Control4 driver for Modbus TCP & RTU integration

Hop Soft Pty Ltd

2023-04-21

Contents

Description

This *Modbus TCP & RTU* driver enables two-way communication between Control4 systems and devices utilizing the de-facto industry-standard Modbus protocol. It supports both the **RTU** flavor for devices most commonly connected using the RS-485 or sometimes RS-232 interfaces, and the **TCP** flavor using standard TCP/IP network protocols. In either case, the driver always acts as the so-called *master* (or *client*), and communicates with a *slave* (or *server*) device.

More specifically this driver (along with a suite of accompanying drivers) has been successfully deployed in many Control4 projects to provide integration with various PLC (e.g. Wago, Tecomat Foxtrot, Amit), HVAC (Daikin, Mitsubishi), HRV (ATREA, Systemair), sauna & spa (USSPA, Hygromatik) and other special-purpose systems and controllers.

Successful integration requires obtaining the Modbus register documentation from the manufacturer of the device to be integrated, and creating a device-specific

data point file (register map) in the CSV format. This, and subsequent driver configuration, requires learning some basic Modbus concepts and terminology, and mastering some test utilities. See the linked resources and example CSV files. The fairly low-level nature of Modbus protocol may seem daunting for an average A/V integrator at first, but following some best practices and using the tools suggested in this document should make this process as easy as possible.

If you are not sure whether this solution is suitable for your use-case, please reach out to our support providing as much information & documentation as possible, and we will help you determine what can be achieved using the existing drivers or if any new software development is necessary. Sometimes a simple helper or intermediate driver is all that's needed. Other times a more complex tailor-made thermostat driver needs to be developed to provide the necessary user interface.

Supported features

- Modbus RTU & TCP protocols.
- Manipulating all Modbus data types (coils, discrete inputs, input registers, holding registers) using the following function codes (the driver automatically chooses the best option):
  - Read coils (FC 1)
  - Read discrete inputs (FC 2)
  - Read holding registers (FC 3)
  - Read input registers (FC 4)
  - Write single coils (FC 5)
  - Write single register (FC 6)
  - Write multiple coils (FC 15)
  - Write multiple registers (FC 16)
  - Mask write register (FC 22)
  - Read/write multiple registers (FC 23)
- Bindings to various Control4 device types (contact sensors, relays, switching lights, dimmers, blinds, thermostats etc.) for UI visualization as well as raw value manipulation via accompanying device drivers.
- The ability to pack/unpack single-bit values to/from 16-bit registers.
- The ability to use 32-bit and 64-bit floating point numbers and (signed and unsigned) integers (utilizing 2 or 4 consecutive standard Modbus registers)
- Periodic polling (reading of Modbus registers).
- Various endiannes options.

Limitations

- It's not possible to upload the CSV file using the Config page remotely over Remote System / Enhanced Connection. Utilizing a full-featured VPN connection should work.
- Modbus slave/server mode is not supported, the driver always acts a master/client.
- If Modbus RTU is used over Control4 or Global Caché serial ports, the driver always configures them to 9600 bauds, 8 data bits, no flow control, 1 stop bit, no parity. If you use any 3rd-party *serial to IP converter*

with a driver that does not change the serial settings in the converter automatically, it doesn't matter which serial settings the Modbus RTU device actually uses to communicate.
- Modbus ASCII or any other protocol variant or proprietary extension is not supported.
- Multiple RS-485 RTU slaves over a single serial connection are not supported natively (a multiplexer driver is required).
- Modbus exceptions (error codes) in the driver's logs are not translated to human-readable names.
- Only the standard TCP port 502 is supported for Modbus TCP.
- Packing/unpacking nibbles (4-bit values) is currently not supported.

Licensing & trial

This driver is licensed through the driverCentral Cloud driver and does not function without it. Make sure you have added it to the project and that it's been successfully registered with the licensing server. Please read the Cloud driver's documentation if you are unfamiliar with the driverCentral licensing process.

The usual, once-per-system 21-day trial period and showroom licensing are available, to allow you to properly evaluate the driver and test the integration with Modbus devices.

Installation

Before you start

First you should familiarize yourself with some basic Modbus terms: Modbus TCP vs RTU, address vs offset, function code vs exception response, coils (1 bit, read/write), discrete inputs (1 bit, read-only), holding registers (2 bytes / 16 bits / word, read/write), input registers (2 bytes / 16 bits / word, read-only) etc. The links in the Modbus Resources & Tools section will help you with this, especially About Modbus (Frequently Asked Questions) by Simply Modbus is an invaluable Modbus primer.

Every integration starts with obtaining the correct Modbus register documentation from the manufacturer. Once you get that, before you even start creating the CSV file and configuring the Modbus driver, it's **absolutely necessary** to try reading and then writing some registers using some Modbus master (scanner, poller, client) software. This will help you verify that the documentation provided by the manufacturer is correct and that you have good a understanding of it. Control4 system and the Modbus driver add several layers of complexity that make troubleshooting problems that may arise right at the beginning painfully difficult for integrators (for those unfamiliar with Modbus in general, and the driver in particular, doubly so).

If you are integrating Modbus RTU devices you should *definitely* get a **USB to RS-485** converter (or a combo of USB to RS-485 and RS-485 to RS-232 converters which may be more universal) to be able to check the communication from your PC.

