INTELLIGENT
LOAD CONTROL SWITCH

USER MANUAL

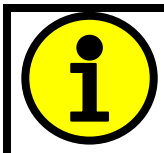
P/N: AX180700

Axiomatic Technologies Oy
Höytämöntie 6
33880 LEMPÄÄLÄ, Finland
Tel. +358 103 375 750
salesfinland@axiomatic.com
www.axiomatic.fi

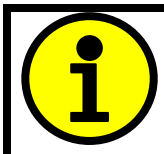
Axiomatic Technologies Corporation
1445 Courtneypark Dr. E.
Mississauga, ON Canada L5T 2E3
Tel. 1 905 602 9270
sales@axiomatic.com
www.axiomatic.com

VERSION HISTORY

Version	Date	Author	Modification
1.0.0	Dec. 13, 2024	Antti Keränen	Initial draft
1.0.1	Apr. 7, 2025	M Ejaz	Marketing review Changed title Added technical specifications



Default configuration password: '**AX180700**'



Default AP SSID: '**Axiomatic_ILCS**'
Default WPA2 key: '**pswd1234**'
Default IP address: **192.168.0.34**

ACCRONYMS

AC	Alternating-current
AP	(WLAN) Access Point
BATT +/-	Battery positive (a.k.a. Vps) or Battery Negative (a.k.a. GND)
DOUT	Digital Output
EA	Axiomatic Electronic Assistant (A Service Tool for Axiomatic ECUs)
ECU	Electronic Control Unit (from SAE J1939 standard)
EEPROM	Electrically Erasable Programmable Read-Only Memory
GND	Ground reference (a.k.a. BATT-)
I/O	Inputs and Outputs
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
MB	Modbus
NTC	Negative Temperature Coefficient
NTP	Network Time Protocol
RTC	Real Time Clock
RTD	Resistance Temperature Detector
SSID	Service Set Identifier
STA	(WLAN) Station
TCP/IP	Transmission Control Protocol / Internet Protocol
Vps	Voltage Power Supply (a.k.a. BATT+)
WLAN	Wireless Local Area Network
WPA2	Wi-Fi Protected Access 2

TABLE OF CONTENTS

List of Figures

List of Tables

REFERENCES

TDAX180700	Technical Datasheet, Intelligent Load Control Switch, Axiomatic Technologies
UMAX07050x	User Manual, Axiomatic Electronic Assistant and USB-CAN, Axiomatic Technologies

NOTE: This product does not have Axiomatic Electronic Assistant support. All configuration and firmware updates are done using the built-in web server.

1. OVERVIEW OF THE CONTROLLER

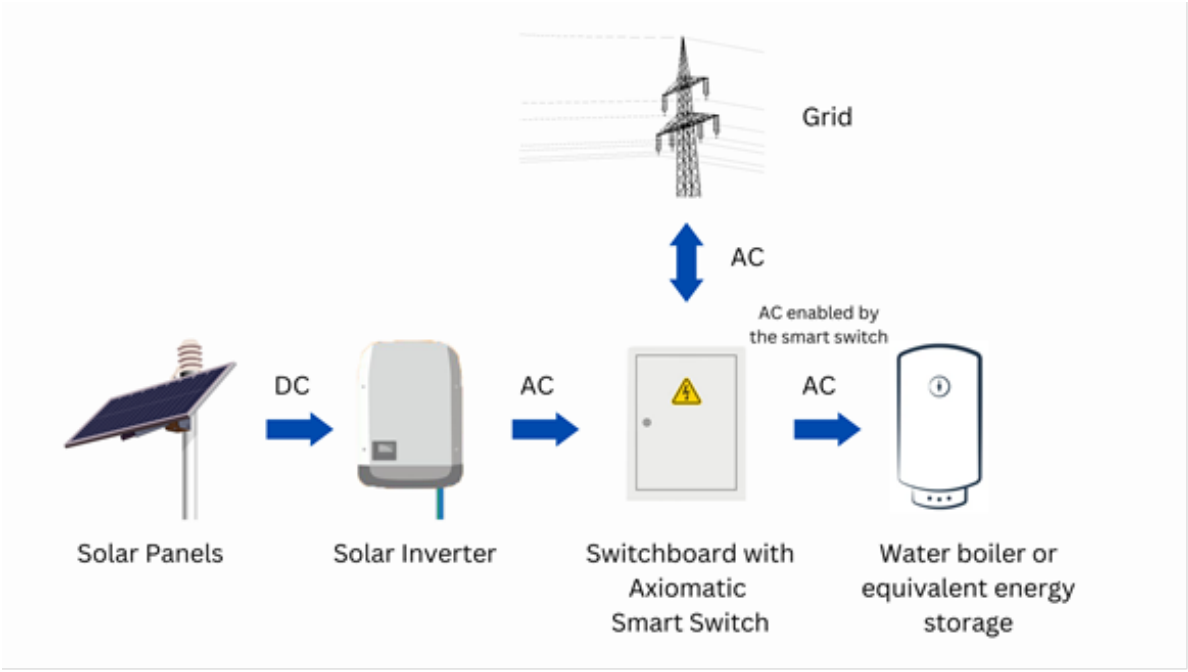


Figure 1 – Application of the Intelligent Load Control Switch

The Intelligent Load Control Switch with Ethernet, WiFi and CAN (later ILCS) is a device that communicates or just listens to ModbusRTU network and actuates a contactor or equivalent electronic component based on the received data and custom configuration. The ILCS configuration can be done using a web browser and the built-in web server running on the ILCS device.

The ILCS supports both WLAN and LAN. The default configuration sets the ILCS into Access Point mode with IP address 192.168.0.34

In addition to the network interfaces, the ILCS has two configurable RTD inputs for temperature measurements, a Contactor output (110VAC/230VAC) and a Digital 24V on/off output.

The ILCS can be used in an application or system where it is advantageous that any extra electrical energy is stored in a local storage. For example, when solar panels generate extra electricity that would be otherwise fed to the grid, it might be more feasible to store the energy as heat in a water boiler or equivalent. The ILCS can make the decision where to route the energy based on the generated energy reading from the inverter with ModbusRTU interface and a configured power threshold.

1.1. Manual switch mode

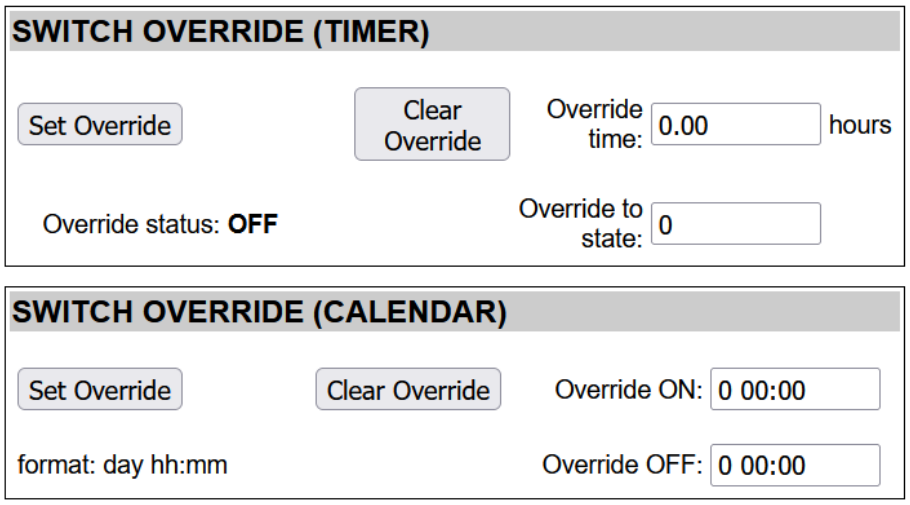


Figure 2 – Switch override

The Contactor output can be manually driven using the **Switch Override (Timer)** settings. **Override time** defines the override time in hours. The override can be either '*on*' or '*off*' depending on the **Override to state** setting.

The **Switch Override (Calendar)** allows to define date and time for the override to be on. This functionality sets the Contactor output on when the override is on, the **Switch override (Timer) Override to state** functionality does not apply with the calendar override.

The **Override ON** and **Override OFF** settings are specified using the format "day hour:minute". For example, setting **Override ON** '*15 19:30*' and **Override OFF** '*16 01:30*' would turn the output ON at 7:30pm on 15th and OFF at 1:30am on 16th.

1.2. ModbusRTU & Solar Inverter settings

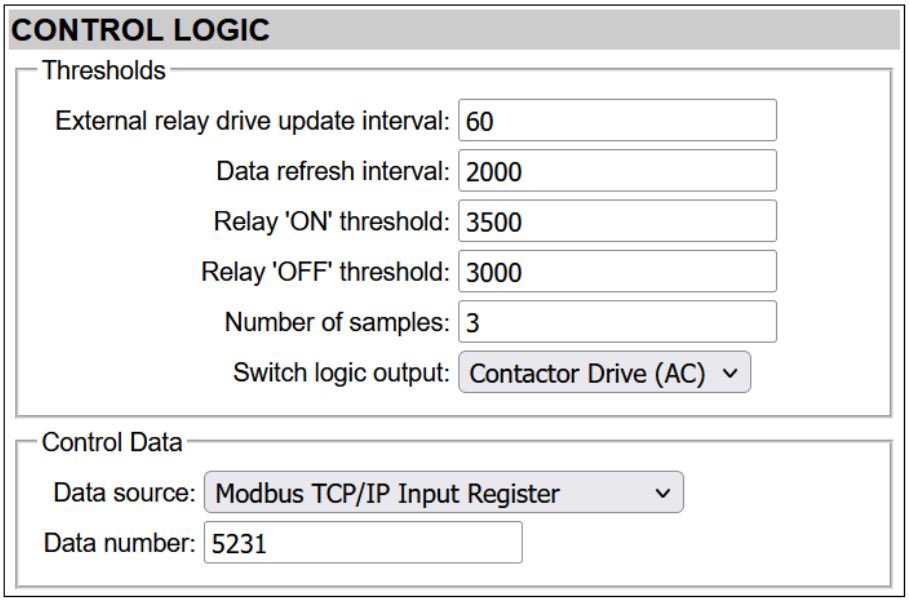


Figure 3 – Control logic

The **External relay drive update interval** defines the switch logic output's update interval in seconds. This setting also defines the ILCS logic update rate.

Data refresh interval is the background data update rate in milliseconds. All internal data in the ILCS logic (Modbus data, RTD readings) are refreshed at this rate.

Relay 'ON' threshold is a (power) value that defines the threshold when the Switch logic output is set 'on'.

