Intelligent Vehicle Controls

CL-713/714
CODESYS Instructions

TABLE OF CONTENTS

INSTALLATION PACKAGE 2
CREATING A PROJECT 5
ADDING I/O 6
I/O TYPES AND CONFIGURATION 9
- STB/STG/TRISTATE INPUTS 9
- RTD INPUT 10
- VTD INPUT 11
- PWM INPUT 11
- FREQENCY INPUT 12
- COUNTER INPUT 12
- BUTTONS 13
- BUTTON BACKLIGHT 14
- KEYSWITCH AND BATTERY INPUTS 14
- AMBIENT LIGHT SENSOR 15
- ENCODER INPUT 15
- DIGITAL OUTPUT 16
- PWM OUTPUT 16
- FREQUENCY OUTPUT 17
- ECC OUTPUT 18
- BACKLIGHT OUTPUT 19
- 5V POWER SUPPLY 19

CAN – SENDING AND RECEIVING CAN MESSAGES/SIGNALS 20
CREATING SCREENS 27
SWITCHING BETWEEN SCREENS IN YOUR APPLICATION 28
CHANGING OF SPLASH SCREEN 30
PROGRAMMING DISPLAY THROUGH CODESYS 31
PRE-PROGRAMMING OPTION 34

CL-713/714 CODESYS DOCUMENTATION

OVERVIEW

This document is meant to assist in guiding you through setting up and developing with a CL-713/714 display in CODESYS.

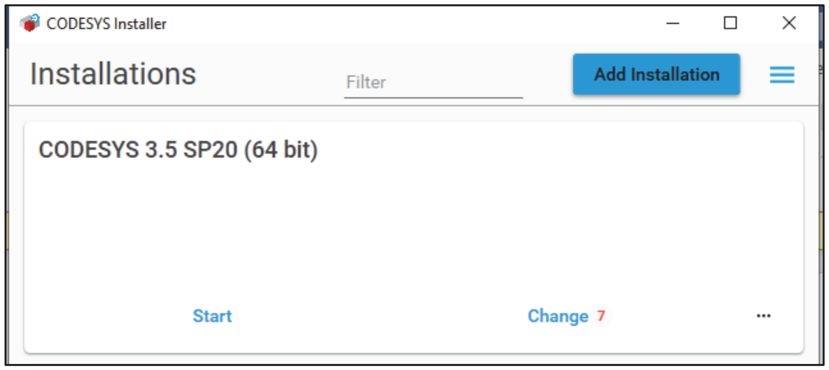
INSTALLING PACKAGE

To add the CL-713/714 device to CODESYS you will first need to obtain the .package file, which can be downloaded from the following location:

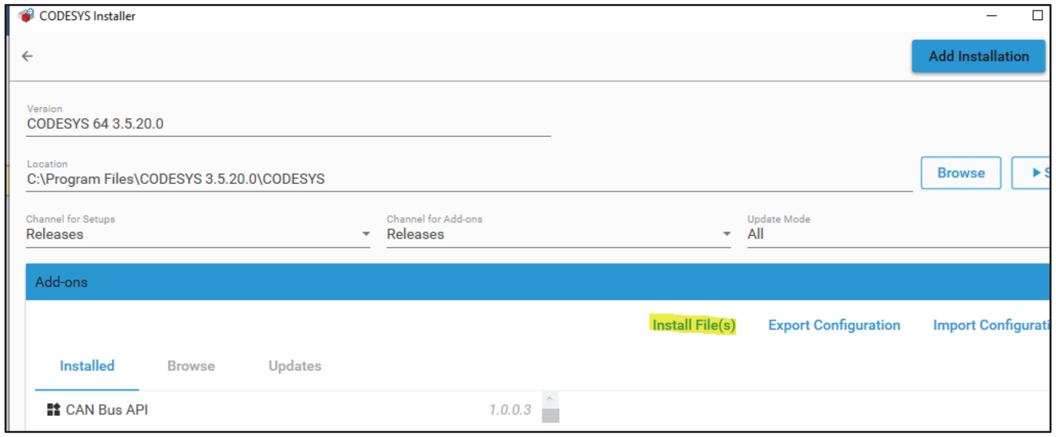
DOWNLOAD .PACKAGE FILE HERE

Once downloaded you will need to install it into your CODESYS environment.

1. Launch your CODESYS Installer



2. Select “Change” and then “Install File(s)”



Version
CODESYS 64 3.5.20.0

Location
C:\Program Files\CODESYS 3.5.20.0\CODESYS

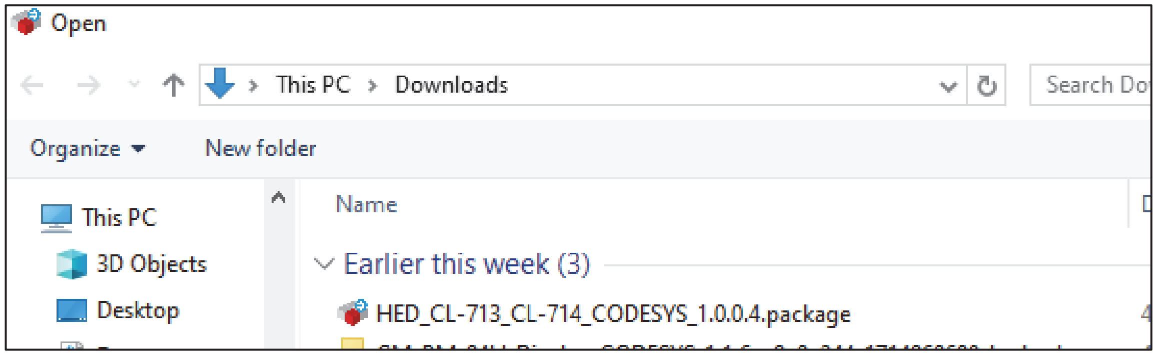
Channel for Setups
Releases

Channel for Add-ons
Releases

Update Mode
All

CAN Bus API 1.0.0.3

3. Navigate to and select the .package file that was downloaded

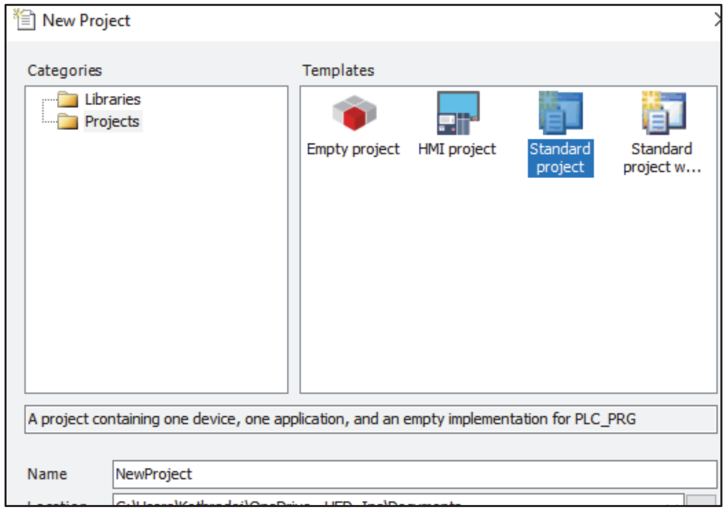


HED_CL-713_CL-714_CODESYS_1.0.0.4.package

Note: CODESYS may warn that the package file is not signed and prompts you if you would like to continue with the installation, continue with the installation. Once the process is complete you can launch CODESYS and have the ability to add this device to your project.

