Modbus I/O Installation Instructions for Digi DAL Routers

Change Log

Version	Notes	DAL OS Firmware Compatibility
1.1.4	Changed logging argument options and persistent file logging path changed from /opt/app/log to /opt/app_log	25.5.52.21
1.1.3	History registers improvements	25.5.52.21
1.1.2	Added capability to convert mV or mA values to business meaningful units ; Added historic value registers	25.5.52.21
1.1.0	Increased default poll interval from 0.5 to 2 seconds ; added configurable Endianness ; added app package.py	
1.0.4	Improved get AIN/DIO Name, it should work for all DAL IO devices up to Qty. 4 AIN and up to qty. 5 DIO ; Changed default mb server host IPv4 from 127.0.0.1 to 0.0.0.0	
1.0.3	Fixed incorrect log info message when mb_io.cfg file does not define dio1_counter	
1.0.2	Fixed typo: Namedio.dio1 listed twice in the get_dio_value index dictionary ; create the persistent logging folder if it's missing	
1.0.1	Added configurable poll interval ; pulse counter result stored in 32 bit integer qty. 2 registers ; updated documentation and code cleanup	
1.0.0	First release	

Contents

Introduction

Modbus I/O is a Python-based application that allows users to poll the Analog and Digital I/O ports of the Digi hardware as Modbus registers from their SCADA software.

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.

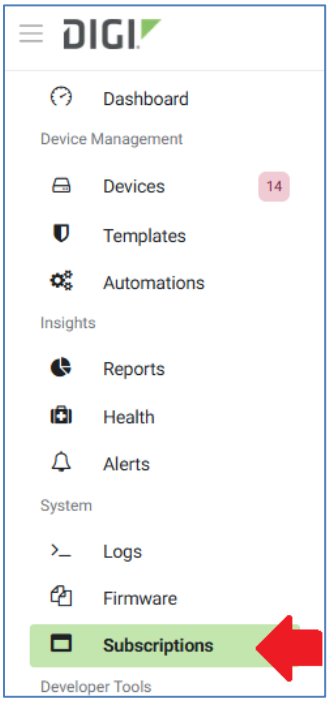
Installation and Auto-Start Instructions

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.

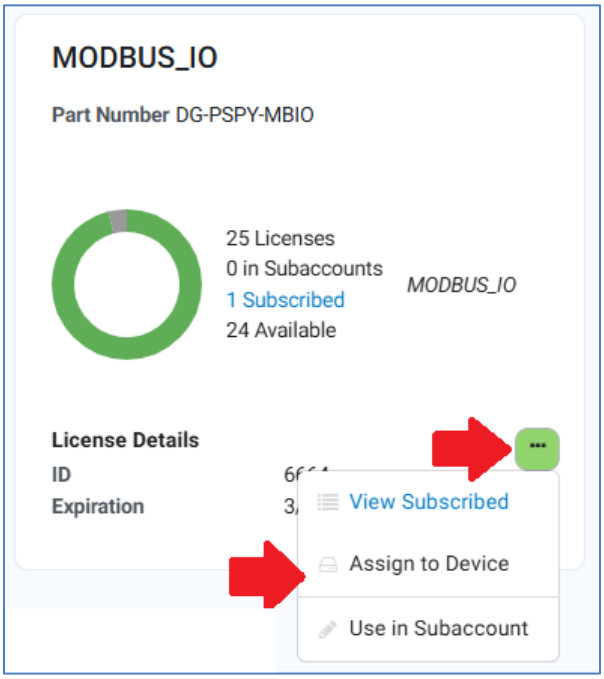
Installation

Apply the application subscription to the target device

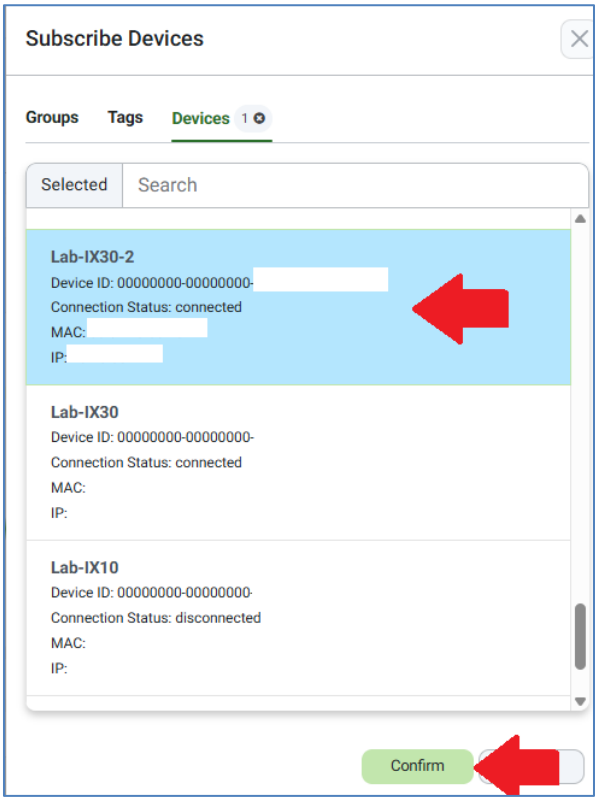
1) Log into Digi Remote Manager (Digi RM) using your credentials. If you do not have a Digi RM account, please purchase one by contacting Digi Sales or your Digi Authorized Reseller.
2) Register your Digi device(s) into Digi RM. For instructions on this process, please see this link.
3) Using the left-hand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