Relay 'OFF' threshold is a (power) value that defines the threshold when the Switch logic output is set 'off'. The units for both ON and OFF threshold depend on the data source.

Number of samples defines how many times the control logic needs to be above or below the relay thresholds for the output to change state. For example, on a cloudy day a single below threshold reading might not provide good results. The number of samples is checked at the **External relay drive update interval**s.

Switch logic output allows to choose from Contactor output and Digital output.

Control **Data source** defines where to read the data from. **Data number** defines the Modbus register or input number, depending on the selected control source.

The above settings configure the ILCS to read Modbus data from Holding Register 5231. Although the ILCS reads ModbusRTU messages in, the Control Logic configuration specifies *ModbusTCP/IP Input Register* instead. This is because the ILCS's Modbus master creates a local copy of the remote node's registers when configured to operate in *Listen-only mode* (see section 1.3.1).

When the configured **Data source** value (Modbus Input Register 5231) is above 3500 three times in a row (180 seconds), the ILCS output is turned on. When the value is below 3000 for the same time, the ILCS output is turned off.

1.3. Modbus master

The ILCS supports 16 Modbus master messages. These messages support reading data from a remote Modbus device as well as writing data to a remote Modbus device.

The configuration options include remote **Device address**, Modbus **Function**, **Register address** and **Number of registers** to address in a single read/write operation.

Both Ethernet and RS485 interfaces support Modbus messaging. The Modbus master support on Ethernet/RS485 needs to be specified using the **Ethernet master enabled** and **RS485 master enabled** options. Please note, that if Modbus slave feature is needed, the Modbus master cannot be active for that interface.

1.3.1. Modbus master in listen mode

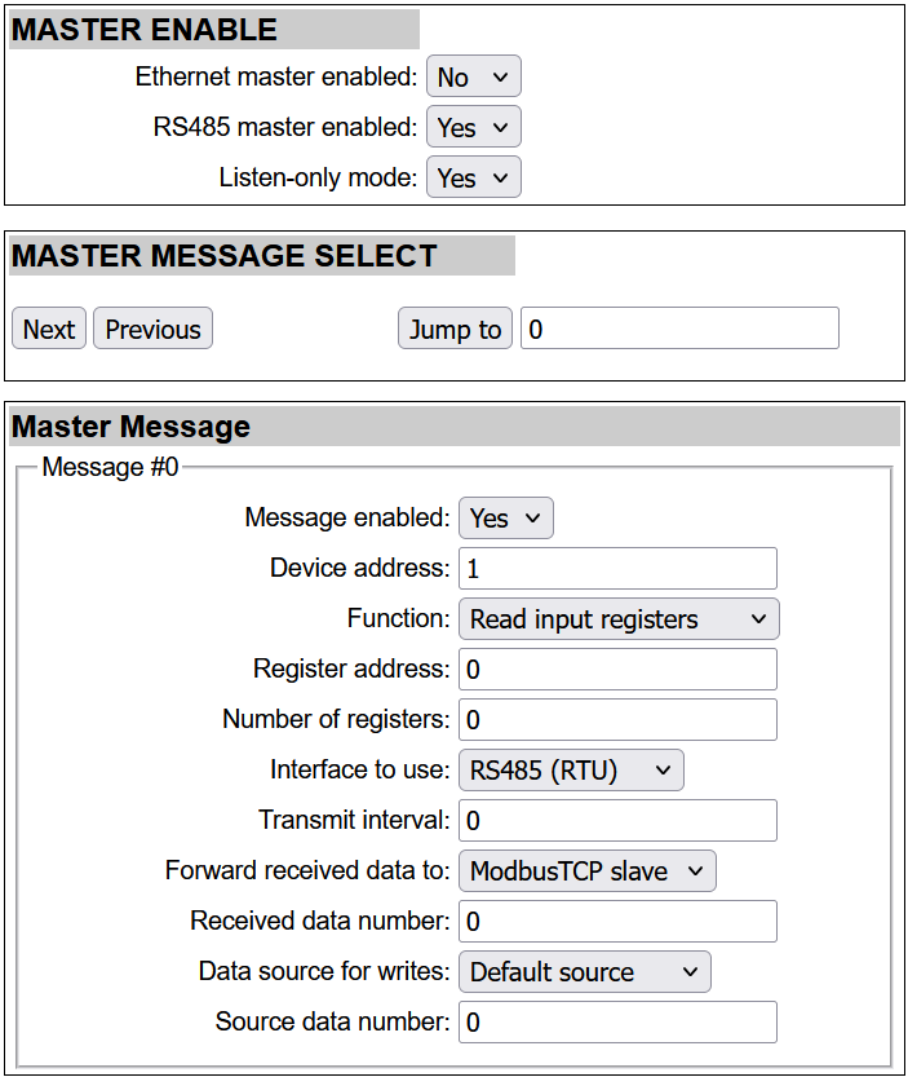


Figure 4 – Modbus master

The **Listen-only mode** is a special feature of the ILCS. When this mode is enabled, the ILCS works as a Modbus master but does not send any messages out. Messages already on the Modbus network are read in as replies to Modbus requests generated by the ILCS. This mode allows listening Modbus traffic generated by other nodes, thus enabling the reading of the operating status of a 3rd party solar inverter or equivalent power generating device that is already communicating with another Modbus master (such as a WLAN-Modbus interface providing remote management features).

The **Listen-only mode** configuration consists of these main points: The **Device address** needs to match the one used by the remote (solar inverter) Modbus device. The **Function** is the function that the other Modbus master uses for reading the solar inverter status.

Forward received data to specifies that the received data is forwarded to the local Modbus slave implementation. This effectively creates a local copy of the remote device's (solar inverter) Modbus registers on the ILCS. When **Forward received data to** is set to *ModbusTCP slave*, the **Received data number** defines the local Modbus register to start from.

1.4. RTD logic

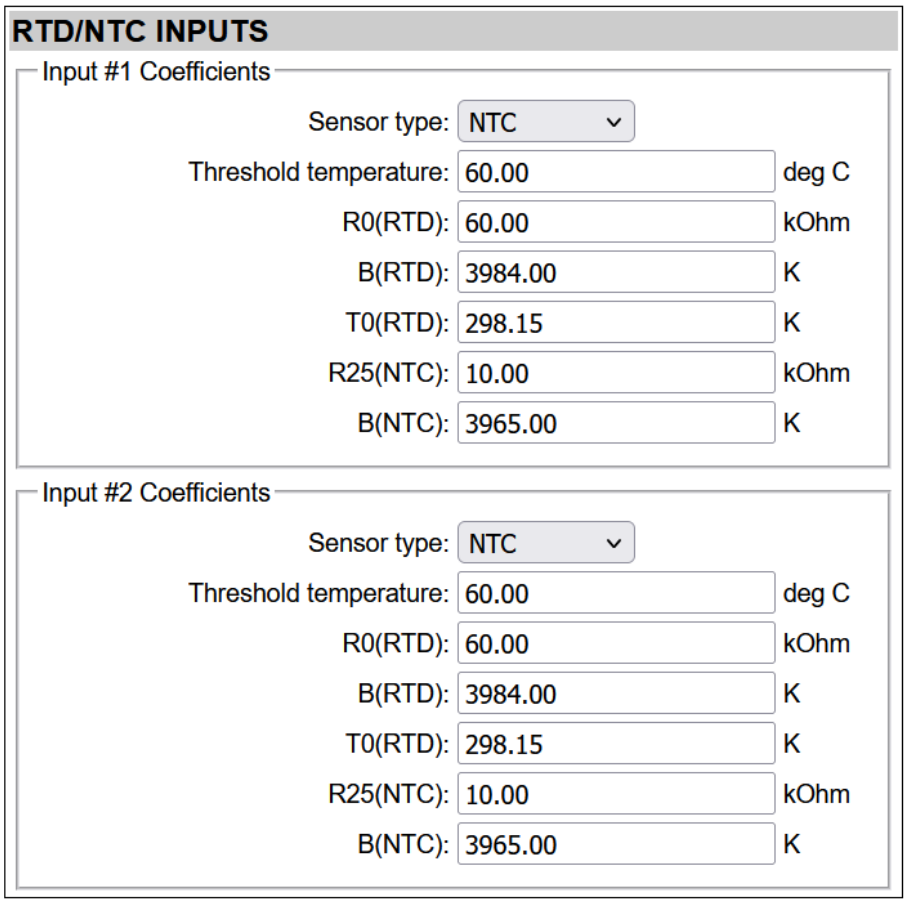


Figure 5 – RTD/NTC settings

In addition to Modbus features, the ILCS has two RTD inputs. These can be used for example for monitoring the temperature of a water boiler. The output logic supports using the RTD temperature measurements for driving the contactor and digital output.

The **Threshold temperature** setting specifies the temperature threshold for the output control logic. When both RTD measurements are above the configured threshold, the output is turned *OFF*. When both measurements are below threshold, the output is turned *ON*. In the case of a water boiler with upper and lower temperature measurements, this provides the hysteresis function needed for controlling a heating element in the boiler as a function of temperature.

The other configuration options are the RTD and NTC coefficients, used in these equations for the temperature sensor in question.

RTD calculations, in °C:

$$T = \frac{B(RTD)}{ln\left(\frac{Rthermistor}{R0(RTD) * e^{\frac{-B(RTD)}{T0(RTD)}}}\right)} - 273.15°C$$

NTC calculations (in which *T25=298.15K*), in °C:

$$T = \frac{1}{\frac{ln\left(\frac{Rthermistor}{R25(NTC)}\right)}{B(NTC)} + \frac{1}{T25}} - 273.15°C$$

1.5. Data logging

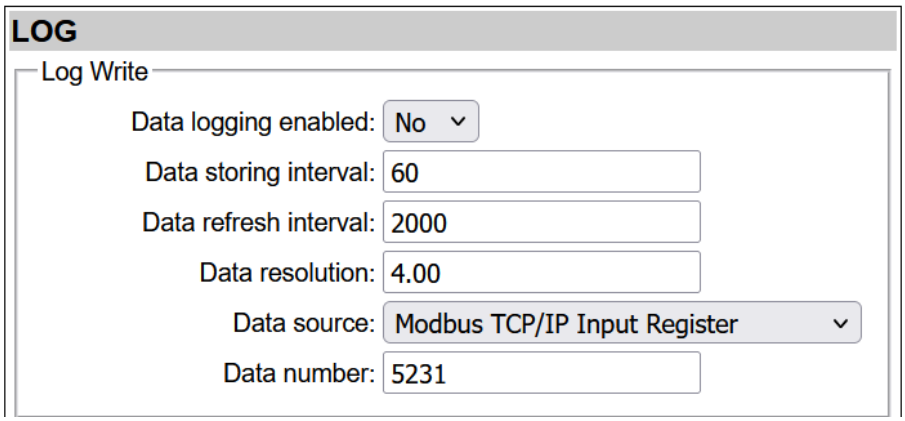


Figure 6 – Data logging settings

The ILCS supports logging of a single variable into non-volatile memory (EEPROM). The data log storage is limited to max 2880 bytes, which can hold up to 24 hours of logging of 16-bit data at 60 second intervals.