Considerations

- If using Modbus RTU, identify the correct serial protocol/interface (usually RS-485, rarely RS-232) and communication parameters (baud rate, data bits, stop bits, flow control, parity).
- *Slave ID* or Modbus device ID. Necessary for RTU devices (to differentiate multiple slaves on a single RS-485 bus); TCP devices often (but not always) ignore it. The device may have a hard-coded slave ID, or you may be able to change it via DIP switches, configuration menu if the devices has a display, or even via one of the Modbus registers.
- Data point *addresses* (or offsets) are usually the most confusing part while integrating Modbus devices. Manufacturers (of both hardware and software) use different ways or systems to number the data points.
  - The driver's data point definition CSV file is using a 1-based addressing in the `ADDRESS` column, i.e., the lowest possible value you'd enter in that column for any type would be *1* (decimal). Incidentally you'd use the same numbering in CAS Modbus Scanner, although the software incorrectly calls it *offset*.
  - You can come across a zero-based addressing (or offsets) and it can be specified using either decimal or hexadecimal numbers. You will need to convert hexadecimal to decimal (if needed) and add 1 to get the correct value to enter in the `ADDRESS` column in the CSV file for each data point.
  - There's the legacy Modicon convention using special prefixes to denote not only the address of the data point, but its type too. You will need to subtract this type prefix to get the correct value to enter in the `ADDRESS` column. The prefix is
    - 0 for coils,
    - 10000 or 100000 for discrete inputs,
    - 30000 or 300000 for input registers,
    - 40000 or 400000 for holding registers.
- Supported Modbus *function codes* (abbreviated as FC). It's absolutely necessary to only enable the supported FCs in the driver, otherwise it might use function codes not supported by the device (they may be considered to be more effective for the given task by the driver) and the operation (read or write) would fail. By default, all functions codes supported by the driver are enabled, but *Modbus: Mask write register (FC 22)* and *Modbus: Read/write multiple registers (FC 23)* are actually almost never implemented by Modbus devices so you may start by setting these properties to *Not supported*. Be aware that some manufacturers' docs represent the FCs as hexadecimal numbers while others use decimal numbers (the driver uses the latter). Convert as necessary using e.g. Windows Calculator's Programmer mode.
- Sometimes devices use read-write types (coils or holding registers) to hold what's actually read-only information or stores values such as temperetures which the driver implicitly treats as holding registers (see the TEMP type definition). You can utilize the `LOCATION` column to override this implicit behavior.
- Sometimes devices use (2-byte) registers to represent single-bit values or even pack multiple 1-bit values into one register. The driver uses the

`LOCATION` and `BIT` columns in the CSV file to support that.

- Sometimes devices don't allow reading (some) holding registers (i.e., they are effectively write-only). For such cases you may need to set the value in the `READ_FACTOR` column to 0 (zero) and possibly the *Modbus Read Blocks* property to *Precise*.
- While according to the Modbus protocol specification (when you consider the maximum message size, protocol overhead etc.) it should be possible to read up to 2008 coils or discrete inputs, or 125 holding or input registers with a single request, in reality some devices (e.g. due to their memory limitations) return errors if you try to read more than a much lower number of values. The driver currently does not support limiting the number of requested values and it reads as many as possible with as few requests as possible. E.g., if you specify only registers 1 and 20 in the CSV file, and the FCs 1-4 to read multiple values are supported, the driver will by default read the whole block of registers 1 to 20 because it's more efficient. In most cases you can just set the *Modbus Read Blocks* property to *Precise* to disable this optimization, at the cost of possibly making read requests less efficient overall. Please note that the driver will still read entire blocks of consecutive registers if they are all specified in the CSV file.
- Usually the values in coils & discrete inputs or holding registers & input registers are completely separate in Modbus devices, so e.g. holding register 4 contains a different value than input register 4 and the driver can only e.g. read input registers using FC 4. Yet in some devices the read-only and read/write values actually share the same address space so e.g. reading register address 4 using either FC 3 or FC 4 returns the value of the same data point. This can be leveraged to optimize the polling process even if some of the addresses contain read-only values (and the Modbus device would either ignore attempts to rewrite them, or return an exception), just make sure to specify 1 (for coils) or 4 (for holding registers) in the `LOCATION` in the CSV file and the driver will read these data points using the appropriate function codes for coils or holding registers respectively.
- Keep in mind that reading or writing data can fail for a plethora of reasons: a hardware issue such as RS-485 wired incorrectly, wrong serial settings, incorrect slave ID, unsupported or wrong function code, illegal data address (even if it's just a part of a range you are trying to read) to list a few, not to mention various Control4-related issues. Trial and error troubleshooting is always much quicker using some Modbus scanner software directly on your PC rather than trying to figure the problem out using the driver. Try reading a single data point at a time, shifting addresses by one in either way, switching the RS-485 A/B wires around etc.

CSV file data point definition format

Creating the CSV file using LibreOffice Calc is generally the easiest way. You can specify the exact CSV format and character encoding when loading and saving the file. Microsoft Excel can be a bit tricky. Depending on your operating system locale, it can save the file using semicolons (;) instead of commas (,) to delimit columns and the driver would fail to parse such file. Of course you can opt for an advanced text editor such as Visual Studio Code (with great Rainbow CSV and/or Edit csv extensions) or Notepad++ to edit the file by

hand. Microsoft Notepad would suffice at a pinch too. Definitely don't use any word processor such as Microsoft Word, LibreOffice Write or even Microsoft WordPad to make changes to the file, as these programs don't produce plain-text files.

- It's a plain-text ASCII or UTF-8 (if special characters are used) file. The UTF-8-encoded file must not start with a BOM (especially pay attention to this when using Microsoft Notepad).
- The columns must be delimited using the comma (,) character. No other characters (semicolons, tabs etc.) are supported.
- The first line must contain a header.
- The recommended header definition is `NAME,TYPE,ADDRESS,BIT,LOCATION,LIST_VALUES,READ_FACTOR,COMMENT`.
- Only the first three columns (`NAME,TYPE,ADDRESS`) are mandatory and as such must always contain values for each data point.
- `LOCATION` is optional. Implicit values are used based on TYPE definition. 0 - coils, 1 - discrete inputs, 3 - input registers, 4 - holding registers
- `BIT` is optional. Used for specifying position of bit values in registers.
- `LIST_VALUES` is optional. Semicolon separated list of key/value mappings. Example: `Auto: 0; Heat: 1; Fan: 2; Cool: 3; Dry: 4`
- `READ_FACTOR` is optional. The implicit value is 1. 0 - never, 1 - every read cycle, 2 - every other read cycle, ...
- Anything between `/*` and `*/` or `(*` and `*)` or after `//` or `--` is treated as a comment and thus ignored while parsing the file. Using the `COMMENT` column is cleaner and more convenient when editing the file in a spreadsheet/table mode though.
- Columns with unrecognized names are ignored by the driver. That actually includes the `COMMENT` column suggested above.