CREATING A PROJECT

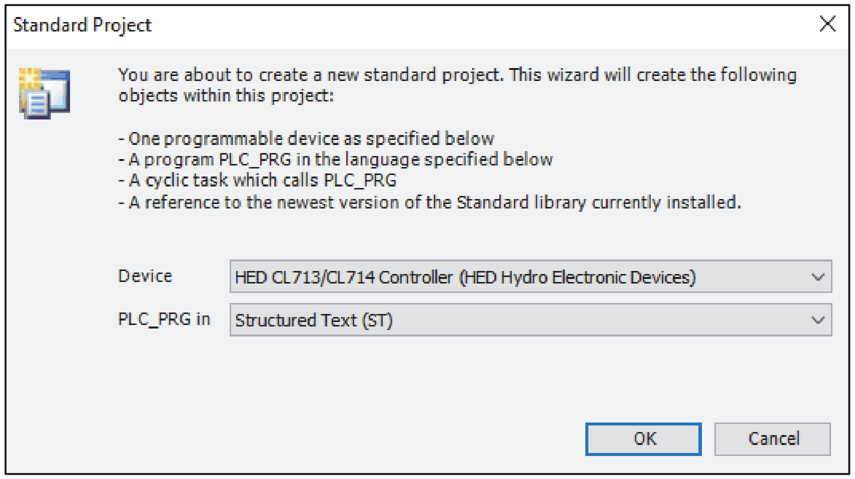
1. Launch CODESYS and select File->New Project
2. Select Standard Project



A project containing one device, one application, and an empty implementation for PLC_PRG

Name: NewProject

3. When it launches it will ask what Device and Language you would like to use, for the Device choose the HED CL-713/714 Controller



You are about to create a new standard project. This wizard will create the following objects within this project:

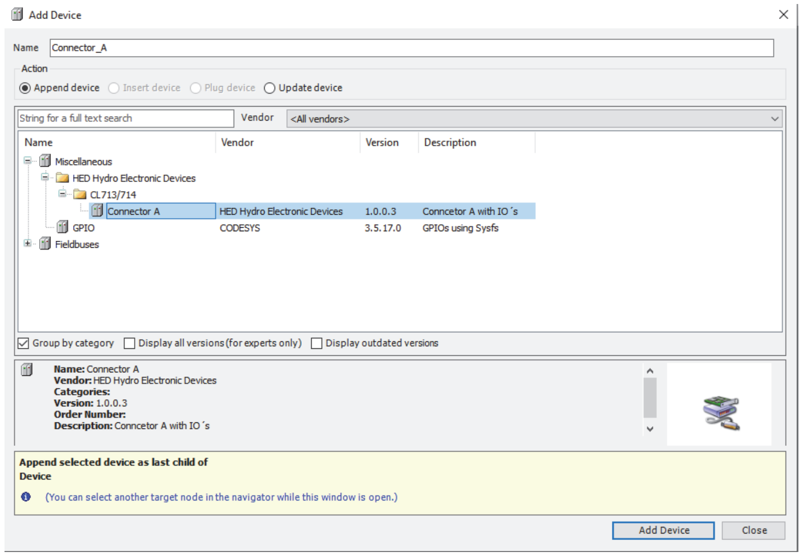
- One programmable device as specified below
- A program PLC_PRG in the language specified below
- A cyclic task which calls PLC_PRG
- A reference to the newest version of the Standard library currently installed.

Device: HED CL713/CL714 Controller (HED Hydro Electronic Devices)

PLC_PRG in: Structured Text (ST)

ADDING I/O

1. To add I/O to the CL-713/714 right click the top-level device and then select Add Device
2. Under Miscellaneous follow the tree down until you see Connector A. Select that and click Add Device.



Add Device

Name: Connector_A

Action: [✓ Append device] Insert device Plug device Update device

String for a full text search Vendor <All vendors>

Name	Vendor	Version	Description
Miscellaneous			
HED Hydro Electronic Devices			
CL713/714			
Connector A	HED Hydro Electronic Devices	1.0.0.3	Conncetor A with IO´s
GPIO	CODESYS	3.5.17.0	GPIOs using Sysfs
Fieldbuses			

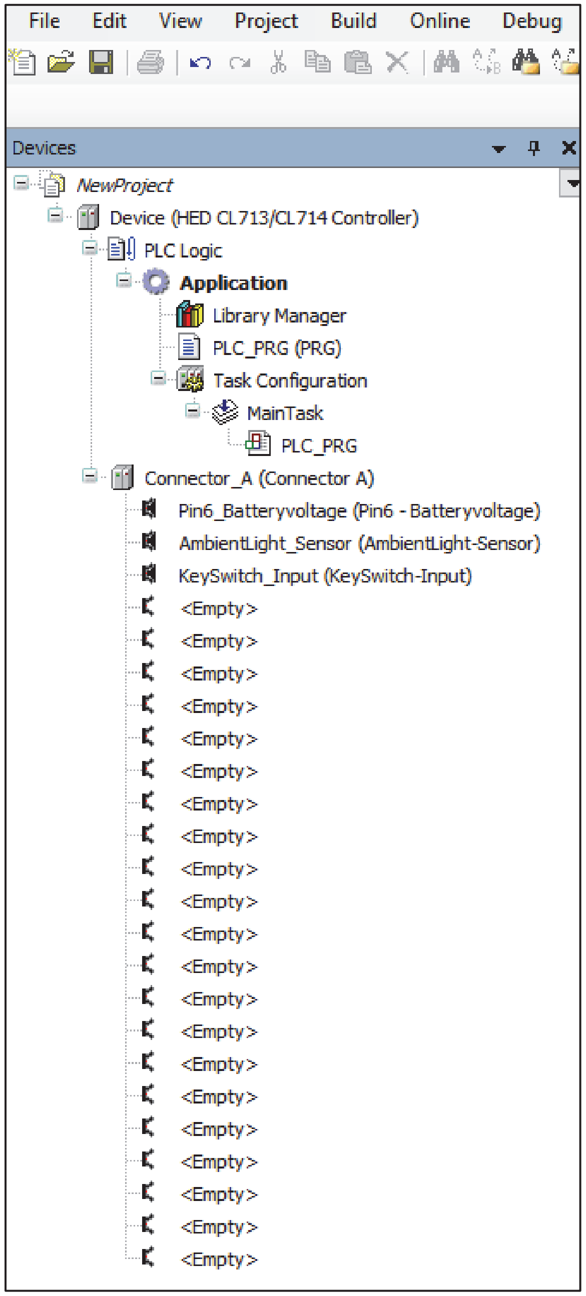
[✓] Group by category [ ] Display all versions (for experts only) [ ] Display outdated versions

Name: Connector A
Vendor: HED Hydro Electronic Devices
Categories:
Version: 1.0.0.3
Order Number:
Description: Conncetor A with IO´s

Append selected device as last child of
Device

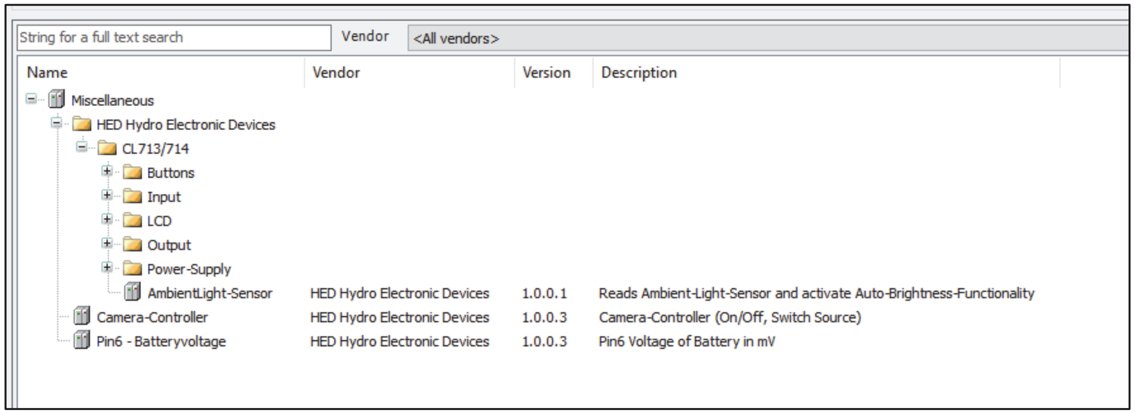
(You can select another target node in the navigator while this window is open.)