16-bit data log entry structure:

```
 MSB                                                              LSB
 15          13 |   12   | 11                                      0
 ---------------------------------------------------------------------
     wrCnt      | State |                  Data
```

The 16-bit log entry consists of 12 bits of logged data, status bit reporting the Switch Logic state and a 3-bit write count.

The **Data refresh interval** is defined in milliseconds and should be smaller than the **Data storing interval**. The refresh interval is used in the data logging task for refreshing the internal status and updating a built-in average filter for the logged data. The data log average filter has 16 entries.

Since the log entry can hold a 12-bit value (max value = 4095), the **Data resolution** can be used for scaling the data. For example, a resolution of *4.00* allows max 16380W solar power to be logged.

The data logging restarts on every boot up, from address 0. The first write after boot up is delayed 60 seconds. The write interval is updated to **Data storing interval** after the first write.

Binary data log file contents (byte numbers are zero-based):

```
5-byte file id tag (fwID in B2 & B3)
30-byte header tag (file type in B26-B33)
6-byte EEPROM tag
8-byte log start date & time tag (day=B43,month=B42,year=B41,hour=B44,minute=B45)
---------------------------------------------------------------------
2-byte data log entry #0 (see the description above)
2-byte data log entry #1
2-byte data log entry #2
…
```

1.6. Time and date

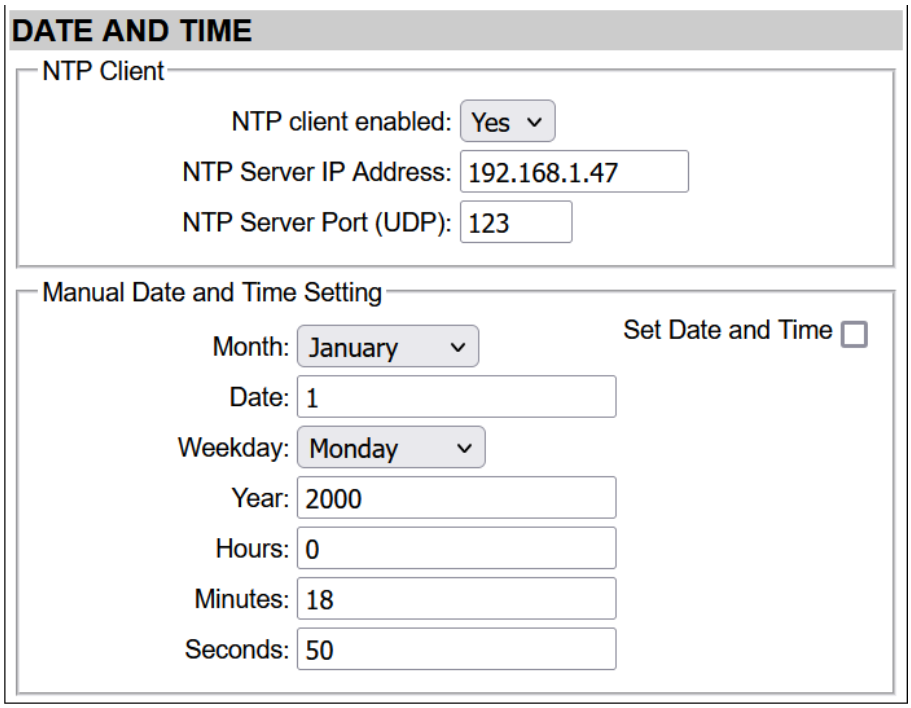


Figure 7 – Date and time settings

The date and time can be read from an external NTP server. Alternatively, the date and time can be set manually. For setting the date and time manually, the **Set Date and Time** option needs to be selected before saving settings from the top of the configuration page.

In case the NTP server is used, the ILCS sends an NTP query once at boot up. The date and time are automatically set on reception of the reply from the NTP server.

The date and time information are used in the data logging and manual calendar-based output control feature (Switch override (Calendar), see section 1.1). The date and time settings don't affect the output control logic in any other way.

2. INSTALLATION INSTRUCTIONS

2.1. Dimensions and Pinout

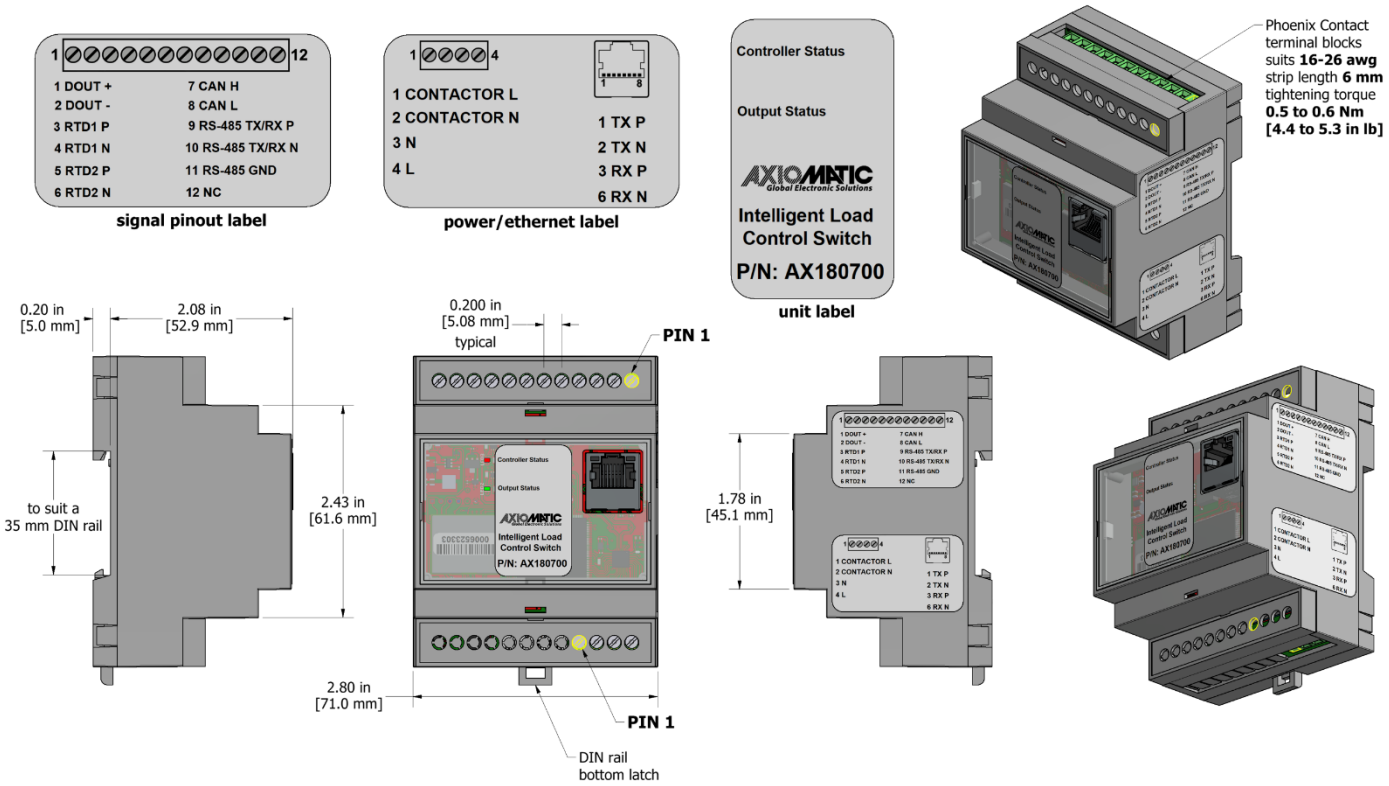


Figure 8 – Controller Dimensions and Label

Top connector pinout

Pin #	Electrical Connection
1	DOUT+
2	DOUT-
3	RTD1_P
4	RTD1_N
5	RTD2_P
6	RTD2_N
7	CAN_H
8	CAN_L
9	RS485_TX/RX_P
10	RS485_TX/RX_N
11	RS485_GND
12	Not Used

Bottom connector pinout

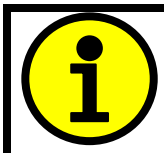
Pin #	Electrical Connection
1	CONTACTOR_L
2	CONTACTOR_N
3	N
4	L

Table 1 – AX180700 Connector Pinout

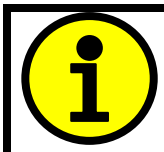
3. WEB BROWSER BASED CONTROLLER CONFIGURATION

The ILCS supports configuration from Ethernet port using a standard web browser.

The web browser-based configuration requires a password before any of the parameters can be viewed or edited.



