see the Templates documentation for more information.

Additional Parameters

There are 3 optional arguments that can be specified when running the application, which would be set during step 8 above. These additional parameters are:

1) -h, --help Show this help message and exit
2) -d, --debug Show extra output
3) -f, --file_log Turn on information and debug messages to persistent file logging
   a. If the -l parameter is not specified, the default location of the log file is stored to volatile memory (RAM). The default file name is **mb_rm_debug.log** and is located at **/logs** directory.
   b. **WARNING**: If using the -l option, a file will be written to Flash instead of to RAM. Excessive file logging to Flash memory may cause premature Flash wear. The file is located in the **/opt/app_log** folder on the local device, and under **/persistent files/app_log** in Digi RM.

servers.cfg - Configuration File

The servers.cfg file contains the information of the end Modbus servers that will communicate with the application. An example file is included in the files provided by Digi when the license for this application was purchased.

The following outlines the available parameters of the servers.cfg configuration file:

1) **[server1]/[server2]** – Configuration section name of specific, individual server. This will be used for the device ID when uploading register values.
2) **mode** – Mode of communication. The options are “tcp” or “rtu”.
3) **server_id** – MB ID of server device. This will be used when creating a read query to the device.
4) **address** – IP address of server. This is required for “tcp” mode only.
5) **port** – The port that will be used to connect to a Modbus device. If using “rtu” mode, this is the port name of the Digi Device serial adapter that is attached to the server (**Example**: /dev/serial/port1). If using “tcp” mode, this is the TCP port of the server (if unknown, use 502).
6) **register_map** – The file path of the register map file.
7) **little_endian** – Enables either little endian or big endian format. 0 = higher bits are in the lower register; 1 = lower bits are in the lower register. If this parameter is not defined in the servers.cfg file, the application will use the default of big endian (little_endian = 0).
8) **baud_rate** – The baud rate for a Modbus RTU connection (default 9600).

config.cfg - Configuration File

The config.cfg file contains the information for how often the application will poll for data and upload that data to Digi Remote Manager. An example file is included in the files provided by Digi when the license for this application was purchased.

The following outlines the available parameters of the config.cfg configuration file:

1) **[modbus_rm]** – The header of the configuration options section. This is a required field for the configuration file, and the application will not function without this field being present at the start of the configuration file.
2) **poll_frequency** – Time, in seconds. A required field, but only takes action if **force_upload** is set to true in the regs.csv file. When force_upload is set to true, the application will upload all registers that have this option enabled in the regs.csv file on the set interval.
3) **upload_on_change** – True/False. When True, the application will continuously scan the defined servers/registers and upload the items that changed. This is based off of 2 factors:
    a. **upload_threshold** – This parameter is within the regs.csv file. If **upload_threshold** is enabled in the regs.csv file, the value of this parameter will be the first priority in determining when the register would be uploaded. The register first needs to pass the threshold of this parameter, and then meet the timer of the **scan_interval** before the data is uploaded.
    b. **scan_interval** – This parameter is within the config.cfg file. If **upload_threshold** is not defined, this value takes priority for when a register should be uploaded based on if the register changed at all from the previous **scan_interval**.
4) **scan_interval** – Number of seconds to wait between scans for changes. This is only used when **upload_on_change** is set to true. Default is 5.
5) **force_upload** – comm_error=true/false; The application will send the message “comm_error = true” or “comm_error = false” when it can/cannot communicate with the end device. This message will be uploaded at the **poll_frequency** value.

regs.csv - Register Map CSV Columns

The resg.csv file is used by the application to map Modbus registers to devices. An example file is included in the files provided by Digi when the license for this application was purchased.

The CSV file must include the following headers (columns), in order from left to right:

1) **register_type**
2) **name**
3) **data_type**
4) **starting_address**

NOTE: If using the **data_type** of **Char**, the **length** value is also required.
NOTE 2: If using **reset_delay** and/or **data_factor**, additional values are required. See the section below for more information.