3. After being added your Device tree should look similar to this:



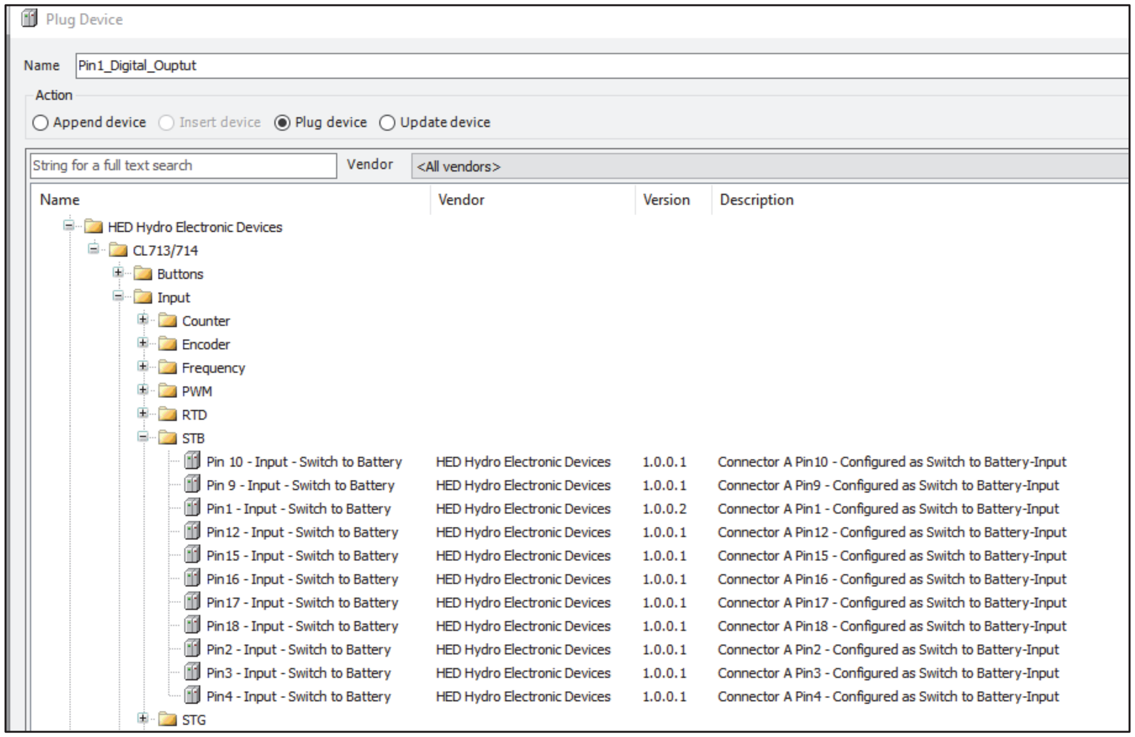
NewProject
Device (HED CL713/CL714 Controller)
PLC Logic
Application
Library Manager
PLC_PRG (PRG)
Task Configuration
MainTask
PLC_PRG
Connector_A (Connector A)
Pin6_Batteryvoltage (Pin6 - Batteryvoltage)
AmbientLight_Sensor (AmbientLight-Sensor)
KeySwitch_Input (KeySwitch-Input)
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>
<Empty>

4. To add IO you can right click on any of the <Empty> sockets, select Plug Device and choose an IO from the list


String for a full text search | Vendor <All vendors>

Name	Vendor	Version	Description
Miscellaneous			
HED Hydro Electronic Devices			
CL713/714			
Buttons			
Input			
LCD			
Output			
Power-Supply			
AmbientLight-Sensor	HED Hydro Electronic Devices	1.0.0.1	Reads Ambient-Light-Sensor and activate Auto-Brightness-Functionality
Camera-Controller	HED Hydro Electronic Devices	1.0.0.3	Camera-Controller (On/Off, Switch Source)
Pin6 - Batteryvoltage	HED Hydro Electronic Devices	1.0.0.3	Pin6 Voltage of Battery in mV

Each IO has a Pin associated with it letting you know what pins can be configured for what IO:


Plug Device

Name: Pin1_Digital_Ouptut

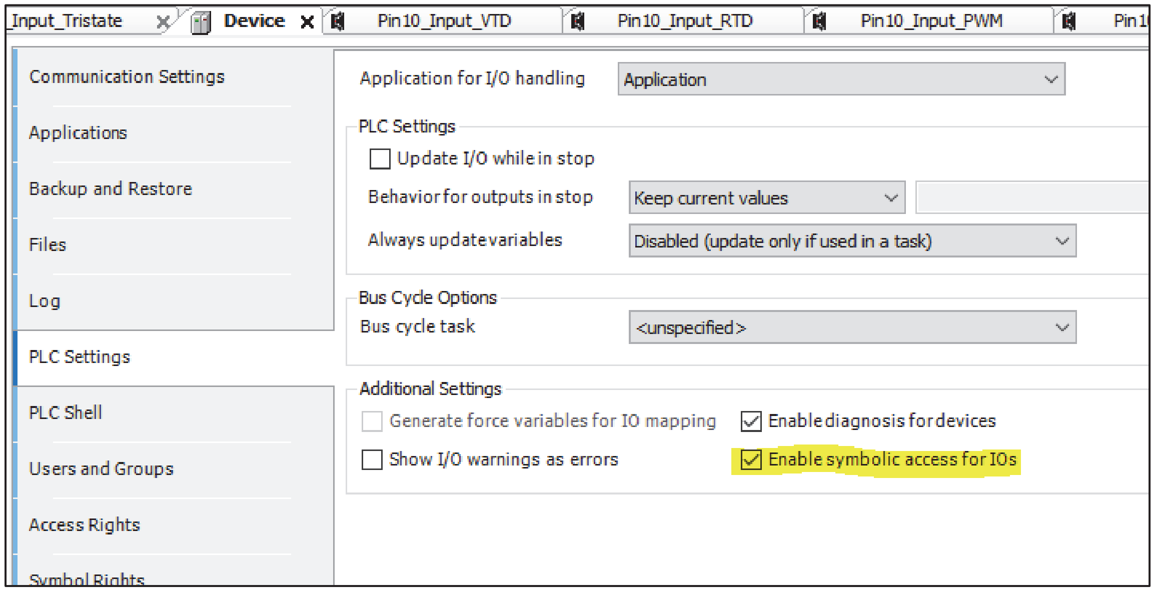
Action: ( ) Append device ( ) Insert device (●) Plug device ( ) Update device

String for a full text search | Vendor <All vendors>

Name	Vendor	Version	Description
HED Hydro Electronic Devices			
CL713/714			
Buttons			
Input			
Counter			
Encoder			
Frequency			
PWM			
RTD			
STB			
Pin 10 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin10 - Configured as Switch to Battery-Input
Pin 9 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin9 - Configured as Switch to Battery-Input
Pin1 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.2	Connector A Pin1 - Configured as Switch to Battery-Input
Pin12 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin12 - Configured as Switch to Battery-Input
Pin15 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin15 - Configured as Switch to Battery-Input
Pin16 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin16 - Configured as Switch to Battery-Input
Pin17 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin17 - Configured as Switch to Battery-Input
Pin18 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin18 - Configured as Switch to Battery-Input
Pin2 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin2 - Configured as Switch to Battery-Input
Pin3 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin3 - Configured as Switch to Battery-Input
Pin4 - Input - Switch to Battery	HED Hydro Electronic Devices	1.0.0.1	Connector A Pin4 - Configured as Switch to Battery-Input
STG			

As a note, you can configure a pin with an input and an output and the system will not flag an error so be careful and keep track of what pins are configured

5. It is highly recommended that you adjust the PLC Settings to allow the use of symbolic access of IO to make programming easier. You can do this by double clicking the CL-713/714 Device and going to the PLC Settings.