Default configuration password: '**AX180700**'



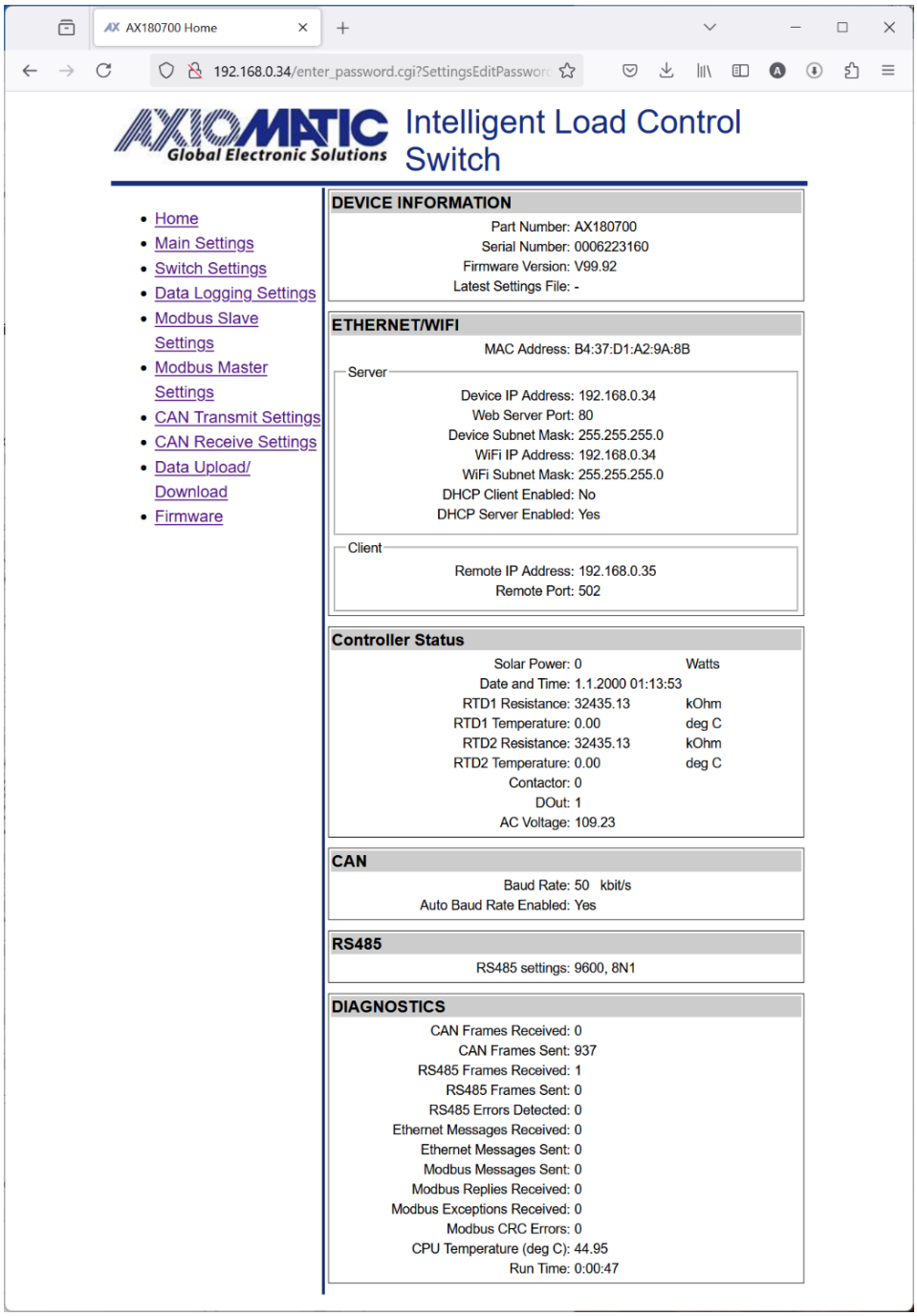
Default AP SSID: '**Axiomatic_ILCS**'
Default WPA2 key: '**pswd1234**'
Default IP address: **192.168.0.34**

3.1. Parameter Editing

The ILCS has a web server running on TCP port 80. Please note that the “Save settings” button must be pressed to save the parameters. If not, the modified parameters are discarded when moving to another configuration page or closing the web browser. The ILCS web server supports the configuration options described on the following pages.

<configured ip>

<configured ip>/index.shtml



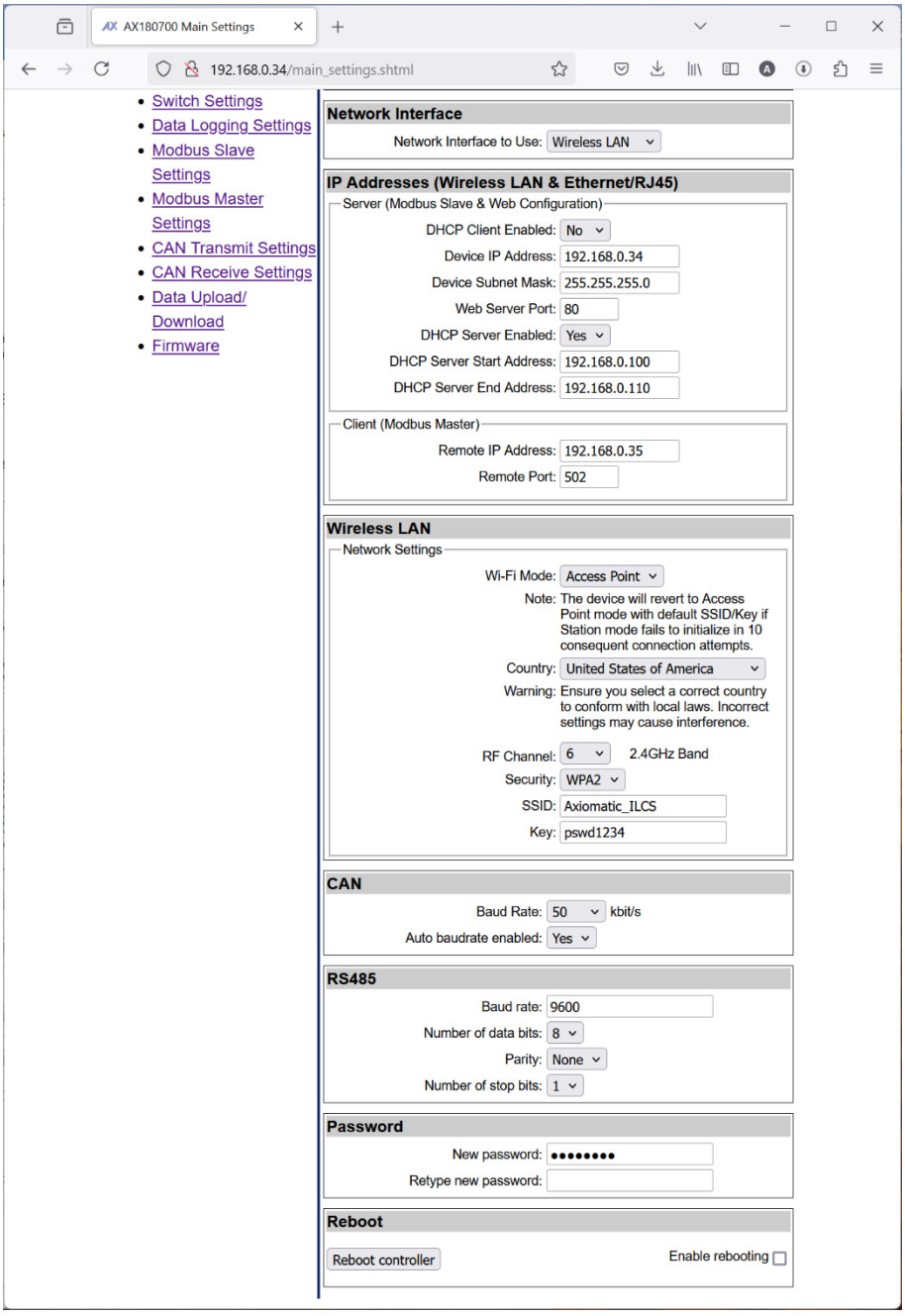
The Home page (index.shtml) shows the main configuration parameters, current controller status and communication statistics for the different interfaces. This page has no editable parameters.

<configured ip>/main_settings.shtml

The Main Settings configuration page allows the user to modify the main configuration parameters for the communication interfaces, such as device's IP address, netmask, DHCP client/server settings and the WLAN settings.

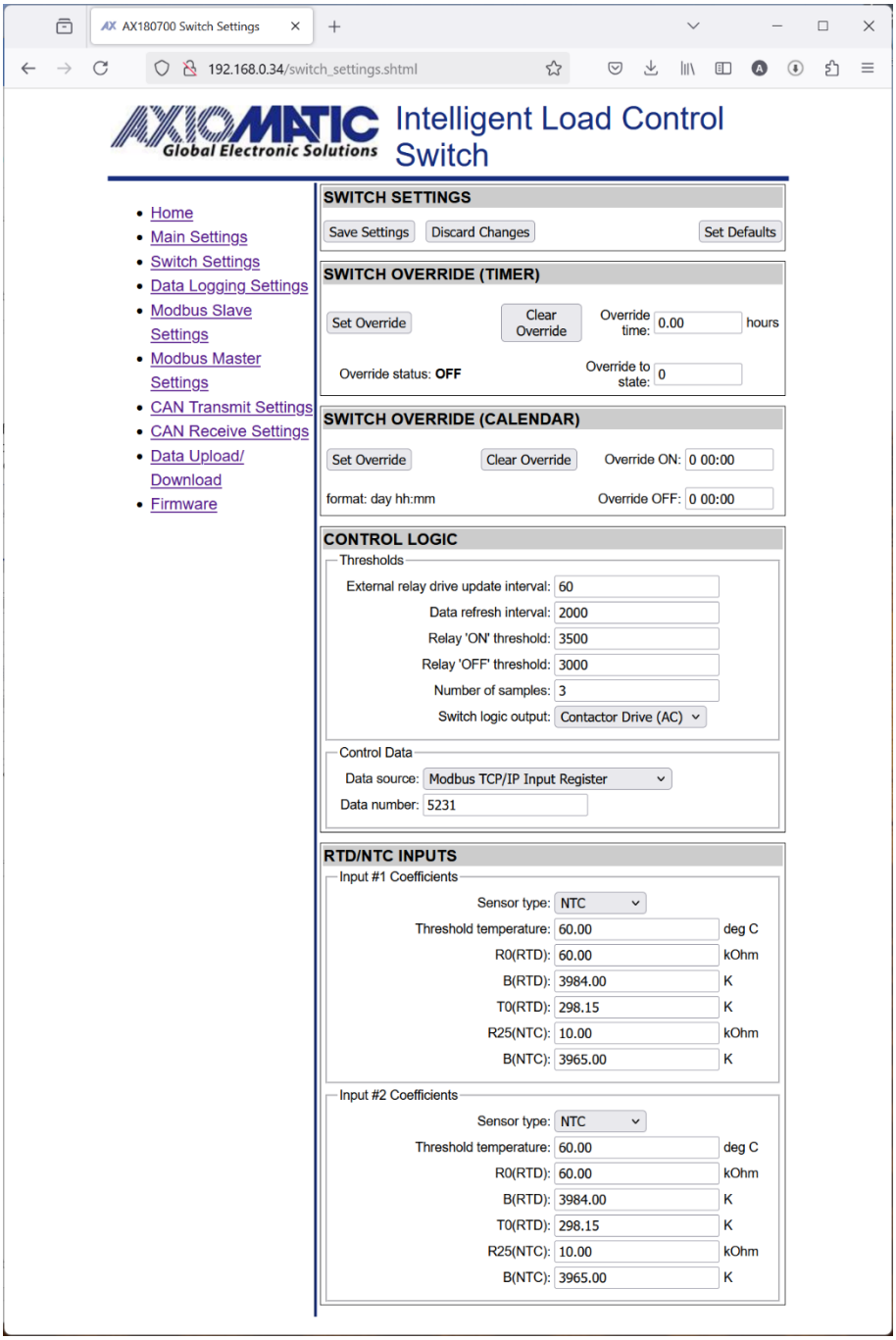
The device can operate as a WLAN Access Point or Station. By default, the AP mode is used for easier initial access.