Supported types

- `LIGHT` - 1 bit, coils, switch/relay light, `GEN_SWITCH`
- `RELAY` - 1 bit, coils, relay, `RELAY`
- `CONTACT` - 1 bit, discrete inputs, digital input, `CONTACT_SENSOR`
- `DIMMER` - 1 register, value in range 0 - 100, holding registers, dimming light, `GEN_DIMMER`
- `DIMMER_ADV` - 1 register, low byte - dimming value in range 1 - 100, high byte - bit 0: 0 = off, 1 = on, bits 1-7: ramp rate in hundreds of milliseconds (eg. 3 = 300 ms, 600 = 1 minute), holding registers, dimming light, `GEN_DIMMER`
- `DIMMER_FLOAT` - 2 registers, floating point number, value in range 0 - 100, negative value = switch off, holding registers, dimming light, `GEN_DIMMER`
- `BLIND` - 6 registers, holding register, blind with slats, `GEN_BLIND`, registers:
  - 1: target position, 0 - 100, 0 = closed, 100 = open, 255 = unknown
  - 2: ramp rate, time in tens of milliseconds before target position is reached, 0 = stopped
  - 3: target slats rotation, 0 - 100, 0 = closed, 100 = open, 255 = unknown
  - 4: slats ramp rate, time in tens of milliseconds before target position is reached, 0 = stopped
  - 5: Control4 writes target position, 0 - 100, 255 = stop right now

  - 6: Control4 writes target slats rotation, 0 - 100
- `BLIND_FLOAT` - 11 registers, holding register, blind with slats, `GEN_BLIND`, registers:
  - 1-2: Control4 writes target position, floating point number, 0 - 100
  - 3-4: Control4 writes target slats rotation, floating point number, 0 - 100
  - 5-6: target position, floating point number, 0 - 100, 0 = closed, 100 = open, 255 = unknown
  - 7-8: target slats rotation, floating point number, 0 - 100, 0 = closed, 100 = open, 255 = unknown
  - 9: Control4 writes to stop movement, 1 = stop right now
  - 10: ramp rate, time in tens of ms before target position is reached, 0 = stopped
  - 11: slats ramp rate, time in tens of ms before target position is reached, 0 = stopped
- `THERMOSTAT` - 13 registers, holding register, dual setpoint thermostat with humidity, `THERMOSTAT_LINK`, registers:
  - 1: current temperature, signed integer, read only, temperature × 10
  - 2: heat setpoint, signed integer, read & write, temperature × 10
  - 3: cool setpoint, signed integer, read & write, temperature × 10
  - 4: HVAC power, unsigned integer - On: 1; Off: 0
  - 5: HVAC mode, unsigned integer - Off: 0; Heat: 1; Cool: 2, Auto: 3
  - 6: HVAC state, unsigned integer - Off: 0; Heat: 1; Cool: 2
  - 7: fan mode, unsigned integer - Off: 0; Low: 1; Medium: 2; High: 3, Auto: 4
  - 8: fan state, unsigned integer - On: 1; Off: 0
  - 9: current humidity, unsigned integer, read only, 0 - 100
  - 10: humidify setpoint, unsigned integer, read & write, 0 - 100
  - 11: dehumnidify setpoint, unsigned integer, read & write, 0 - 100
  - 12: humidify mode, unsigned integer - Off: 0; Humidify: 1; Dehumidify: 2; Auto: 3
  - 13: humidify state, unsigned integer - Off: 0; Humidifying: 1; Dehumidifying: 2
- `THERMOSTAT_FLOAT` - 20 registers, holding register, thermostat with humidity, `THERMOSTAT_LINK`, registers:
  - 1-2: current temperature, float, read only, temperature
  - 3-4: heat setpoint, float, read & write, temperature
  - 5-6: cool setpoint, float, read & write, temperature
  - 7-8: single setpoint, float, read & write, temperature
  - 9: HVAC mode, unsigned integer - Off: 0; Heat: 1; Cool: 2, Auto: 3
  - 10: fan mode, unsigned integer - Off: 0; Low: 1; Medium: 2; High: 3, Auto: 4
  - 11-12: current humidity, float, read only, 0 - 100
  - 13-14: humidify setpoint, float, read & write, 0 - 100
  - 15-16: dehumnidify setpoint, float, read & write, 0 - 100
  - 17: humidify mode, unsigned integer - Off: 0; Humidify: 1; Dehumidify: 2; Auto: 3
  - 18: HVAC state, unsigned integer - Off: 0; Heat: 1; Cool: 2