Application for I/O handling: Application

PLC Settings
- [ ] Update I/O while in stop
- Behavior for outputs in stop: Keep current values
- Always update variables: Disabled (update only if used in a task)

Bus Cycle Options
- Bus cycle task: <unspecified>

Additional Settings
- [ ] Generate force variables for IO mapping
- [✓] Enable diagnosis for devices
- [ ] Show I/O warnings as errors
- [✓] Enable symbolic access for IOs

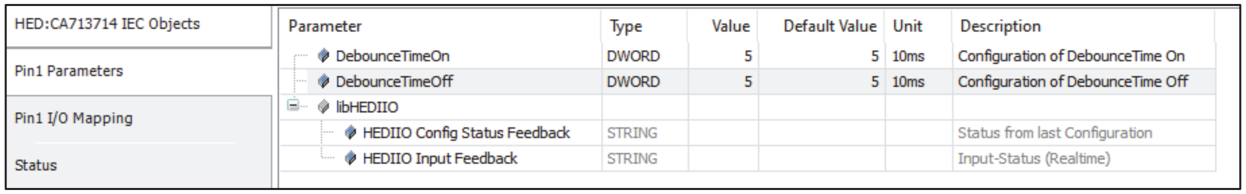
Keep in mind that this setting is a per project setting and will need to be done for every new project that you do (the property is tied to project not workspace).

I/O TYPES AND CONFIGURATION

STB/STG/TRISTATE INPUTS

STB, STG, and TRISTATE inputs refer to digital inputs where the input is ON when connected to their designated type and OFF when not i.e. STB is ON when it sees battery voltage and OFF when it see ground and vice versa for STG, it is ON when it sees ground and OFF when it sees battery voltage. TRISTATE specifically can detect when connected to a source, ground, or open.

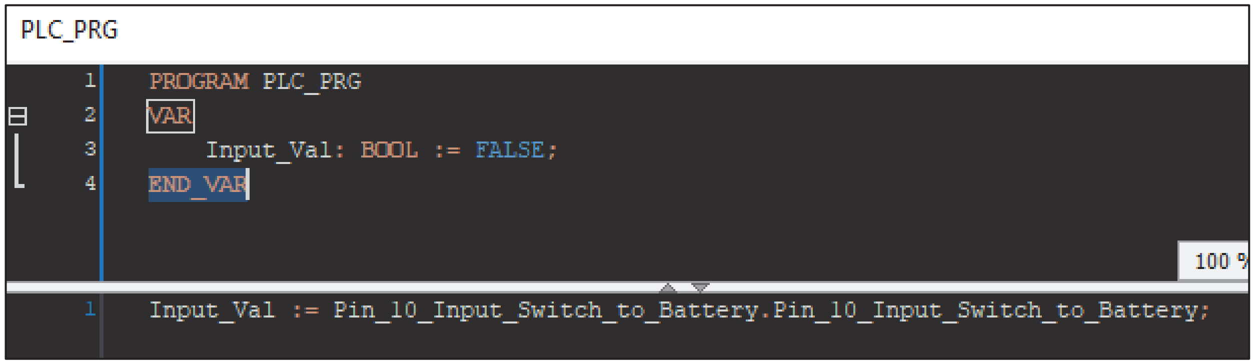
All three of these inputs have 2 properties associated with them for configuration, DebounceTimeOn and DebounceTimeOff



Parameter	Type	Value	Default Value	Unit	Description
DebounceTimeOn	DWORD	5	5	10ms	Configuration of DebounceTime On
DebounceTimeOff	DWORD	5	5	10ms	Configuration of DebounceTime Off
libHEDIIO					
HEDIIO Config Status Feedback	STRING				Status from last Configuration
HEDIIO Input Feedback	STRING				Input-Status (Realtime)

Both of these values default to 5 with a unit of 10mS, so by default the debounce time for both is 50mS. Debounce is a property that the input must see the value for at least this long to be considered ON/OFF.

For example, a STB input sees battery voltage, it must see that voltage for at least 50mS before the input will consider it valid and report the input as ON. With the symbolic IO enabled, accessing the input value would be done from looking at the input property.

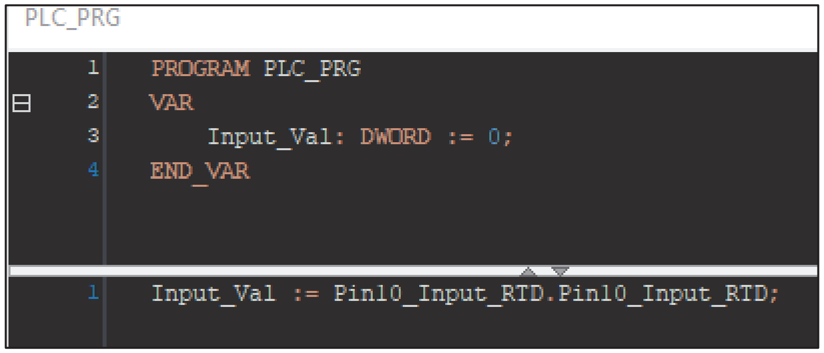


```
PLC_PRG
PROGRAM PLC_PRG
VAR
    Input_Val: BOOL := FALSE;
END_VAR

Input_Val := Pin_10_Input_Switch_to_Battery.Pin_10_Input_Switch_to_Battery;
```

RTD INPUT

RTD inputs refer to inputs that read in a resistance value and report back the resistance in ohms. This input has 1 property, Analog HS Filter Frequency. That property should be set to 0 so that the input is read only once per loop. With symbolic IO enabled, accessing the input would be done from the Input_RTD property.

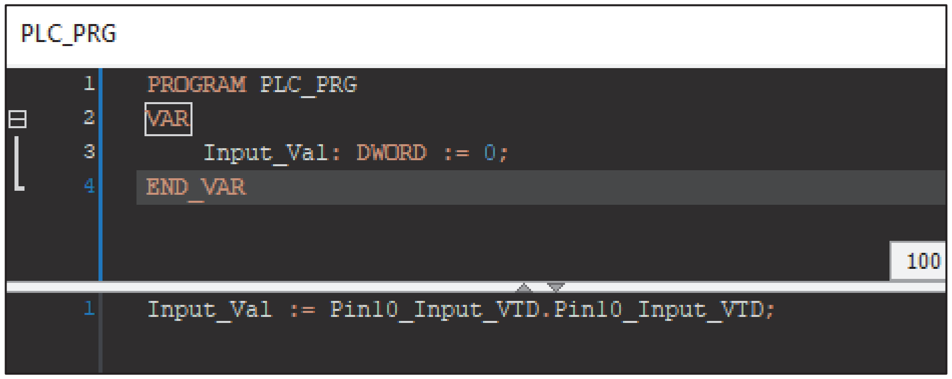


```
PLC_PRG
PROGRAM PLC_PRG
VAR
    Input_Val: DWORD := 0;
END_VAR

Input_Val := Pin10_Input_RTD.Pin10_Input_RTD;
```

VTD INPUT

VTD inputs refer to inputs that read in a voltage and report back what that voltage is in mV. Generally, this is a 0-5V range with the exception of the Battery Voltage Pin. This input has 1 property, Analog HS Filter Frequency. That property should be set to 0 so that the input is read only once per loop. With symbolic IO enabled, accessing the input would be done from the Input_VTD property.