4) Locate the **MODBUS_IO** 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.



Upload the Configuration Files to the Digi Device

There are 2 files: 1 configuration file and 1 application file, that need to be installed onto the target Digi device. These files are:

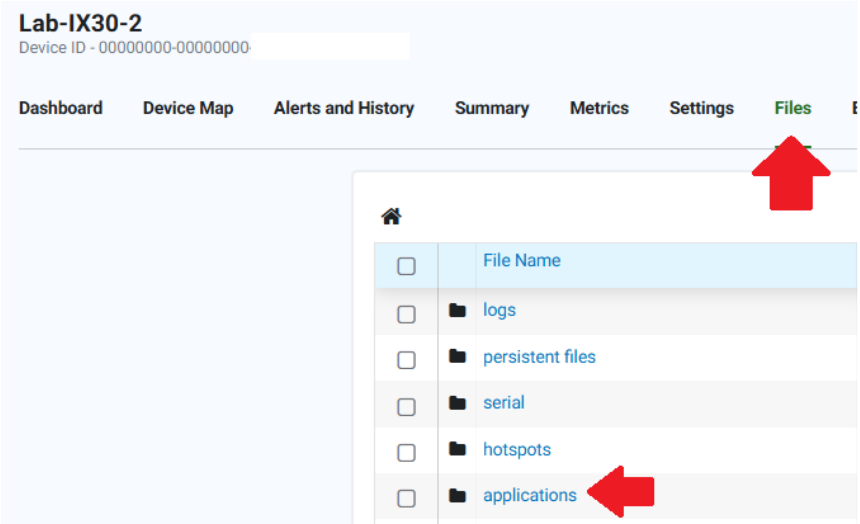
1) mb_io.zip
2) mb_io.cfg

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.

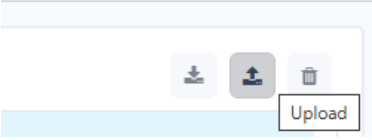
NOTE: Before uploading the mb_io.cfg file to the Digi device, ensure that the file has been configured in the desired manner for your use case. Sections further below in this manual describe how to configure the file if more information is needed.

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_io.py file will need to be uploaded to the device. The same will be true if newer versions of the configuration file are also needed.

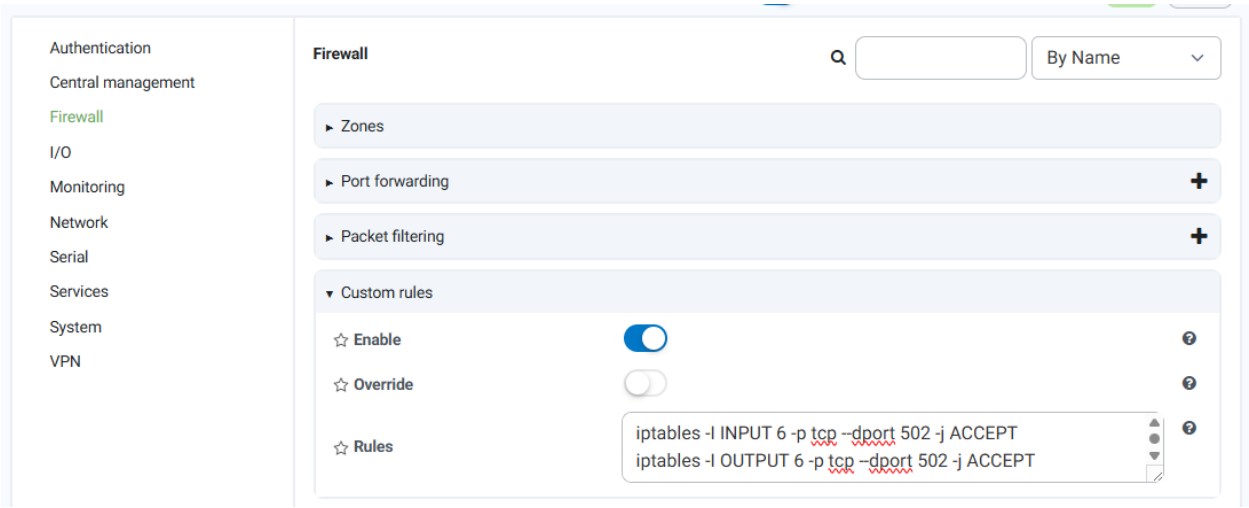
Add Firewall Rules to the Device

The Modbus I/O application needs a TCP port opened on the device's firewall to allow communication to work successfully. The following describes how to add the necessary rules to the device:

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** and then click on **Firewall**.
3) Expand Custom rules
4) Turn the toggle for Enable to the on position.
5) **DO NOT enable Override. Doing so will override all firewall rules on the device and will likely lock all access from the device and result in needing a factory default to gain access back to the device.**
6) Apply these 2 rules to the Rules area:
   iptables -I INPUT 6 -p tcp --dport 502 -j ACCEPT
   iptables -I OUTPUT 6 -p tcp --dport 502 -j ACCEPT

   NOTE: If a port other than 502 is being used by the application, use that port instead when applying the firewall rules to the device.

   This screenshot below shows what the default rule would look like on a device:



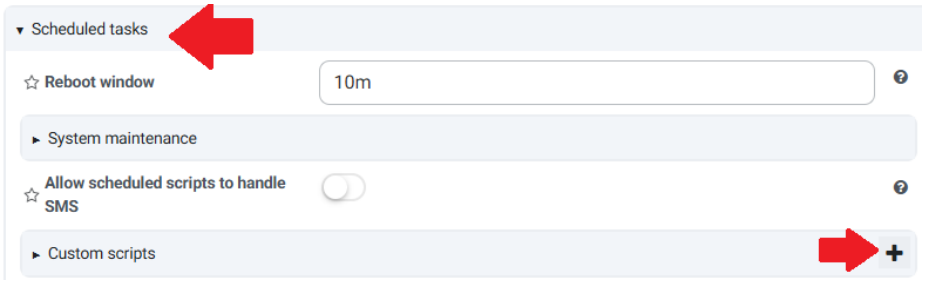
7) Click **Apply** to save and apply the changes.

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.

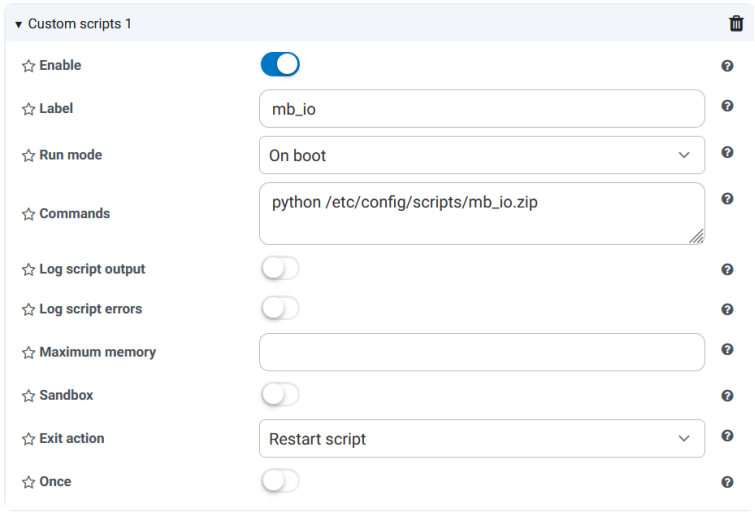
Setting the Application to Auto-Start

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** and then 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_io.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 8 optional arguments that can be specified when running the application, which would be set during step 8 in the section above. These additional parameters are:

1) **-h, --help**
    a. Show this help message and exit
2) **-H, --host HOST**
    a. Host (default: 0.0.0.0)
3) **-p, --port PORT**
    a. TCP port (default: 502)
4) **-i, --poll_interval POLL_INTERVAL**
    a. Time (seconds) between polling I/O and updating Modbus server register values (default: 2)
5) **-d, --debug**
    a. Enable debug mode
6) **-m, --max_log_file_size MAX_LOG_FILE_SIZE**
    a. Maximum size for this parameter is 4 (this equals 4MB).
7) **-l, --file_log**
    a. Turn on information and debug messages to persistent file logging
        i. 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_io.log** and is located at **/logs/** directory.
        ii. **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.'
8) **-le, --little_endian**
    a. Enables little endian format. If enabled, 32-bit registers least significant 16-bit register is first. If this parameter is not defined, the application will use the default of big endian.

mb_io.cfg - Configuration File

The mb_io.cfg file is a text file that the application will use for defining which registers will hold the Analog and Digital I/O values.