The password can be changed by entering the same password to both **New password** and **Retype new password** entries. In case the **Retype new password** is left empty, or the two passwords don't match, the password is not modified.



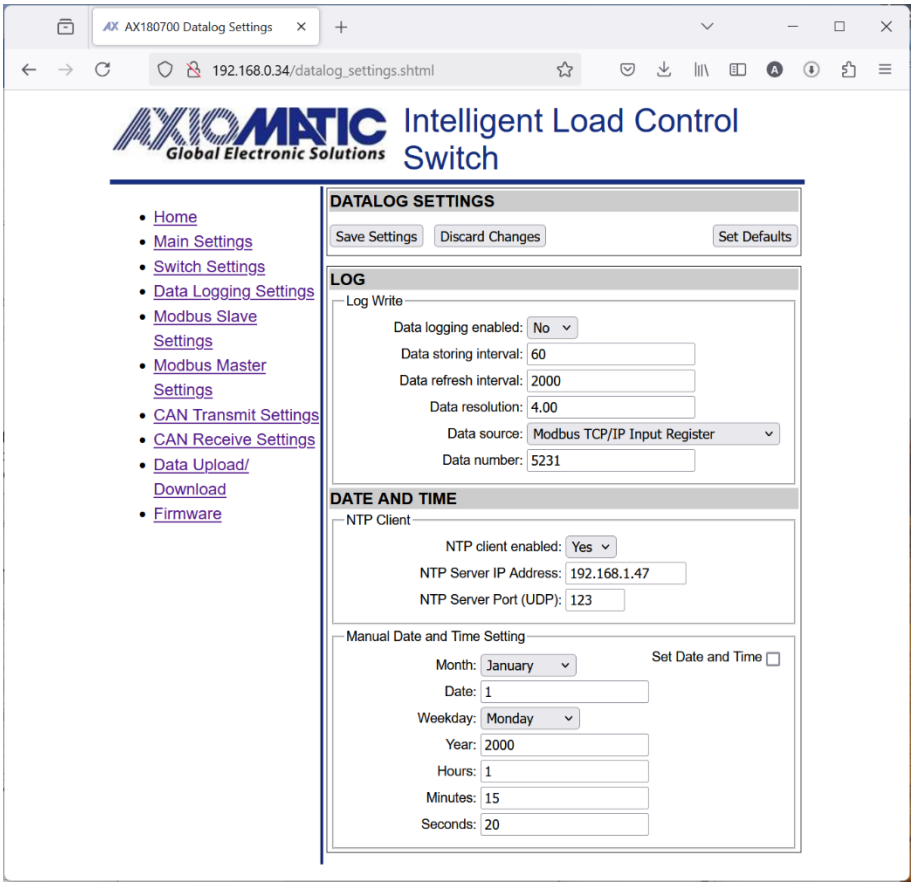
<configured ip>/switch_settings.shtml

The switch settings page contains the main ILCS control logic configuration options.



<configured ip>/datalog_settings.shtml

The ILCS has an EEPROM chip for local data storage. A part of this EEPROM is reserved for data logging purposes. There is also a built-in RTC for local timekeeping. The time and date information can be manually entered or an external NTP server can be utilized.



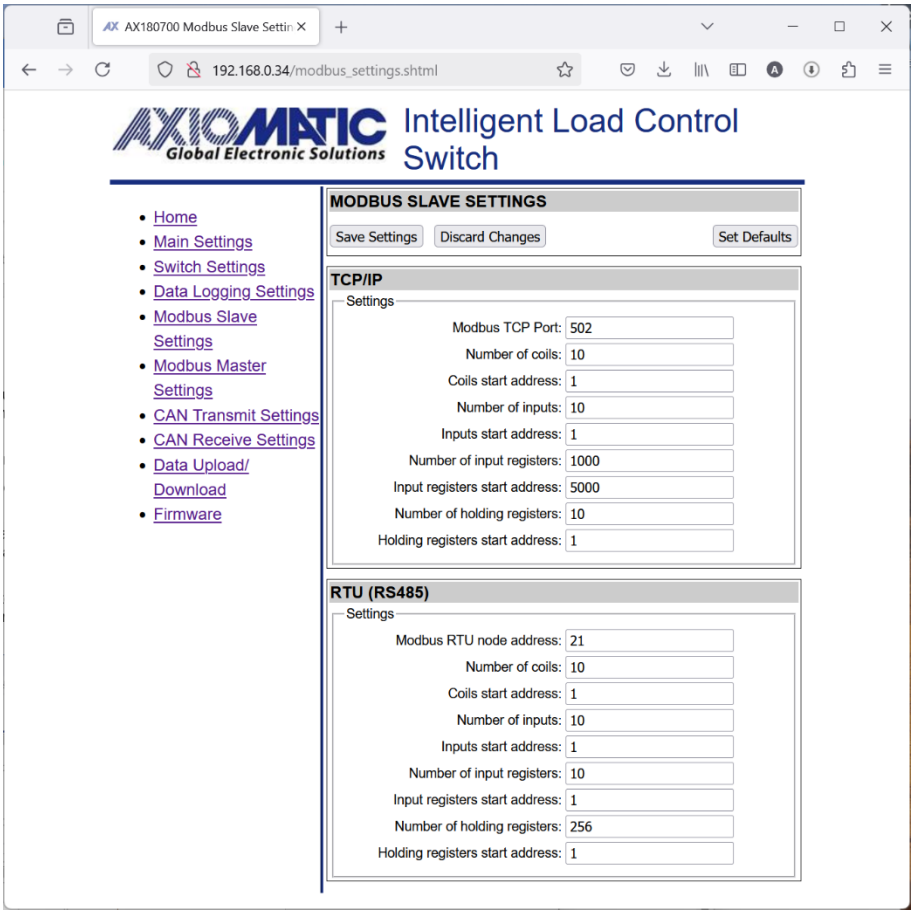
<configured ip>/modbus_settings.shtml

The ILCS supports Modbus RTU and Modbus TCP/IP slaves running on the RS485 and Ethernet ports, respectively. The number and start address for each of the Modbus slave's inputs, coils and registers can be specified.

Please note that the slave interface is enabled only if the Modbus master implementation is not running on that interface.

The Modbus TCP/IP node address is a "don't care", but the RTU slave interface will read in only the messages that are targeted to the configured Modbus node address.

The number of inputs, coils and registers have an upper limit, and the web server will not accept values beyond the built-in maximum limit. If the upper limit is exceeded, the value is saturated to the maximum allowed value.



<configured ip>/modbus_master.shtml

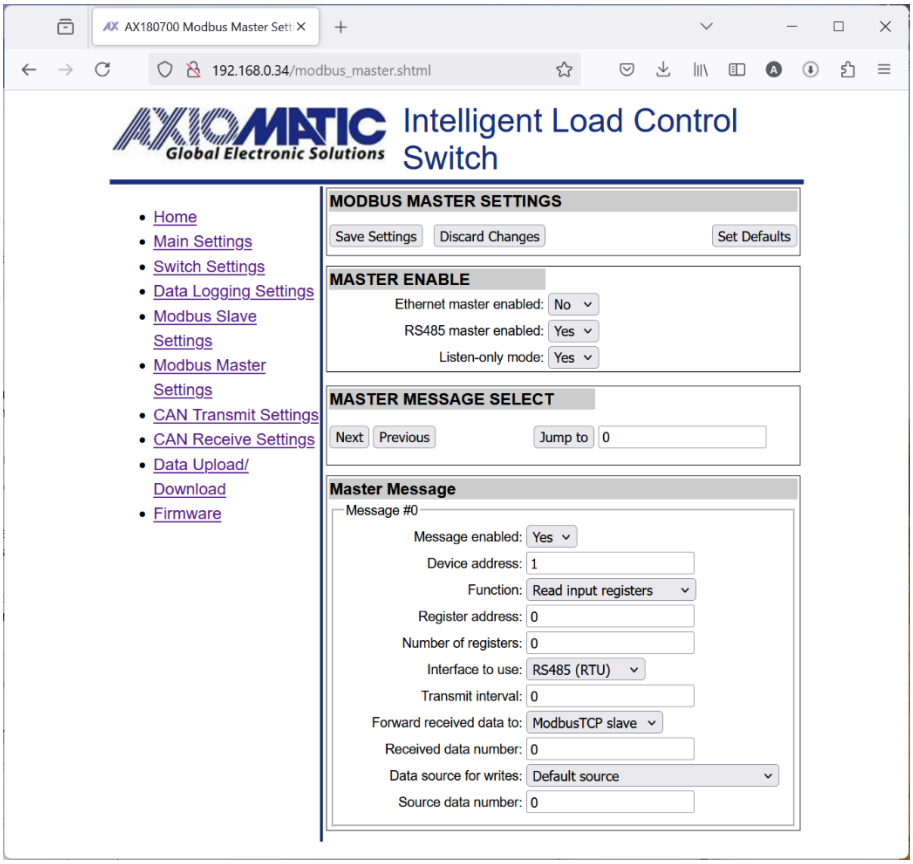
The ILCS supports Modbus RTU and Modbus TCP/IP master running on the RS485 and Ethernet ports, respectively. The **Ethernet master enabled** and **RS485 master enabled** options need to be set to 'Yes' for enabling the corresponding Modbus master.

Please note that enabling the Modbus master will disable the corresponding Modbus slave.

The **Forward received data to** and **Default source for writes** data routing default options are the built-in variables for each Modbus master message definition.