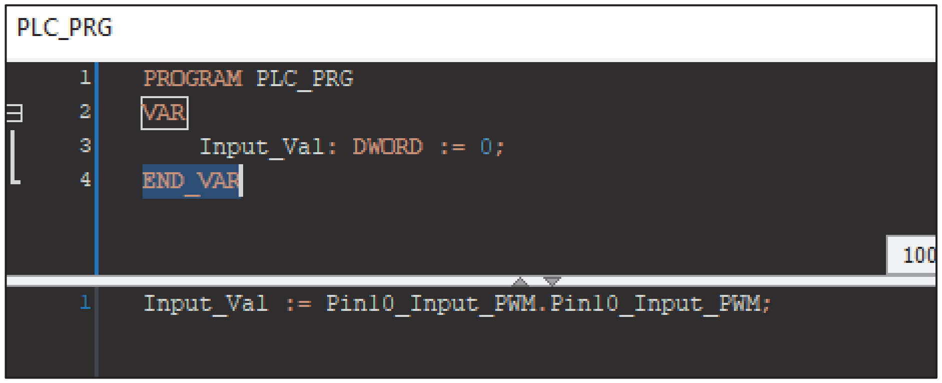

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    Input_Val: DWORD := 0;
END_VAR
```

```
Input_Val := Pin10_Input_VTD.Pin10_Input_VTD;
```

PWM INPUT

PWM inputs read and report back the on-percentage duty cycle of a square wave signal in a percentage of 0-100.0% (real value reported is 0-1000). This input type currently has no adjustable parameters. Using the symbolic IO you can access the input value by looking at the Input_PWM property.

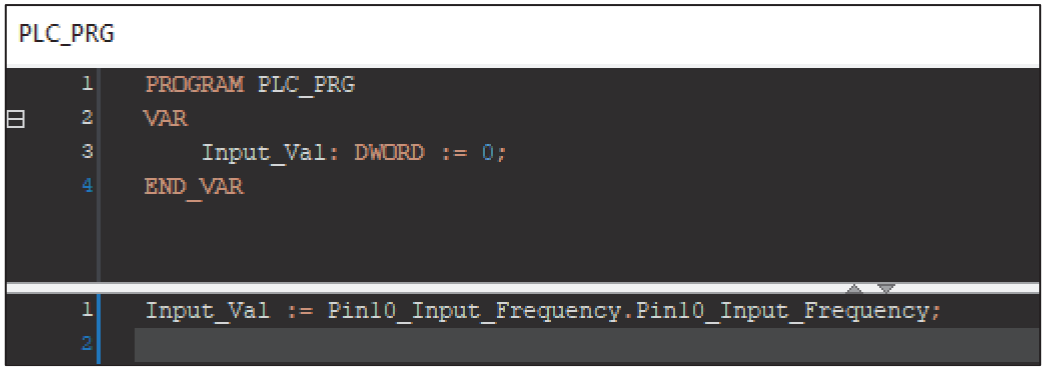

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    Input_Val: DWORD := 0;
END_VAR
```

```
Input_Val := Pin10_Input_PWM.Pin10_Input_PWM;
```

FREQENCY INPUT

Frequency inputs read and report back the frequency of a square wave signal in Hz. This input type currently has no adjustable parameters. Having the symbolic IO enabled you can access the input value by looking at the _Input_Frequency property.



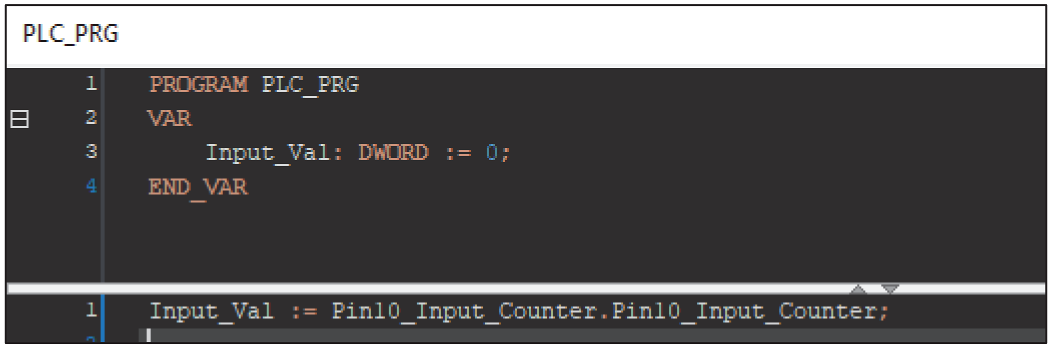
PLC_PRG

```
PROGRAM PLC_PRG
VAR
    Input_Val: DWORD := 0;
END_VAR
```

```
Input_Val := Pin10_Input_Frequency.Pin10_Input_Frequency;
```

COUNTER INPUT

The counter input is a pulse counter that will report back the number of square wave pulses the input experiences over the loop time of the application. The counts will be accumulated based on the falling edge detected. There aren't any adjustable parameters for this input type. If you have the symbolic IO enabled you can read the number of pulses by accessing the _Input_Counter property.



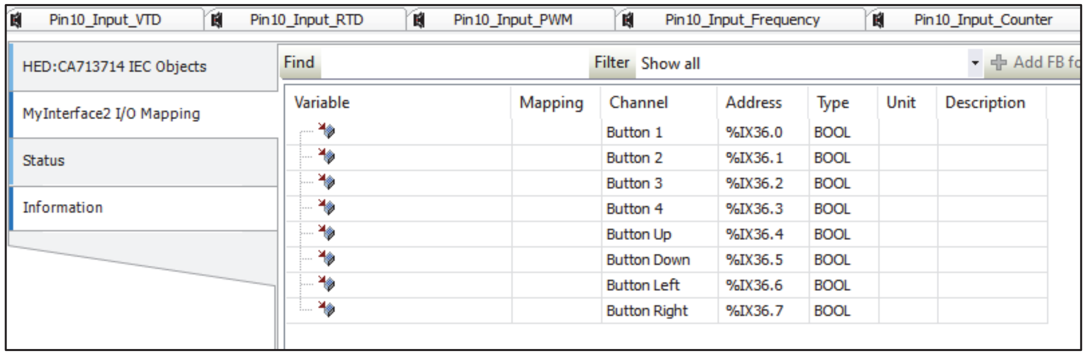
PLC_PRG

```
PROGRAM PLC_PRG
VAR
    Input_Val: DWORD := 0;
END_VAR
```

```
Input_Val := Pin10_Input_Counter.Pin10_Input_Counter;
```

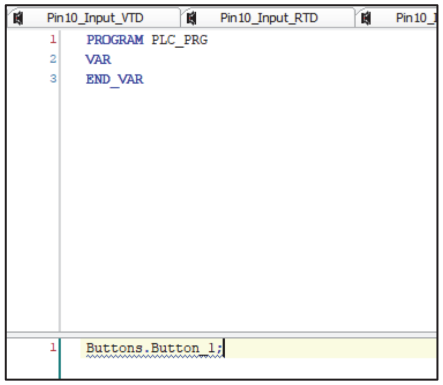
BUTTONS

The buttons are grouped together into a single socket called Buttons. Add them to a socket just as you would for any other I/O (Right click Plug Device). Double clicking on the object you can view the I/O Mapping to see all 4 buttons and the 4 Navigation buttons pre-defined.