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

1) **[mb_regs]** – Header for the configuration file. This is a required field.
2) **ain[1-4] = [register]** – Big-endian float 32 bit holding register address representing ain1-4 voltage (V) or current (mA), slope, offset, adder, history registers % threshold
    a. **NOTE**: If the "history registers % threshold" is not defined, the default value is 1%.
3) **dio[1-4] = [register, bit]** – 16 bit holding register address used for stacked digital I/O values, register bit position (0 = LSB)
4) **dio1_counter = [register]** – To use this option, Digital I/O 1 must be in **Input** mode, and the **Pulse counter** option must also be enabled. Big- endian integer 32 bit holding register address used for digital DIO1 counter value (pulse counter). If **dio1_counter** is not defined, the digital value will be stored in Modbus holding register 10.
5) **watchdog_timer = [register]** – 16-bit register updated with IX30 uptime in seconds, from the start of device boot, and wraps back to 0 if the register's value exceeds 65535.
6) **history_registers = true/false** - Enable history registers. See next section for options.
    a. **NOTE**: The **history_registers** configuration option in the Modbus I/O Python application enables logging of historical changes in Analog Inputs (AIN) and Digital Inputs/Outputs (DIO). When enabled, this feature records time-stamped changes to Modbus server registers in memory, allowing for historical data tracking via Modbus.

       The current register and history register values are the transformed ainx value: (ain value + adder) * slope + offset

history_registers Details Explained

The history registers feature enables time-series tracking of Modbus I/O changes directly within the Modbus register map. It's a memory-efficient, cyclic buffer approach for both analog and digital signals, enabling downstream systems to query and reconstruct recent input history.

This feature allows you to query and reconstruct data within your data sets if there are gaps in your historical data sets due to items such as connectivity loss, network transmission issues, etc.

This section will cover the overall data structure of the history registers and logging of the data.

Data Structure

AIN (Analog Inputs)

1. Each AIN channel (1 to 4) is associated with the Current Value Register, which is configured by the `ain[1-4] = [register]` parameter in the mb_io.cfg file.
2. The History Storage Start Address stores the most recent history value register location (16-bit integer). The starting registers for each AIN are listed below:
   - AIN 1 = 1000
   - AIN 2 = 3000
   - AIN 3 = 5000
   - AIN 4 = 7000
3. The total number of registers the history can contain is up to 1998 per AIN channel.
4. The max history length of the historical data is 499 data sets. If the history goes beyond 499 data sets, the data will wrap back to the beginning of the history registers. The wrap indicator will be the “History Storage Start Address + 1”.
5. The data within each data entry will contain 4 registers:
   - 2 registers for timestamp (32-bit integer)
   - 2 registers for value (32-bit float)

See the table below for an example of AIN1’s history data sets:

Register	Purpose
1000	Current History Pointer (points to value)
1001	Wrap Status (0 or 1)
1002-1005	First entry (timestamp, value); **Example**: 1002-1003 are the timestamp and 1004-1005 are the value
1006-1009, 1010-1013, …	The pattern of subsequent entries; up to 499 entries

DIO (Digital Inputs/Outputs)

1. Each DIO channel (1 to 4) is associated with the Current Value Register, which is configured by the `dio[1-4]' register "bit" position parameter in the mb_io.cfg file.
2. The History Storage Start Address stores the most recent history value register location (16-bit integer). The starting registers for each DIO are listed below:
   - DIN1 = 8000
   - DIN2 = 8300
   - DIN3 = 8600
   - DIN4 = 8900
3. The total number of registers the history can contain is up to 302 per DIO channel.
4. The max history length of the historical data is 100 data sets. If the history goes beyond 100 data sets, the data will wrap back to the beginning of the history registers. The wrap indicator will be the "History Storage Start Address + 1".
5. The data within each data entry will contain 3 registers:
   - 2 registers for timestamp (32-bit integer)
   - 1 register for value (16-bit integer)

See the table below for an example of DIO1's history data sets:

Register	Purpose
8000	Current History Pointer
8001	Wrap Status (0 or 1)
8002-8004	First entry (timestamp, value) ; **Example**: 8002-8003 are the timestamp and 8004 is the value
8005-8007, 8008-8010, …	The pattern of subsequent entries; up to 100 entries

Web-Based Monitor Application

There is an optional, lightweight, web-based Modbus TCP monitor written in Python that can be installed along with the Modbus I/O application. It connects to the Modbus I/O application, reads the data from the Modbus I/O application, and displays the data on an authenticated web interface.

The web application is a separate installation from the core Modbus I/O application. For information on installing the web application, please see the Modbus I/O Web Application Installation Instructions.

Known issues

None