The following outlines the available parameters of the regs.csv configuration file:

1) **register_type** – Type of register that holds defined data. Options are 'coil', 'discrete_input', 'holding_register', and 'input_register'. The register_type value is required.
2) **name** – Name of datastream that will receive register data. This name will also be used when writing values. The name value is required.
3) **data_type** – Type of data contained in register. The options and the number of registers they will use are listed below:
    a. **Int (2)**
    b. **Float (2)**
    c. **Char (user defined)**
    d. **long_long (4)**
    e. **unsigned_long (2)**
    f. **unsigned_short (1)**
    g. **unsigned_long_long (4)**
    h. **short (1)**
    i. **double (4)**
    j. **long (2)**
    k. **unsigned_int (2)**
4) **starting_address** – Address of Modbus register that contains data item. The starting address is required.
5) **length** – Count of registers to poll to get entire data item. This field is required if **data_type = char**, otherwise the application will override this field with the # registers defined in the **data_type** description above. A value (empty value is acceptable) is required for this column if the **reset_delay** or **data_factor** are being used.
6) **reset_delay** – After writing a register, this value is the minimum number of seconds that will elapse before the register is set to '0'. Registers with a **reset_delay** will be set to '0' when the application starts. This column is required (default = 0 = no reset) If a data_factor is being used.
7) **data_factor** – Register write: value will be divided by this value before the register write is performed. Register read: value will be multiplied by this value before uploading the value to Digi Remote Manager. This column is not required. The default value is "1".
8) **upload_threshold** – The percentage that the value needs to change by before the register value is uploaded. Default is upload upon any value change.

9) **force_upload** – Always upload the Modbus register value at app config.cfg configured **poll_frequency**.
10) **alarm_low** – Force upload of the Modbus register value at the configured **poll_frequency** when the register value is less than or equal to this value. The default is no alarm.
11) **alarm_high** – Force upload of the Modbus register value at the configured **poll_frequency** when the register value is greater than or equal to this value. The default is no alarm.

Reading Modbus registers via Digi RM Device Request

The application supports the Digi Remote Manager SCI request "scan_now", triggering an upload of the registers configured as "force_upload" within the regs.csv file.

The values of the Modbus registers defined in the configuration files will be uploaded to Remote Manager DataStreams. The stream name will be based on the device ID of the Digi device and the server name. Routine polling will occur based on the "poll_frequency" configuration setting.

A poll can be initiated via the "scan_now" callback.

Example SCI call:

```
<sci_request version="1.0">
    <data_service>
        <targets>
            <device id="00000000-00000000-00409DFF-00000000"/>
        </targets>
        <requests>
            <device_request target_name="scan_now">name_of_device</device_request>
        </requests>
     </data_service>
 </sci_request>
```

The "device_id" should be your desired Digi Device ID and "name_of_device" should be replaced with the name of the device configured in servers.cfg.

Writing Modbus register via Digi RM Device Request:

The application supports the Digi Remote Manager SCI request “modbus_write”, which triggers the writing of a Modbus register.

Example SCI call:

```
<sci_request version="1.0">
    <data_service>
        <targets>
            <device id="00000000-00000000-00042DFF-FF000000"/>
        </targets>
        <requests>
                <device_request target_name="modbus_write">
                       name_of_server,name_of_data,5
                </device_request>
        </requests>
    </data_service>
</sci_request>
```

The text of the “device_request” element will be sent to the application. Three arguments, separated by commas, are required.

- **First argument**: Name of server, defined in servers.cfg
- **Second argument**: Name of data, defined in register map assigned to name of server
- **Third argument**: Value to write. Must be an integer or float value.

The response will indicate success or failure of the write in the “status” attribute of the “response” element.

Example:

Success – status=“response_received”:

```
<sci_reply version="1.0">
  <data_service>
    <device id="00000000-00000000-00042DFF-FF000000">
      <requests>
        <device_request target_name="modbus_write" status="0">
          <response status="response_received">100</response>
        </device_request>
      </requests>
    </device>
  </data_service>
</sci_reply>
```

This response indicates that a response was received from the Modbus server. The value received from the Modbus server is contained in the text of the “response” element. Confirm that this value is what you expect after writing the register/coil. The expected “response” from the server device include response = “register value” when writing a Holding Register of length 1 (short for example) and response

= “2” (word count or number of registers) when writing a Holding Register of length 2 (integer and float for example)

The following shows an example of a failed response to the device request:

Failure – status=”failed”

```
<sci_reply version="1.0">
  <data_service>
    <device id="00000000-00000000-00042DFF-FF076A5B">
      <requests>
        <device_request target_name="modbus_write" status="0">
          <response status="failed">Server not found: server1</response>
        </device_request>
      </requests>
    </device>
  </data_service>
</sci_reply>
```

If the “status” attribute of the “response” element is “failed”, one of the following error messages will be shown in the text of the “response” element.

- **Incorrect arguments** – The arguments supplied in the SCI request were not in the following format:
  - Name_of_server,name_of_data,5
- **Write failed** – The Modbus server write failed to successfully process.
- **Server not found** – The servers.cfg file does not contain information about the requested server name.
- **Failed to attempt write before timeout, resource busy** – The application was performing other IO operations before the task was processed within the 30 second timeout.

Known issues

None