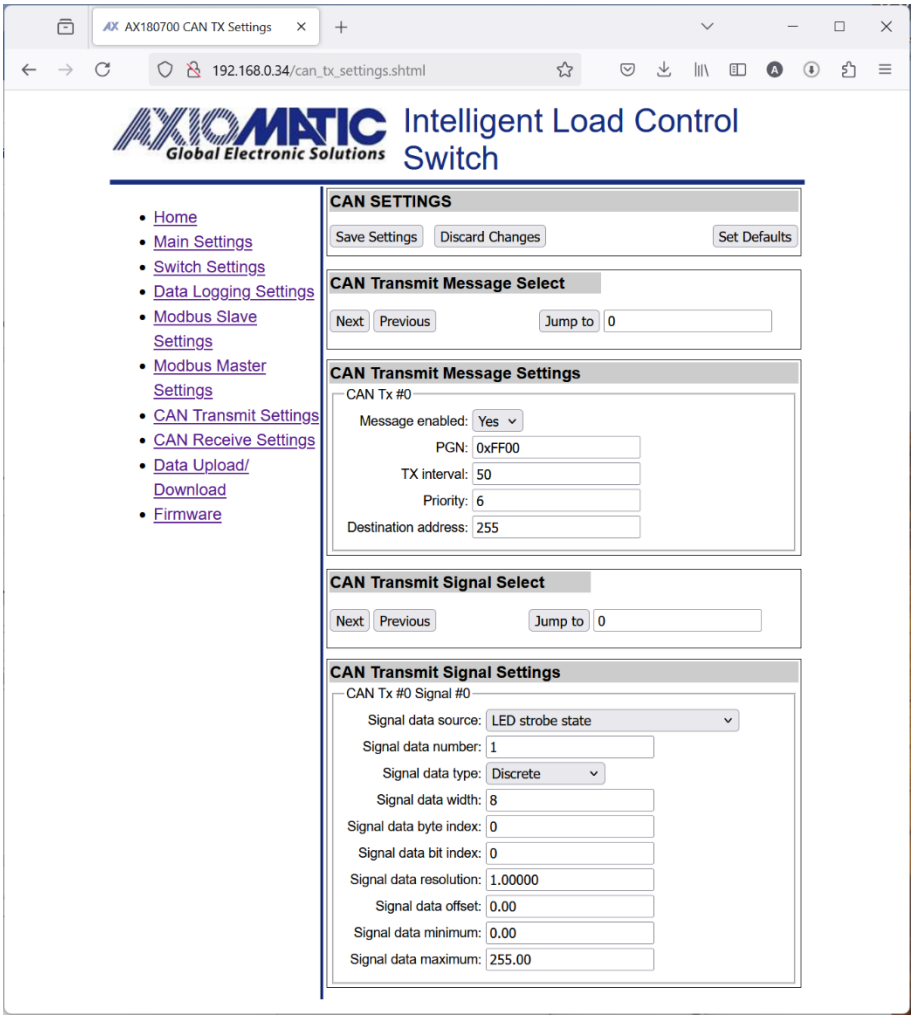
In addition to the normal Modbus master functionality, the ILCS also supports **Listen-only mode**. In this mode, the ILCS reads in Modbus replies sent by other nodes as if those replies would be targeted to the ILCS.

The **Listen-only mode** is useful in cases when the ILCS needs to listen to Modbus communications from a Solar Inverter, requested by another master on the bus, for example a 3rd party WLAN adapter attached to the Solar Inverter for remote monitoring purposes.



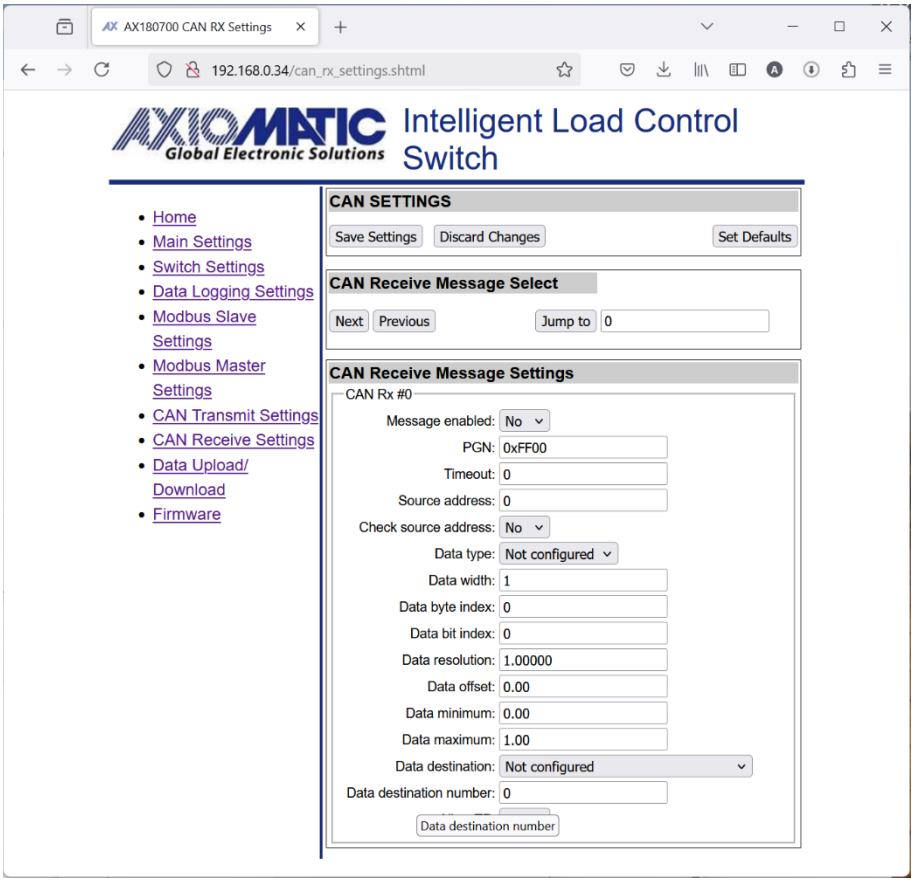
<configured ip>/can_tx_settings.shtml

The ILCS supports two ‘J1939 like’ CAN transmit messages with 29-bit Extended IDs, both containing up to 6 signals (variables). All parameters can be configured, including PGN, transmit interval, data source and data scaling and positioning in the CAN frame data bytes.



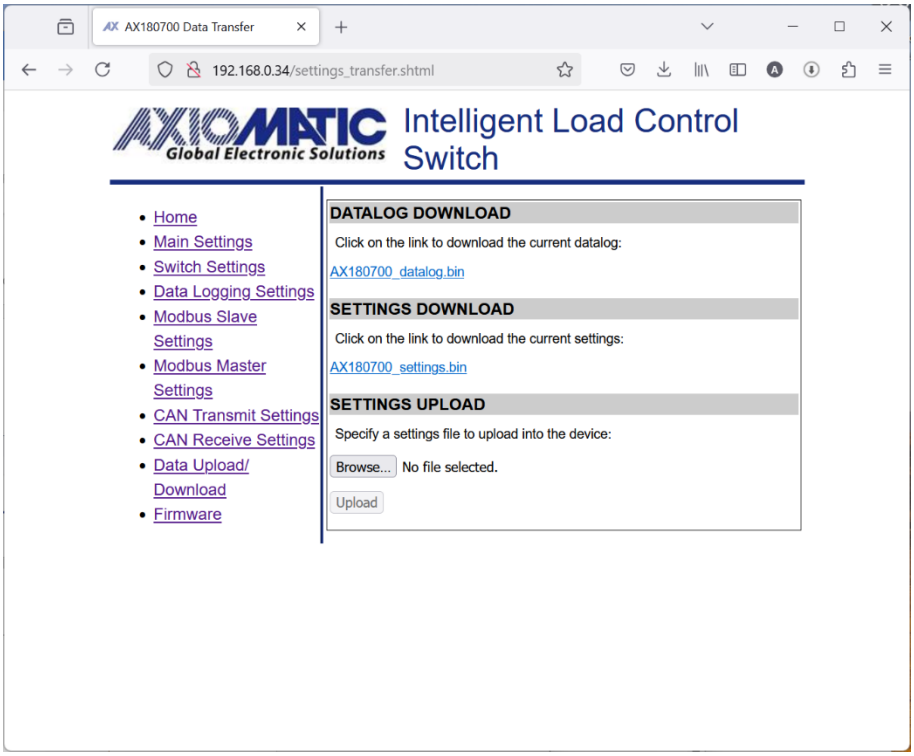
<configured ip>/can_rx_settings.shtml

Four CAN receive messages are supported. The received data can be forwarded directly to the Modbus slave registers if needed or used in the Switch control logic. The CAN receive message is ‘J1939 like’, 29-bit Extended IDs are expected.



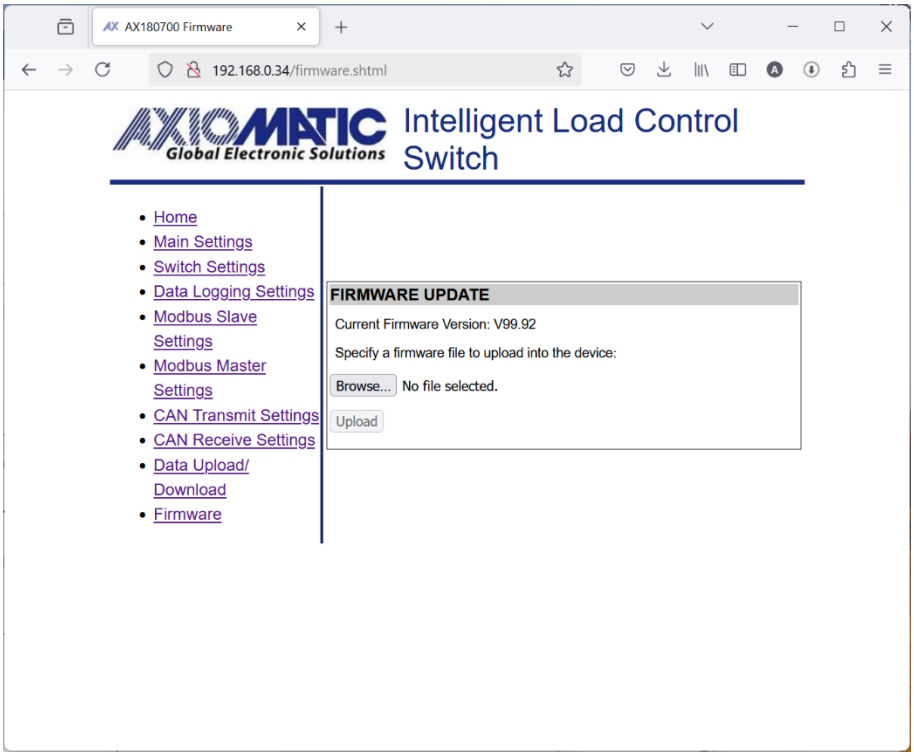
<configured ip>/settings_transfer.shtml

The ILCS settings and Data log contents can be downloaded to a binary file using the functionality available on the Data Upload/Download page.