  - 19: fan state, unsigned integer - On: 1; Off: 0
  - 20: humidify state, unsigned integer - Off: 0; Humidifying: 1; Dehumidifying: 2
- `TEMP` - 1 register, float, holding registers, read only, temperature, `THERMOMETER`
- `TEMP_10` - 1 register, signed integer, holding registers, read only, temperature × 10, `THERMOMETER`
- `TEMP_100` - 1 register, signed integer, holding registers, read only, temperature × 100, `THERMOMETER`
- `TEMP_FLOAT` - 2 registers, floating point number, holding registers, read only, temperature, `THERMOMETER`
- `SETPOINT` - 1 register, float, holding registers, read & write, temperature, `SETPOINT`
- `SETPOINT_10` - 1 register, signed integer, holding registers, read & write, temperature × 10, `SETPOINT`
- `SETPOINT_100` - 1 register, signed integer, holding registers, read & write, temperature × 100, `SETPOINT`
- `SETPOINT_FLOAT` - 2 registers, floating point number, holding registers, read & write, temperature, `SETPOINT`
- `LIST` - 1 register, unsigned integer, holding registers, read & write, `STRING_VARIABLE`
- `LIST_C` - 1 register, low byte - list value, high byte - set to 1 on write from driver, reset to 0 by PLC, holding registers, read & write, `STRING_VARIABLE`
- `VAR_STRING` - variable length, null-terminated string, holding registers, UTF-8 encoded text, `STRING_VARIABLE`
- `VAR_FLOAT` - 2 registers, holding registers, floating point number, `NUMBER_VARIABLE`
- `VAR_DOUBLE` - 4 registers, holding registers, floating point number, `NUMBER_VARIABLE`
- `VAR_UINT16` - 1 register, holding registers, unsigned integer, `NUMBER_VARIABLE`
- `VAR_INT16` - 1 register, holding registers, signed integer, `NUMBER_VARIABLE`
- `VAR_UINT32` - 2 registers, holding registers, unsigned integer, `NUMBER_VARIABLE`
- `VAR_INT32` - 2 registers, holding registers, signed integer, `NUMBER_VARIABLE`
- `VAR_UINT64` - 4 registers, holding registers, unsigned integer, `NUMBER_VARIABLE`
- `VAR_INT64` - 4 registers, holding registers, signed integer, `NUMBER_VARIABLE`
- `VAR_TIMESTAMP` - 4 registers, holding registers, signed 64-bit integer UNIX time, `NUMBER_VARIABLE`

The types such as `DIMMER`, `DIMMER_ADV`, `DIMMER_FLOAT`, `DIMMER_FLOAT`, `BLIND`, `BLIND_FLOAT`, `THERMOSTAT`, `THERMOSTAT_FLOAT`,`LIST_C` are generally only useful with PLCs where the function blocks can be tailor-made to fit these types.

Example CSV file in plain-text form

```
NAME,TYPE,ADDRESS,BIT,LOCATION,LIST_VALUES,READ_FACTOR,COMMENT
Light in Coil 1,LIGHT,1,,,,,
Motor in Coil 2,RELAY,2,,,,,
Button in Discrete Input 1,CONTACT,1,,,,,
```

```
Dimmer in Register 15,DIMMER,15,,,,,
Temperature in Register 16,TEMP,16,,,,,
Temperature × 10 in Register 17,TEMP_10,17,,,,,235 = 23.5
Temperature × 100 in Register 18,TEMP_100,18,,,,,235 = 2.35
Setpoint in Register 124,SETPOINT,124,,,,,
Setpoint × 10 in Register 125,SETPOINT_10,125,,,,,
Setpoint × 100 in Register 126,SETPOINT_100,126,,,,,
Float,VAR_FLOAT,18,,,,,Float takes up two registers
Double,VAR_DOUBLE,20,,,,,Double takes up four registers
Unsigned integer 16b,VAR_UINT16,24,,,,,
Signed integer 16b,VAR_INT16,25,,,,,
Unsigned integer 32b,VAR_UINT32,26,,,,,
Signed integer 32b,VAR_INT32,28,,,,,
Unsigned integer 64b,VAR_UINT64,30,,,,,
Signed integer 64b,VAR_INT64,34,,,,,
/* Examples of relays in holding registers */
/* Location = 4 = Holding Register */
Relay 1 in Holding Register 1,RELAY,1,0,4,,,
Relay 2 in Holding Register 1,RELAY,1,1,4,,,
Relay 16 in Holding Register 1,RELAY,1,15,4,,,
/* Examples of binary sensors in input registers */
/* Location = 3 = Input Register */
Sensor 1 in Input Register 1,CONTACT,1,0,3,,,
Sensor 2 in Input Register 1,CONTACT,1,1,3,,,
Sensor 16 in Input Register 1,CONTACT,1,15,3,,,
```

CSV in LibreOffice Calc

Make sure to select only *Comma* as a separator. Check the Fields window to see if the CSV file is going to be imported fine before pressing OK.

Driver installation

1. Copy automation_ip_modbus_plc.c4z and the other accompanying drivers to your local driver database, i.e., typically Documents\Control4\Drivers\.
2. Backup the project.
3. Add & configure the DriverCentral Cloud driver if not already in the project.
4. Assign the *Modbus TCP & RTU PLC* driver licence to the project on the DriverCentral portal.
5. Add the Modbus driver to the project via the Search tab.