Variable	Mapping	Channel	Address	Type	Unit	Description
		Button 1	%IX36.0	BOOL		
		Button 2	%IX36.1	BOOL		
		Button 3	%IX36.2	BOOL		
		Button 4	%IX36.3	BOOL		
		Button Up	%IX36.4	BOOL		
		Button Down	%IX36.5	BOOL		
		Button Left	%IX36.6	BOOL		
		Button Right	%IX36.7	BOOL		

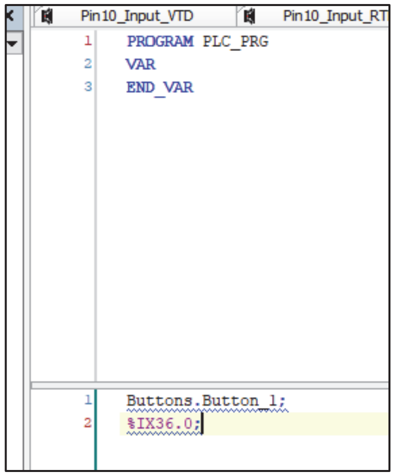
If you set up your device to utilize the symbolic IO you can reference the buttons themselves based on the Channel Name



```
PROGRAM PLC_PRG
VAR
END_VAR
```

```
Buttons.Button_1;
```

Otherwise, you will have to reference them by their Address which can change depending on what socket you plugged the Buttons into on the Connector A Device.

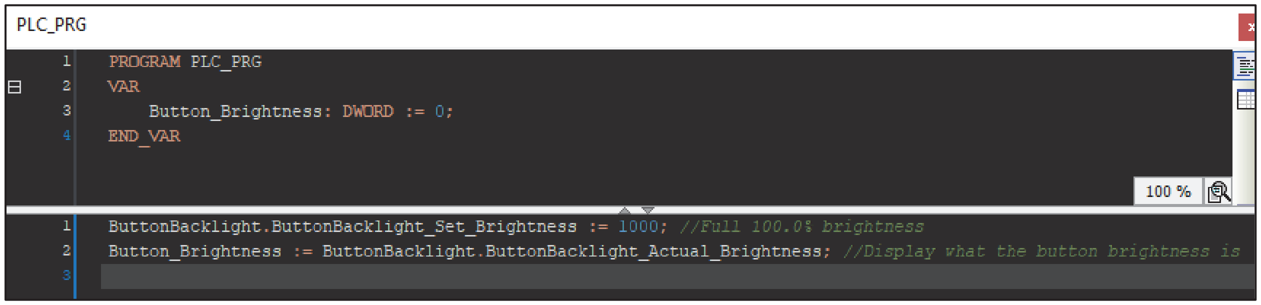


```
PROGRAM PLC_PRG
VAR
END_VAR
```

```
Buttons.Button_1;
%IX36.0;
```

BUTTON BACKLIGHT

This is an output that allows you to control the backlight brightness of the buttons. If you have the symbolic IO mapping enabled you can access the 2 properties easily to assign a value to. The ButtonBacklight_Set_Brightness is what you would assign a value of 0-1000 (100.0%) to control the backlight. The second property _Actual_Brightness is what the driver is actually setting the backlight to and would only differ from the _Set_Brightness if the controller was throttling the backlight due to thermal conditions. Use the _Actual_Brightness for displaying what the backlight is on a visualization and use _Set_Brightness to actually control the backlight.

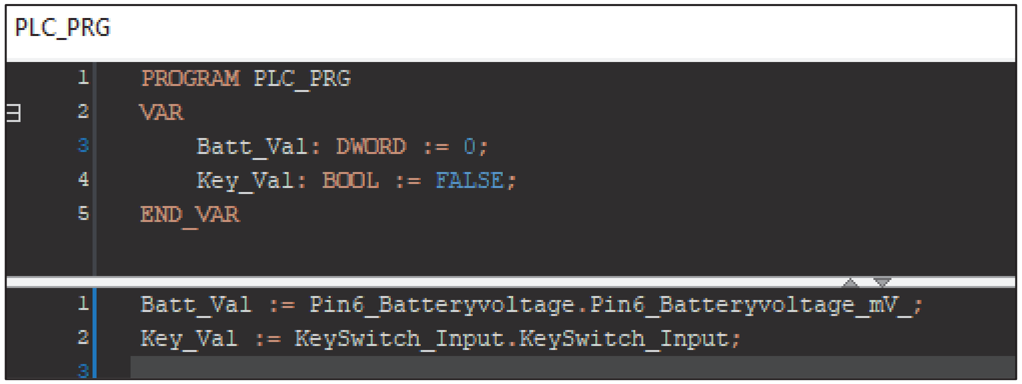


```
PROGRAM PLC_PRG
VAR
    Button_Brightness: DWORD := 0;
END_VAR
```

```
ButtonBacklight.ButtonBacklight_Set_Brightness := 1000; //Full 100.0% brightness
Button_Brightness := ButtonBacklight.ButtonBacklight_Actual_Brightness; //Display what the button brightness is
```

KEYSWITCH AND BATTERY INPUTS

The Keyswitch and Battery Inputs are already pre-socketed into the project when you add the CL-713/CL-714 device. These monitor battery voltage (in mV) and the switched power pin (TRUE/FALSE for detection) for you. With the symbolic IO enabled you can access the values read by them by looking at their respective properties.

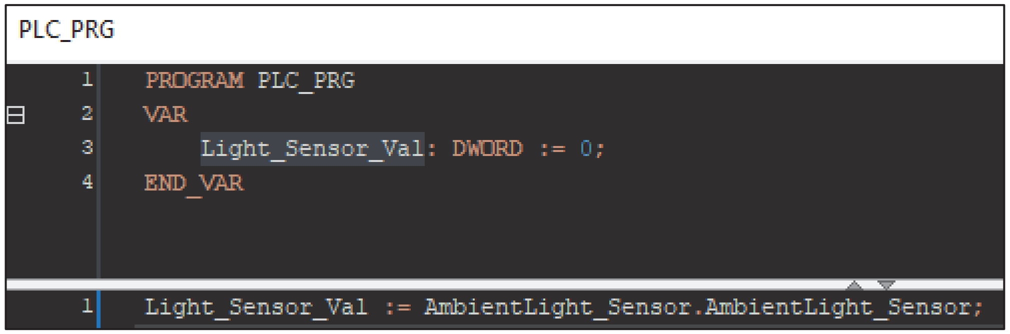

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    Batt_Val: DWORD := 0;
    Key_Val: BOOL := FALSE;
END_VAR
```

```
Batt_Val := Pin6_Batteryvoltage.Pin6_Batteryvoltage_mV_;
Key_Val := KeySwitch_Input.KeySwitch_Input;
```

AMBIENT LIGHT SENSOR

The ambient light sensor is another input that is pre-socketed into the project upon adding the CL-713 device. This is a sensor that reports back a reading, in lux, based on the amount of light it is exposed to where the higher the value the more light it is exposed to. With the symbolic IO enabled you can access the value read by the sensor but looking at the sensor property.

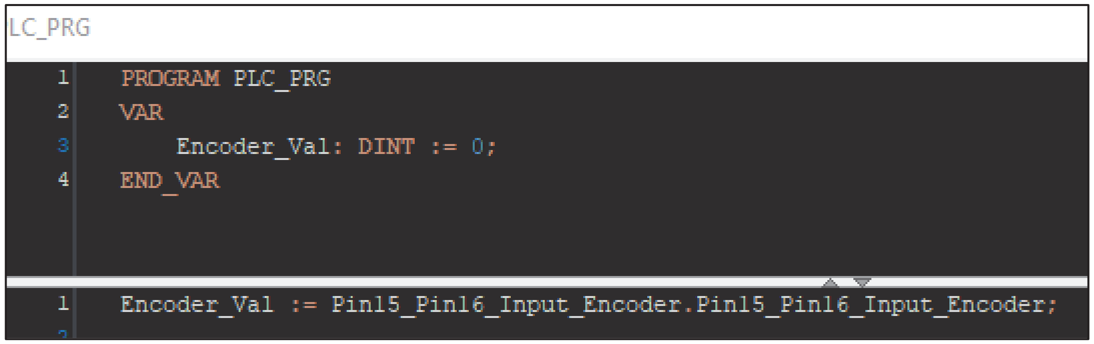

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    Light_Sensor_Val: DWORD := 0;
END_VAR
```

```
Light_Sensor_Val := AmbientLight_Sensor.AmbientLight_Sensor;
```

ENCODER INPUT

The encoder inputs are a pair of pins that act as 1 input type. These inputs do not have any configurable parameters and reading it will return a signed value giving you the amount of change and in which direction for that given application loop cycle. You will need to create and maintain your own count within your code. With the symbolic IO enabled you can obtain the value by looking at the input parameter.