<configured ip>/firmware.shtml

Please see section 4 for details of firmware upgrade procedure.

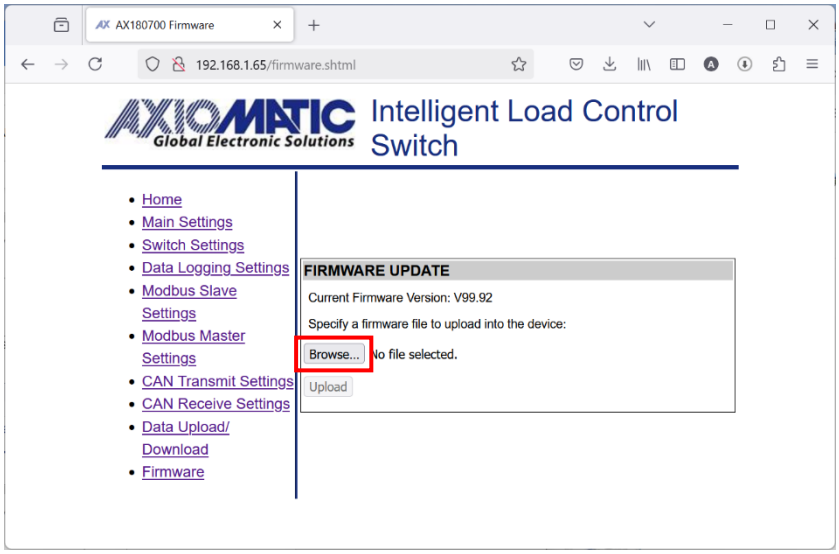


4. REFLASHING OVER ETHERNET USING A WEB BROWSER

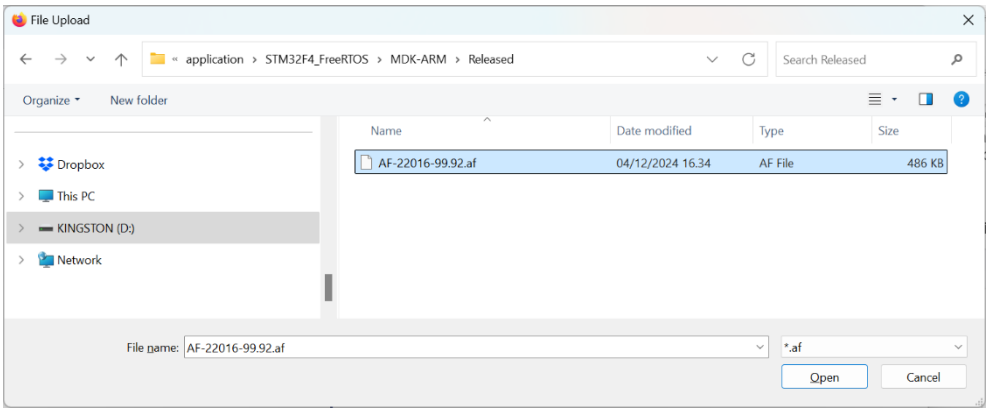
The AX180700 can be upgraded with new application firmware using a web browser. Once the correct configuration password is entered, the firmware reflash can be done using the 'Firmware' page.

Note that the firmware can be only upgraded using the Ethernet interface available on the RJ45 connector.

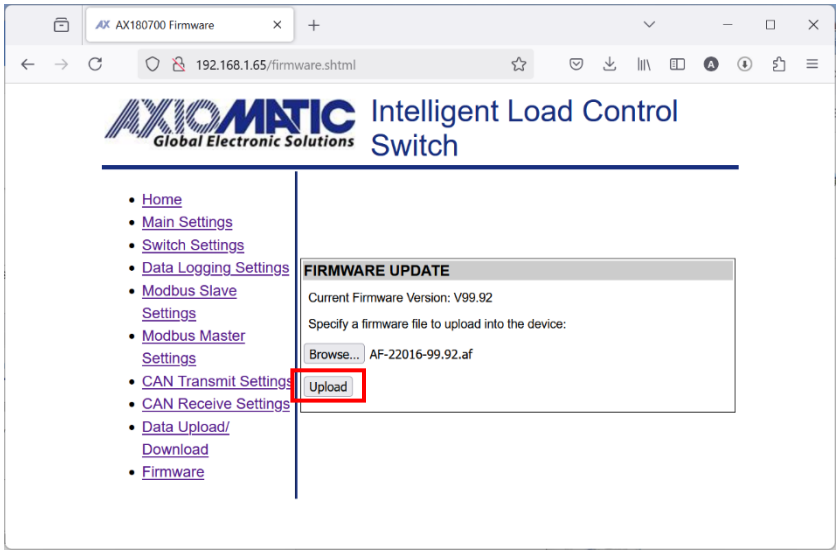
<configured ip>/firmware.shtml



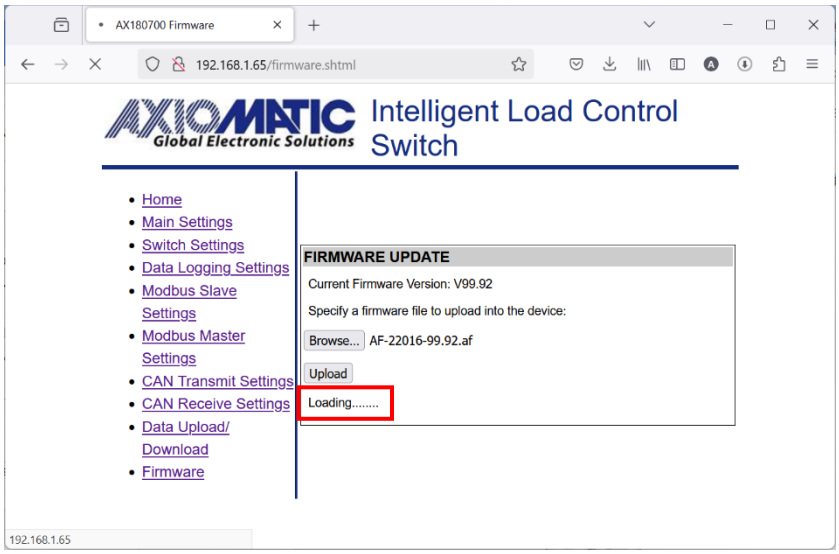
On the 'Firmware' page, a file selection dialog can be opened by pressing the 'Browse…' button.



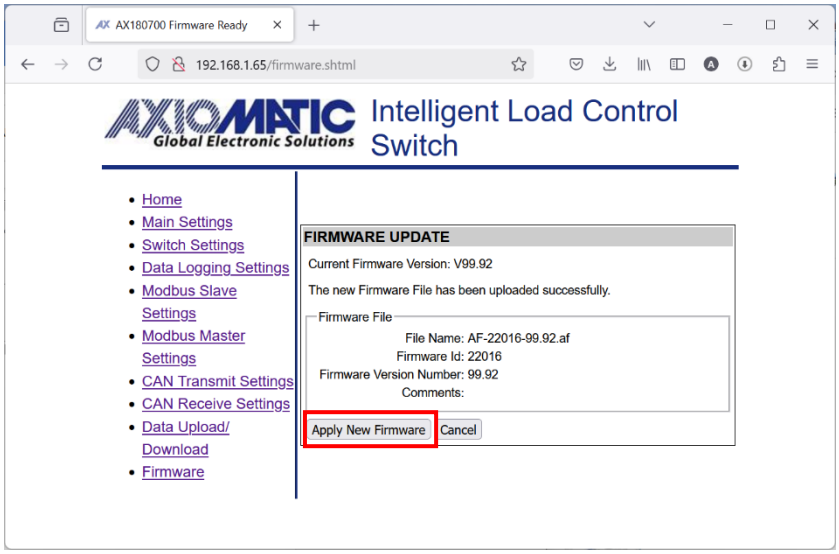
Navigate to where you had saved the **AF-22016-x.xx.af** file sent from Axiomatic. (Note: only binary (.af) files can be flashed using the web browser firmware update interface.)



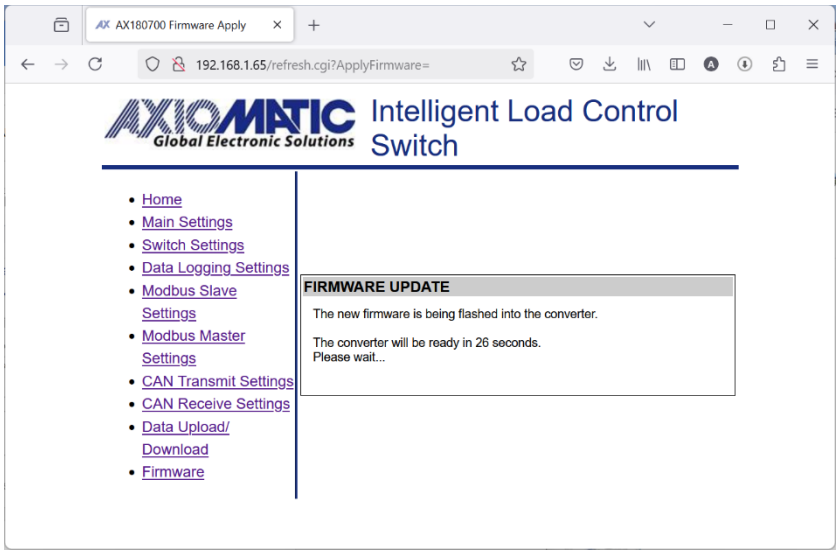
Once the file is selected, the actual upload/upgrade process is started by pressing the ‘Upload’ button.



The firmware upload process is shown below the ‘Upload’ button.



Once the upload is finished and the file checked and stored to a temporary location on the AX180700, the user is prompted to either to ‘Apply New Firmware’ or cancel the operation.



The firmware reflash procedure takes 30 seconds to finish. After this the AX180700 reboots automatically to the new firmware and returns to the password dialog.

APPENDIX A - TECHNICAL SPECIFICATION

Specifications are indicative and subject to change. Actual performance will vary depending on the application and operating conditions. Users should satisfy themselves that the product is suitable for use in the intended application.
All our products carry a limited warranty against defects in material and workmanship. Please refer to our Warranty, Application Limitations & Return Materials Process as described on https://www.axiomatic.com/service/.

All specifications are typical at nominal input voltage and 25 degrees C unless otherwise specified.