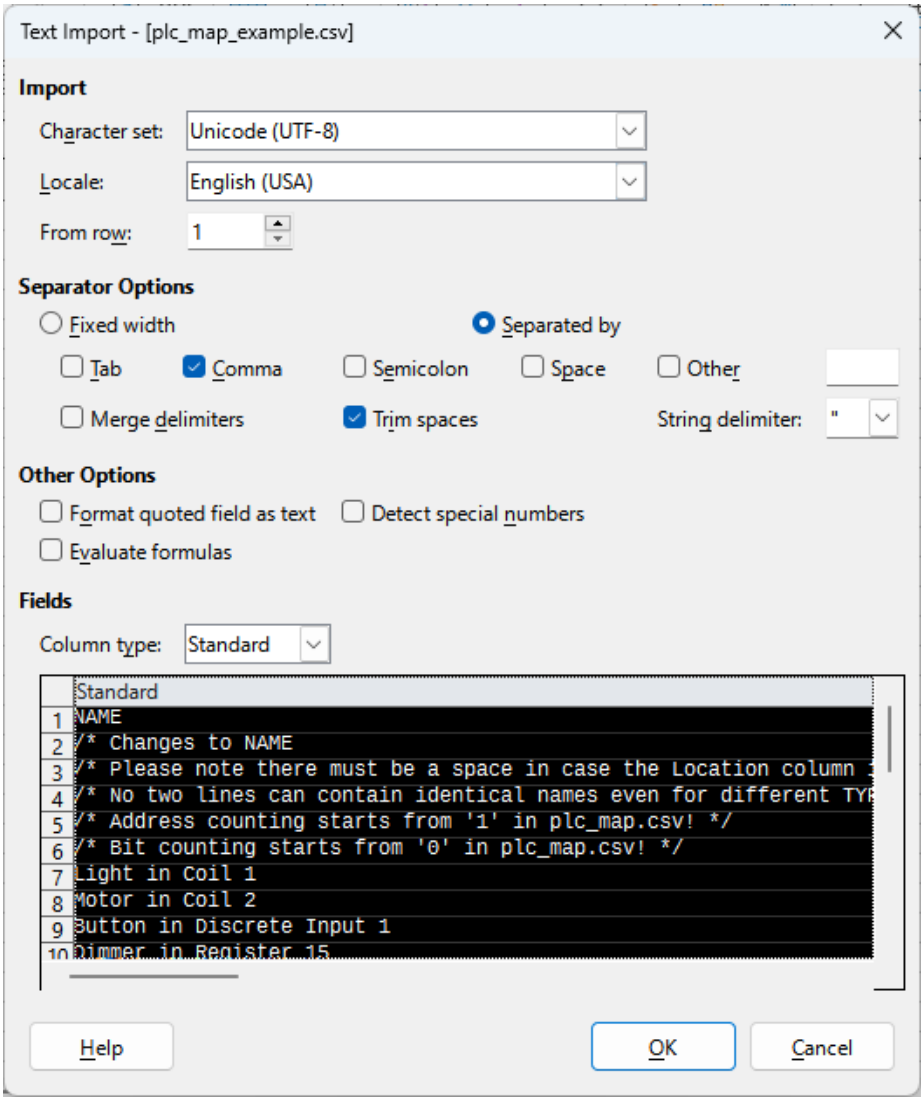
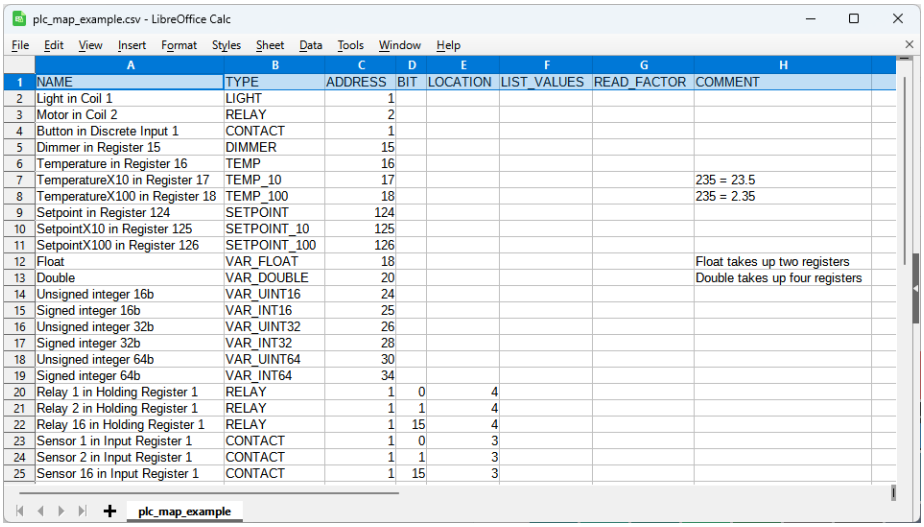
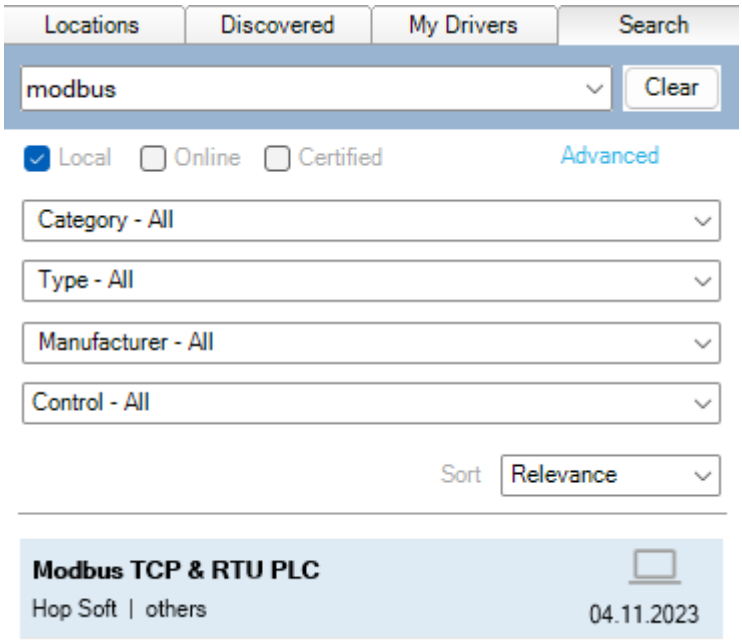