LC_PRG

```
PROGRAM PLC_PRG
VAR
    Encoder_Val: DINT := 0;
END_VAR
```

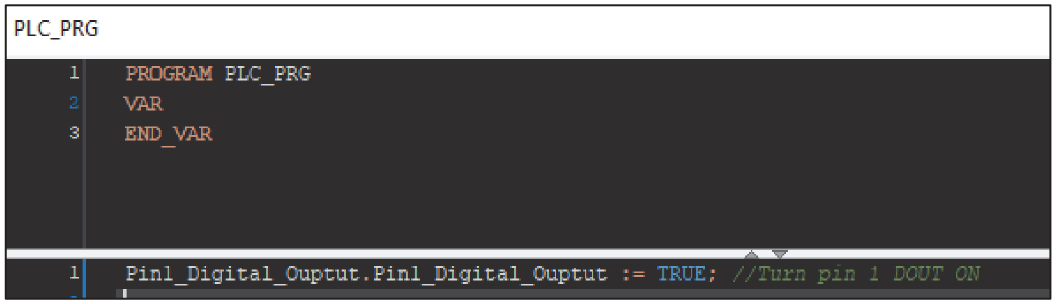
```
Encoder_Val := Pin15_Pin16_Input_Encoder.Pin15_Pin16_Input_Encoder;
```

DIGITAL OUTPUT

Digital outputs are controlled to be ON and OFF without any variance between. These outputs will have a few configuration parameters:

- Softwarefuse trip point: The value in mA in which a software fuse will be set at so that when the output is detected to go over that point it will trigger an overcurrent error.
- Softwarefuse Debounce time: The time in 10mS in which the output current needs to be above the trip point for the overcurrent error to trip.
- OpenCircuit-Detection: A value in mA in which the open circuit detection will turn on when below it.
- Open when Off-Detection-Mode: Can enable or disable the detection for open circuit when off.
- Load Type: Can enable certain protections on the pin based on the type of load, Load NA and Non-Inductive are standard loads and lastly if you are connecting to an inductive load ensure you use the Inductive Load type.

With the symbolic IO enabled, all that needs to happen to turn the output on or off would be to set the Boolean parameter true or false.



PLC_PRG

```
PROGRAM PLC_PRG
VAR
END_VAR
```

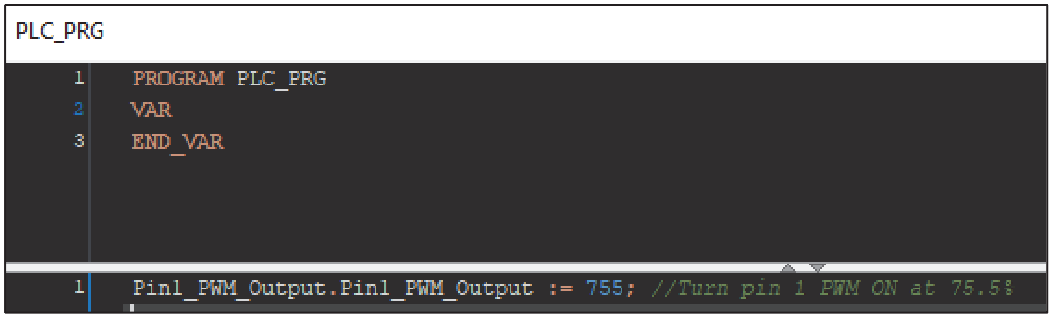
```
Pin1_Digital_Ouptut.Pin1_Digital_Ouptut := TRUE; //Turn pin 1 DOUT ON
```

PWM OUTPUT

The PWM output is an output that has a fixed frequency and a variable duty cycle that you command from 0-1000 (0-100.0%). This output type has some configurable parameters:

- Softwarefuse trip point: The value in mA in which a software fuse will be set at so that when the output is detected to go over that point it will trigger an overcurrent error.
- Softwarefuse Debounce time: The time in 10mS in which the output current needs to be above the trip point for the overcurrent error to trip.
- OpenCircuit-Detection: A value in mA in which the open circuit detection will turn on when below it.
- Open when Off-Detection-Mode: Can enable or disable the detection for open circuit when off.
- Load Type: Can enable certain protections on the pin based on the type of load, Load NA and Non-Inductive are standard loads and lastly if you are connecting to an inductive load ensure you use the Inductive Load type.
- Frequency: The frequency in which you desire the output to operate at ranging from 50-1000Hz.

With the symbolic IO enabled, all that needs to happen to turn the output on or off would be to set the output parameter to a value ranging from 0-1000.


PLC_PRG

```
PROGRAM PLC_PRG
VAR
END_VAR
```

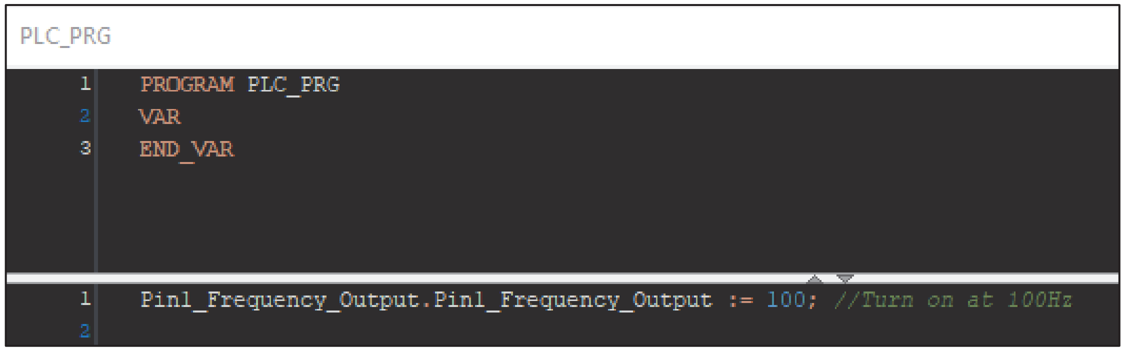
```
Pin1_PWM_Output.Pin1_PWM_Output := 755; //Turn pin 1 PWM ON at 75.5%
```

FREQUENCY OUTPUT

The Frequency output is an output that has a fixed Duty Cycle and a variable Frequency that you command up to 1000Hz. This output type has some configurable parameters:

- Softwarefuse trip point: The value in mA in which a software fuse will be set at so that when the output is detected to go over that point it will trigger an overcurrent error.
- Softwarefuse Debounce time: The time in 10mS in which the output current needs to be above the trip point for the overcurrent error to trip.
- OpenCircuit-Detection: A value in mA in which the open circuit detection will trip.
- Open when Off-Detection-Mode: Can enable or disable the detection for open circuit when off.
- Load Type: Can enable certain protections on the pin based on the type of load, Load NA and Non-Inductive are standard loads and lastly if you are connecting to an inductive load ensure you use the Inductive Load type.
- Duty Cycle: Set the Duty Cycle of the signal from 100 to 900 (10.0-90.0%)

With the symbolic IO enabled, all that needs to happen to turn the output on or off would be to set the output parameter to a value ranging from 0-1000 and it will operate at the configured duty cycle with the commanded frequency.