Specifications

Power Supply	100 to 240 Vac
Inputs	2 RTD inputs for PT100 temperature measurements
Outputs	1 ON/OFF solenoid output (100/240Vac, maximum 100mA) 1 ON/OFF solenoid output (24V, maximum 1A) The output drives a 3-phase contactor.
Status LEDs	Output state Controller health/mode
Enclosure and Dimensions	Bud Industries DIN Rail Mount Multi-Board Box (P/N : **DMB-4772**) ABS/PC blend plastic housing Flammability Rating: UL94V-0 Refer to Figure 1.
Communication	Modbus RTU (RS-485) Wi-Fi Ethernet CANopen®
Weight	0.204 kg. (0.45 lb.)
Temperature Rating	Operating: -40 to 85°C (-40 to 185°F) Storage: -50 to 90°C (-58 to 194°F)
Humidity	0-99% relative humidity (non-condensing)
Protection Rating	IP20
Electrical Pinout	Power: Phoenix Connector P/N: **1729144** Input/Output/CAN/RS-485: Phoenix Connector P/N: **1729225**
Compliance	RoHS Directive
User Interface	Webserver
Installation	AC power – 10A fuse **Follow local electrical codes.**
Mounting	DIN rail mounting

APPENDIX B - DATA LOG CONVERSION UTILITY EXAMPLE

```
/*****************************************************************************
 *
 * Project name: Convert AX180700 Data log file to csv
 *
 * Created     : Fri Dec 13 18:27:12 2024
 *
 * Copyright 2005- Axiomatic Technologies Oy
 * All Rights Reserved.
 *
 * $Source$
 *
 *****************************************************************************/
/*
 * Build using MinGW in Windows: gcc -Wall -o convert_datalog.exe convert_datalog.c
 */
/*****************************************************************************
 * System includes
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <Windows.h>

/*****************************************************************************
 * Application includes
 *****************************************************************************/

/*****************************************************************************
 * Constant definitions
 *****************************************************************************/

/*****************************************************************************
 * Type definitions
 *****************************************************************************/

/*****************************************************************************
 * Global variables
 *****************************************************************************/

/*****************************************************************************
 * Static variables
 *****************************************************************************/

/*****************************************************************************
 * Function Prototypes
 *****************************************************************************/

/*****************************************************************************
 * Functions
 *****************************************************************************/
#define EXIT_AFTER_ERROR              \
        Sleep(2000);           \
        return EXIT_FAILURE

/*****************************************************************************//**
 *<b> Function name </b>: main \n
 *<b>    returns    </b>: int \n
 *<b>    arg 1      </b>: int argc \n
 *<b>    arg 2      </b>: char **argv \n
 *<b> Created by    </b>: antti \n
 *<b> Date created  </b>: Wed Oct 25 10:35:36 2017 \n
 *<b> Description   </b>: main function \n
 *<b> Notes         </b>: \n
 *****************************************************************************/
```

```
int main(int argc, char *argv[]) {

	FILE *file_in, *file_out;
	char fileName_input[128], fileName_output[128], tmpStr[128], *pFileType;
	unsigned char line[64];
	size_t len = 0;
	ssize_t read;
	unsigned int logEntryData, fwID, wrRound, outState;
	int firstLines, entriesConverted;


	if( argc == 3 ) {
		memset((void *)fileName_input, 0x00, sizeof(fileName_input));
		strncpy(fileName_input, argv[1], sizeof(fileName_input));
		memset((void *)fileName_output, 0x00, sizeof(fileName_output));
		strncpy(fileName_output, argv[2], sizeof(fileName_output));
	}
	else {
		printf("Usage: %s AX180700_datalog.bin AX180700_datalog.csv\n", argv[0]);
		EXIT_AFTER_ERROR;
	}

	/*
	 * Reading in the lines from the specified file
	 */
	printf("Converting \"%s\" to \"%s\"...\n", fileName_input, fileName_output);

	if( (file_in = fopen(fileName_input, "rb")) == NULL ) {
		printf("ERROR opening \"%s\" for reading\n", fileName_input);
		EXIT_AFTER_ERROR;
	}
	if( (file_out = fopen(fileName_output, "w")) == NULL ) {
		printf("ERROR opening \"%s\" for writing\n", fileName_output);
		EXIT_AFTER_ERROR;
	}

	// Read lines from original file and write them into the new one
	/*
	-------------------------------------------------------------------------------------------------------
---
	Datalog (binary) file contents:
	-------------------------------------------------------------------------------------------------------
---
	(5 byte file id tag)
	(30 byte header tag)
	(6 byte EEPROM size tag)
	(8 byte date & time tag)
	(2 byte data log entry #1: 4 byte metadata, 12 byte data)
	(2 byte data log entry #2: 4 byte metadata, 12 byte data)
	(2 byte data log entry #3: 4 byte metadata, 12 byte data)
	...
	-------------------------------------------------------------------------------------------------------
---
	CSV file contents:
	-------------------------------------------------------------------------------------------------------
---
	line; write round; logic state; data; data(hex)
	0; 7; 0; 1060; 424
	1; 7; 0; 1060; 424
	2; 7; 0; 1060; 424
	3; 7; 0; 1060; 424
	...
	-------------------------------------------------------------------------------------------------------
```

```
---
	*/
	// Setting a flag for skipping over first lines with no CAN frames
	firstLines = 1;
	len = 49; // first read is the headers & tags
	entriesConverted = 0;

	while( (read = fread(line, len, 1, file_in)) != -1 ) {
		if( read > 0 ) {
			// Got something, parsing contents
			if(firstLines) {
				memset((void *)tmpStr, 0x00, sizeof(tmpStr));
				if((pFileType = strchr((char *)&line[26], '"'))==NULL) {
					pFileType = tmpStr;
					strcat(pFileType, "FAIL");
				}
				else {
					strncpy(tmpStr, (char *)&line[26], (pFileType-(char *)&line[26]));
				}
				fwID = (line[2]<<8)+line[3];
				printf("file type = %s (%s), created by %d (%s)\n", tmpStr,
					((strcmp(tmpStr, "data log")==0)?"OK":"not OK"), fwID,
				(fwID==22016)?"OK":"not OK");
				printf("Converting file with timestamp: \"%d.%d.%d %02d:%02d\"\n", line[43],
line[42], line[41],
					line[44], line[45]);
				printf("latest write round = %d\n", line[46]);
				firstLines = 0;
				len = 2;

				memset((void *)tmpStr, 0x00, sizeof(tmpStr));
				snprintf(tmpStr, 128, "line; write round; logic state; data; data(hex)\n");
				fwrite((const void *)tmpStr, sizeof(char), strlen(tmpStr), file_out);
			}
			else {
				logEntryData = ((line[1]&0x0F)<<8)+line[0];
				wrRound = (line[1]&0xE0)>>5;
				outState = ((line[1] & 0x10) > 0);
				if(entriesConverted < 5)
				{
					printf("line %d: wrRound=%d, LogicState=%d, ", entriesConverted,
wrRound, outState);
					printf("data = %d (0x%x)\n", logEntryData, logEntryData);
				}
				else if(entriesConverted == 5) printf("...\n");

				memset((void *)tmpStr, 0x00, sizeof(tmpStr));
				snprintf(tmpStr, 128, "%d; %d; %d; %d; %x\n", entriesConverted, wrRound,
outState,
					logEntryData, logEntryData);
				fwrite((const void *)tmpStr, sizeof(char), strlen(tmpStr), file_out);

				entriesConverted++;
				firstLines=0;
			}
		}
		else if(read == 0) {
			// end of file.
			break;
		}
		else {
			printf("reading failed.\n");
			break;
		}
	}

	// Closing files
	fclose(file_in);
	fclose(file_out);

	printf("Converted %d log entries from bin log to csv log.\n", entriesConverted);

	return EXIT_SUCCESS;
}
```

OUR PRODUCTS

AC/DC Power Supplies

Actuator Controls/Interfaces

Automotive Ethernet Interfaces

Battery Chargers

CAN Controls, Routers, Repeaters

CAN/WiFi, CAN/Bluetooth, Routers

Current/Voltage/PWM Converters

DC/DC Power Converters

Engine Temperature Scanners

Ethernet/CAN Converters, Gateways, Switches

Fan Drive Controllers

Gateways, CAN/Modbus, RS-232

Gyroscopes, Inclinometers

Hydraulic Valve Controllers

Inclinometers, Triaxial

I/O Controls

LVDT Signal Converters

Machine Controls

Modbus, RS-422, RS-485 Controls

Motor Controls, Inverters

Power Supplies, DC/DC, AC/DC

PWM Signal Converters/Isolators

Resolver Signal Conditioners

Service Tools

Signal Conditioners, Converters

Strain Gauge CAN Controls

Surge Suppressors

OUR COMPANY

Axiomatic provides electronic machine control components to the off-highway, commercial vehicle, electric vehicle, power generator set, material handling, renewable energy and industrial OEM markets. ***We innovate with engineered and off-the-shelf machine controls that add value for our customers.***

QUALITY DESIGN AND MANUFACTURING

We have an ISO9001:2015 registered design/manufacturing facility in Canada.

WARRANTY, APPLICATION APPROVALS/LIMITATIONS

Axiomatic Technologies Corporation reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. Users should satisfy themselves that the product is suitable for use in the intended application. All our products carry a limited warranty against defects in material and workmanship. Please refer to our Warranty, Application Approvals/Limitations and Return Materials Process at https://www.axiomatic.com/service/.

COMPLIANCE

Product compliance details can be found in the product literature and/or on axiomatic.com. Any inquiries should be sent to sales@axiomatic.com.

SAFE USE

All products should be serviced by Axiomatic. Do not open the product and perform the service yourself.

This product can expose you to chemicals which are known in the State of California, USA to cause cancer and reproductive harm. For more information go to www.P65Warnings.ca.gov.

SERVICE

All products to be returned to Axiomatic require a Return Materials Authorization Number (RMA#) from rma@axiomatic.com. Please provide the following information when requesting an RMA number:

- Serial number, part number
- Runtime hours, description of problem
- Wiring set up diagram, application and other comments as needed

DISPOSAL

Axiomatic products are electronic waste. Please follow your local environmental waste and recycling laws, regulations and policies for safe disposal or recycling of electronic waste.

CONTACTS

Axiomatic Technologies Corporation
1445 Courtneypark Drive E.
Mississauga, ON
CANADA L5T 2E3
TEL: +1 905 602 9270
FAX: +1 905 602 9279
www.axiomatic.com
sales@axiomatic.com

Axiomatic Technologies Oy
Höytämöntie 6
33880 Lempäälä
FINLAND
TEL: +358 103 375 750
www.axiomatic.com
salesfinland@axiomatic.com

Copyright 2025