Figure 1: Importing a CSV file into LibreOffice Calc



	A	B	C	D	E	F	G	H
1	NAME	TYPE	ADDRESS	BIT	LOCATION	LIST_VALUES	READ_FACTOR	COMMENT
2	Light in Coil 1	LIGHT	1					
3	Motor in Coil 2	RELAY	2					
4	Button in Discrete Input 1	CONTACT	1					
5	Dimmer in Register 15	DIMMER	15					
6	Temperature in Register 16	TEMP	16					
7	TemperatureX10 in Register 17	TEMP_10	17					235 = 23.5
8	TemperatureX100 in Register 18	TEMP_100	18					235 = 2.35
9	Setpoint in Register 124	SETPOINT	124					
10	SetpointX10 in Register 125	SETPOINT_10	125					
11	SetpointX100 in Register 126	SETPOINT_100	126					
12	Float	VAR_FLOAT	18					Float takes up two registers
13	Double	VAR_DOUBLE	20					Double takes up four registers
14	Unsigned integer 16b	VAR_UINT16	24					
15	Signed integer 16b	VAR_INT16	25					
16	Unsigned integer 32b	VAR_UINT32	26					
17	Signed integer 32b	VAR_INT32	28					
18	Unsigned integer 64b	VAR_UINT64	30					
19	Signed integer 64b	VAR_INT64	34					
20	Relay 1 in Holding Register 1	RELAY	1	0	4			
21	Relay 2 in Holding Register 1	RELAY	1	1	4			
22	Relay 16 in Holding Register 1	RELAY	1	15	4			
23	Sensor 1 in Input Register 1	CONTACT	1	0	3			
24	Sensor 2 in Input Register 1	CONTACT	1	1	3			
25	Sensor 16 in Input Register 1	CONTACT	1	15	3			

Figure 2: A CSV file open in LibreOffice Calc



6. Check that the Cloud Status property reads *License Activated* (or *Trial Running* with the remaining time).
7. Configure the IP address to connect to in Connections → Network → IP Network for Modbus TCP devices, or bind the serial connection in Connections → Control/AV for Modbus RTU devices.

Driver setup

After having read the device manufacturer's Modbus register manual (integration guide etc.) and installing the driver you can proceed with configuring the driver.

Properties

- **Config Import Port** (9000): Only increment if you are using multiple instances of the Modbus driver in the project or in case there's a conflict with another driver. Each instance of the driver needs a unique port to listen on. The driver needs to try each port in sequence when you execute the *Config Import* action, so the Config Import interface will take longer to initialize on drivers with higher port number.
- **Address** (255): Corresponds to the Slave or Device ID of your Modbus slave device. Refer to the device documentation on how to obtain or change this ID. Generally values 1-247 are used with Modbus RTU devices. Address 0 is a broadcast address and 248-255 are reserved for other purposes.
- **Modbus Read Delay [in ms]** (100): How often the driver polls the slave device. Generally, lower value means Control4 will get changes quciker, but it also causes higher load on the Control4 controller and the slave device itself which may lead to overall poor end user experience due to slowness in other parts of the system, lost messages etc. Generally 100 milliseconds is the lowest useful value but often 250 ms or more is used. You also need to take into consideration how many read requests the driver needs to do each cycle to obtain values of all the data points specified in the CSV file. If the delay is set to 250 ms and 10 requests are needed to read everything then the whole cycle will take 2.5 seconds. You can optimize this by setting READ_FACTOR to 0 for data points which are write-only, and if you are programming a PLC, you can further optimize this by lumping all the data points as closely as together, even using read/write coils or holding registers to represent read-only values.
- **Modbus Read Blocks** (Can include space): By default the driver optimizes read requests by trying to read as many data points as possible, including the ones inbetween, not specified in the CSV file, with a single request. This may cause issues with some devices where the intermediate addresses are actually illegal, or the device cannot handle as many data points in a single reply. Setting this property to *Precise* will force the driver to read only the data points specified in the CSV file.
- **PLC: Register Endianness** (Big endian): This has effect on how registers are ordered if the non-standard 2- and 4-register (or 32- and 64-bit) types (TEMP_FLOAT, VAR_FLOAT, VAR_DOUBLE, VAR_UINT32 and similar) are used. Systemair SAVE and Tecomat Foxtrot are devices known to order their registers the *Little endian* way. If the device documentation doesn't say specifically, try experimenting with this using a Modbus scanner software until you get sensible values.
- **PLC: Byte Endianness** (Big endian): This has effect on how individual bytes are ordered in registers. Spec-compliant Modbus devices are always big-endian.
- **Modbus Functions Codes 1-23** (Supported): The driver normally prefers the more effective functions, e.g. Write multiple registers (FC

16) over Write single register (FC 6), which may be unsupported by the device, so this needs to be set accordingly based on the manufacturer's documentation.

Actions

- **Enable Config Import** enables file upload on the *Config Import* tab.
- **Delete Connections** clears all connections on the Modbus driver.
- **Debug Info** prints debugging information useful for advanced troubleshooting in the Lua Output
- **Print startup log** prints log messages from the driver's initialization if *Logger: Keep Startup Log* was set to *Keep* **before** director restart.

Config Import

You need to upload your CSV file through this panel. The driver immediately updates (creates, deletes, renames) the connections.

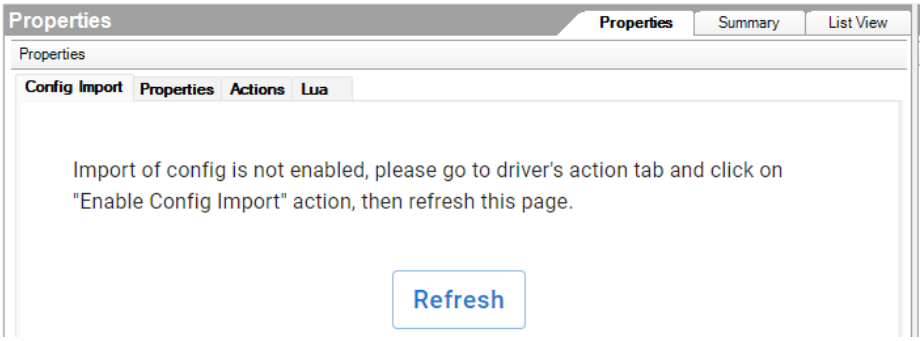


Figure 3: Waiting to Enable Config Import

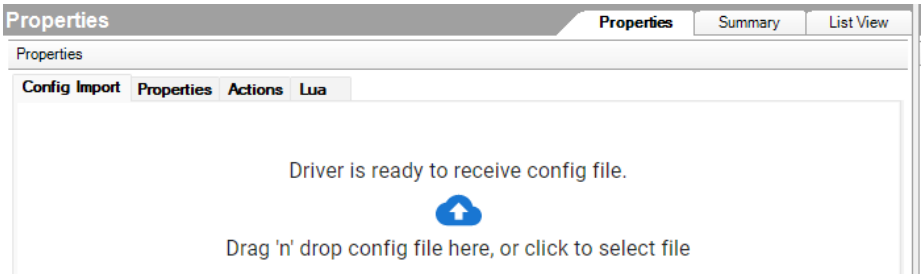


Figure 4: Ready to upload the CSV config file

Connections

The driver creates the following connection classes based on the data point definitions in the CSV file, which can be bound to various drivers to provide the appropriate user interface, or a programming interface in the form of events, actions and device variables.

- `CONTACT_SENSOR` binds to the standard Control4 generic *Sensors* drivers such as Contact Switch, Door Contact Sensor, Motion Sensor etc.
- `RELAY` binds to the bundled `Relay Light` driver (`light_relay.c4i`) and to the standard Control4 generic *Motorization* drivers such as Door Lock, Fan, Pump etc.
- `GEN_SWITCH` binds to the bundled *Switching Light* driver (`light_generic.c4i`).
- `GEN_DIMMER` binds to the bundled *Dimming Light* driver (`light_gen_dimmer.c4i`).
- `GEN_BLIND` binds to the bundled *PLC Blind* and *PLC Blind with Louvers* drivers (`blind_plc.c4z`, `blind_plc_slats_control.c4z`).
- `THERMOSTAT_LINK` binds to the bundled *Software Thermostat V3 (Split Setpoints)*, *Software Thermostat V3 (Single Setpoint)* and *Temperature Display* drivers (`thermostat_sw_thermostat.c4z`, `thermostat_sw_thermostat_single.c4z`, `temperature_display.c4i`).
- `NUMBER_VARIABLE` binds to the bundled *Number Variable* driver (`variable_number.c4i`) (no user interface) and to the bundled thermostat drivers to send or receive relative humidity values and setpoints.
- `STRING_VARIABLE` binds to the bundled thermostat drivers to pass e.g. HVAC Mode or Fan Mode names to the Modbus driver where they get translated to integer values based on the associated `LIST_VALUES` column key/value mapping.
- `THERMOMETER` binds to the bundled thermostat drivers to send current temperature to them.
- `SETPOINT` binds to the bundled thermostat drivers to send and receive temperature setpoints.

Example: HVAC integration

This is an example of connections between the Modbus and Software Thermostat (Split Setpoints) driver. Study the plc_map_example_hvac.csv file to see how the connections were specified. The example HVAC device has a separate register for power (On/Off) control. Some devices turn on automatically by selecting a HVAC Mode (e.g., Heat), and the Off mode is just another value (usually `0`) in the same HVAC Mode register.

	A	B	C	D	E	F	G	H
1	NAME	TYPE	ADDRESS	BIT	LOCATION	LIST_VALUES	READ_FACTOR	COMMENT
2	Room HVAC Power	LIST	1			On: 1; Off: 0		
3	Room HVAC Mode	LIST	2			Auto: 0; Heat: 1; Cool: 2		
4	Room HVAC State	LIST	3			Off: 0; Heat: 1; Cool: 2		
5	Room Fan Mode	LIST	4			Auto: 0; Low: 1; Medium: 2; High: 3		
6	Room Fan State	LIST	5			On: 1; Off: 0		
7	Room Current Temperature	TEMP_10	6					
8	Room Heat Setpoint	SETPOINT_10	7					
9	Room Cool Setpoint	SETPOINT_10	8					
10	// For (Optional) Humidity Panel Support							
11	Room Humidity Input	VAR_UINT16	9					
12	Room Humidify Mode	LIST	10			Off: 0; Humidify: 1; Dehumidify: 2; Auto: 3		
13	Room Humidify State	LIST	11			Off: 0; Humidifying: 1; Dehumidifying: 2		
14	Room Humidify Setpoint	VAR_UINT16	12					
15	Room Dehumidify Setpoint	VAR_UINT16	13					

Figure 5: CSV file with data points

For example, if you change the HVAC Mode using Control4 thermostat UI, the thermostat driver will pass the mode name (as specified in the *Modes Configuration* → *HVAC Modes* property) through the `STRING_VARIABLE` connection to the Modbus driver. If the HVAC mode name matches a key in the `LIST_VALUES` column for the corresponding register, the driver will write the associated value

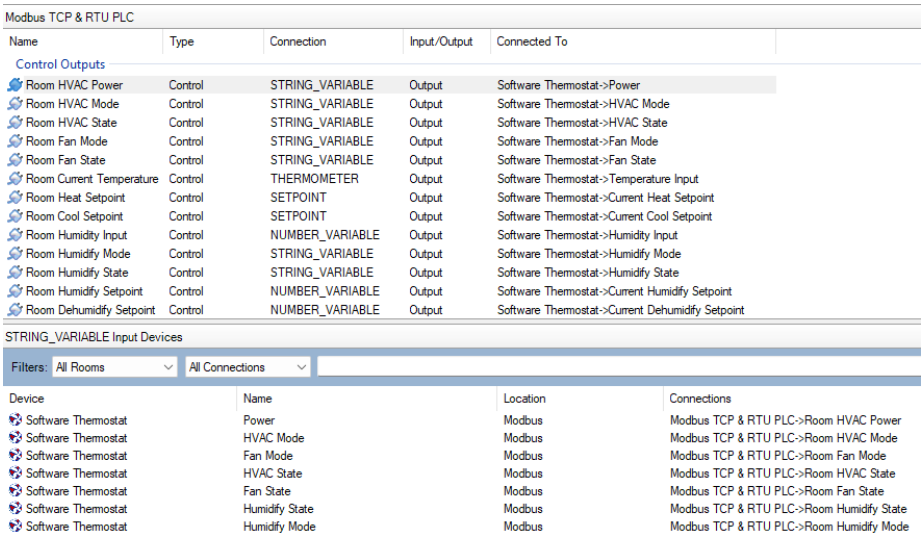


Figure 6: Connections on the Modbus driver

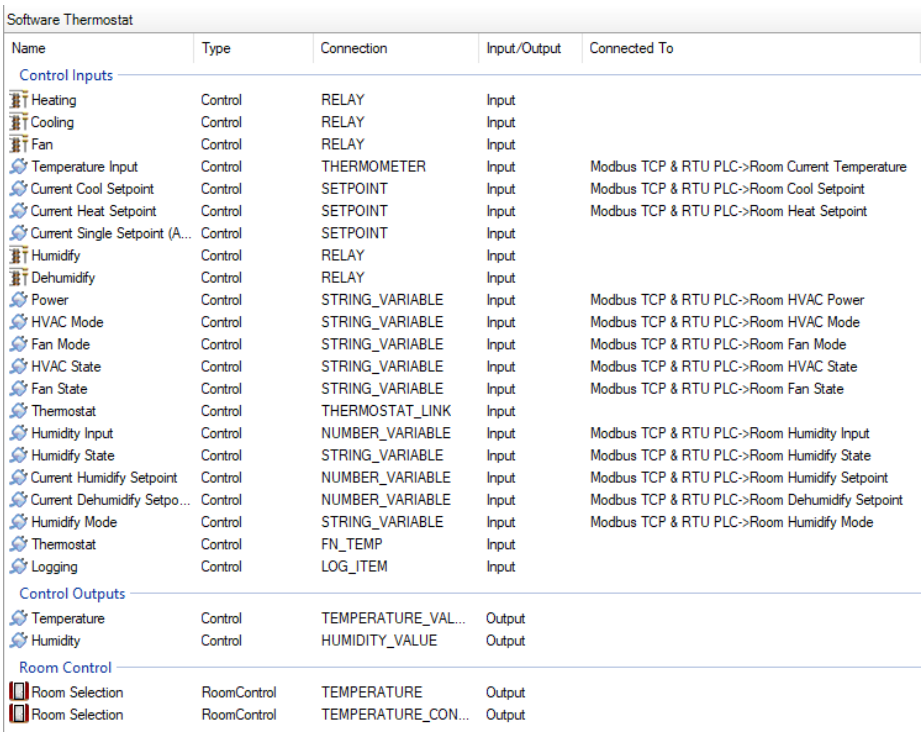


Figure 7: Connections on the thermostat driver

to the appropriate register in the Modbus device. And vice versa, if the Modbus driver reads a new register value from the HVAC Mode register, it will translate it into the text name and send it to the thermostat driver which will then update the user interface accordingly.

Figure 8: List of HVAC Modes specified in the thermostat driver

Programming

All the Events, Actions and Device Variables available on the Modbus driver in the Programming section have been deprecated. Support is not provided if you use these. They are kept for backwards compatibility and may be removed in the future. Only use suitable drivers bound through connections to read and write Modbus data points.

Upgrading from YATUN's yatundev.eu versions

Make sure to read this section very carefully if you are upgrading from any of the older Modbus RTU PLC or Modbus TCP PLC drivers released through yatundev.eu. There is now a single driver *Modbus TCP & RTU PLC* that supports both Modbus TCP and Modbus RTU protocols. If done correctly, you can upgrade the driver in existing installations to the current version without losing your connections and their bindings.

1. Copy automation_ip_modbus_plc.c4z and the other accompanying drivers to your local driver database, i.e., typically Documents\Control4\Drivers\.
2. Backup the project.
3. Add & configure the DriverCentral Cloud driver if not already in the project.
4. Assign the *Modbus TCP & RTU PLC* driver licence to the project on the DriverCentral portal.
5. Open Driver Manager through Driver - Manage Drivers and Agents.
6. Open the context menu on the driver you wish to update/replace and from the *Update Options*:
   - If updating from automation_ip_modbus_plc.c4z: make sure to choose *Sync Local* (the old driver on the controller actually has a higher version number, the updated driver reset the version numbering).
   - If updating from one of the obsolete c4i drivers: choose *Replace Obsolete* under the obsolete Modbus PLC driver. You you might get

an error. Replace the obsolete driver again. This time it should be successful.

7. Restart the director.
8. Verify that the new version of the drivers is now running by the presence of the new Config tab and by checking that the Cloud Status property reads *License Activated* (or *Trial Running*).
9. If you're using Modbus RTU (serial communication): reconnect the serial binding in Connections → Control/AV.
10. Check that all the device driver bindings are didn't get disconnected or mismatched.
11. Check that Modbus communication is working again.

Obsolete c4i drivers that can be replaced

- automation_ip_modbus_plc.c4i
- automation_232_modbus_plc.c4i
- automation_232_modbus_plc_9600_8_none_1_none.c4i
- automation_232_modbus_plc_9600_8_none_1_hardware.c4i
- automation_232_modbus_plc_19200_8_even_1_none.c4i

Let our support know if you are using any other driver filename that you wish to be able to replace with the current version of the driver.

Modbus Resources & Tools

About Modbus

- About Modbus | Simply Modbus Software
- Modbus Tutorial from Control Solutions
- Modbus for Field Technicians (PDF)
- Common Modbus Protocol Misconceptions - Continental Control Systems, LLC
- Modbus - Wikipedia
- Master/slave (technology) - Wikipedia
- RS-485 - Wikipedia
- RS-232 - Wikipedia

Modbus scanners (masters)

- CAS Modbus Scanner (or on Windows Store)
- QModMaster

Modbus simulators (slaves)

- ModRSsim2

Modbus parsers, analyzers

- Rapid SCADA Modbus Parser
- Online Modbus / Analyzer / Simulator / Parser/

Other utilities

- HW VSP3 - Virtual Serial Port