PLC_PRG

```
PROGRAM PLC_PRG
VAR
END_VAR
```

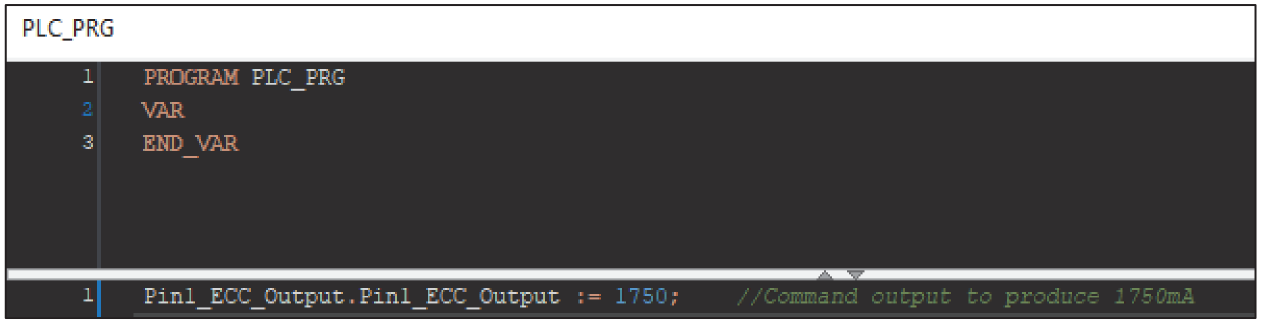
```
Pin1_Frequency_Output.Pin1_Frequency_Output := 100; //Turn on at 100Hz
```

ECC OUTPUT

The ECC output is an estimated constant current output, meaning it will try to output a specific amount of current that the user commands, in mA. Since there is no feedback to measure the actual current, the estimated value is determined internally through circuitry and algorithms. This output has a few things to configure with a bit of a nuance.

- Softwarefuse trip point: The value in mA in which a software fuse will be set at so that when the output is detected to go over that point it will trigger an overcurrent error.
- Softwarefuse Debounce time: The time in 10mS in which the output current needs to be above the trip point for the overcurrent error to trip.
- OpenCircuit-Detection: A value in mA in which the open circuit detection will turn on when below it.
- Open when Off-Detection-Mode: Can enable or disable the detection for open circuit when off.
- Load Type: This output will require the load type to be set so you must use the Inductive or Non Inductive load type to allow the calculations to run correctly.
- Frequency: The frequency in which you desire the output to operate at ranging from 50-1000Hz.
- Duty Cycle Offset: This property is an offset to start the PID output command at a higher setpoint to allow for slightly faster response/ramp up to target.
- P gain: The P Gain term for the internal PID control loop that will run to maintain a constant current output.
- I gain: The I Gain term for the internal PID control loop that will run to maintain a constant current output.
- D gain: The D Gain term for the internal PID control loop that will run to maintain a constant current output.

Having the symbolic IO enabled you can command a current by setting the property to the value you desire.



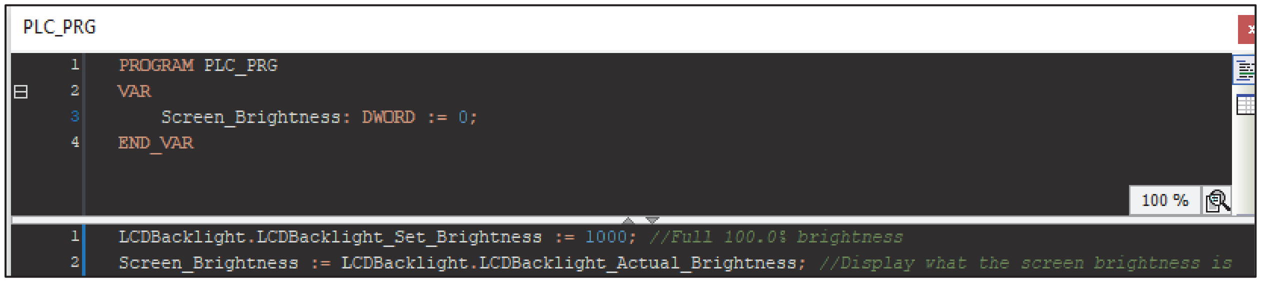
PLC_PRG

```
PROGRAM PLC_PRG
VAR
END_VAR
```

```
Pin1_ECC_Output.Pin1_ECC_Output := 1750;     //Command output to produce 1750mA
```

BACKLIGHT OUTPUT

This is an output that allows you to control the backlight brightness of the display. If you have the symbolic IO mapping enabled you can access the 2 properties easily to assign a value to. The LCDBacklight_Set_Brightness is what you would assign a value of 0-1000 (100.0%) to control the backlight. The second property _Actual_Brightness is what the driver is actually setting the backlight to and would only differ from the _Set_Brightness if the controller was throttling the backlight due to thermal conditions. Use the _Actual_Brightness for displaying what the backlight is on a visualization and use _Set_Brightness to actually control the backlight.



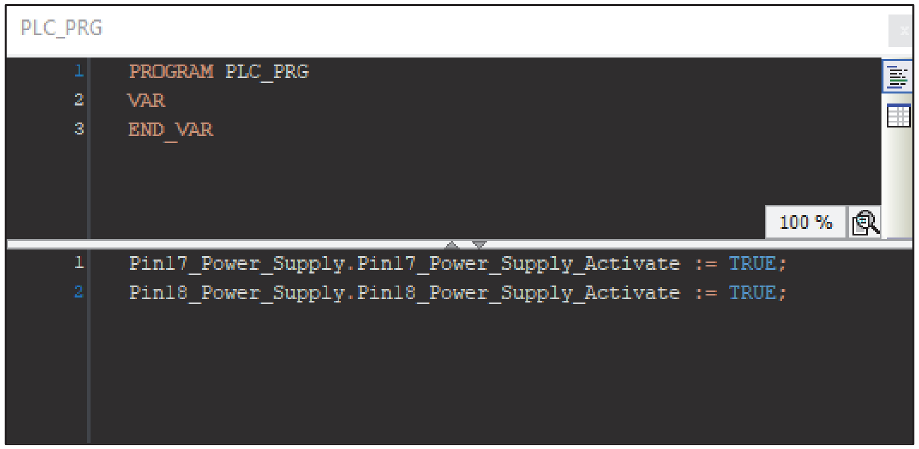
PLC_PRG

```
PROGRAM PLC_PRG
VAR
    Screen_Brightness: DWORD := 0;
END_VAR
```

```
LCDBacklight.LCDBacklight_Set_Brightness := 1000; //Full 100.0% brightness
Screen_Brightness := LCDBacklight.LCDBacklight_Actual_Brightness; //Display what the screen brightness is
```

5V POWER SUPPLY

The 5V supply is actually 2 pins, a supply (Pin 18) and a return (Pin 17). You will need to plug both into sockets in order to use this feature. Both of the pins will have a min and max voltage setting, the min must remain below the max. You will want to set your min and max for both pins and then use the _Activate property to turn them on (both pins need to be turned on for it to work properly).



```
PROGRAM PLC_PRG
VAR
END_VAR
```

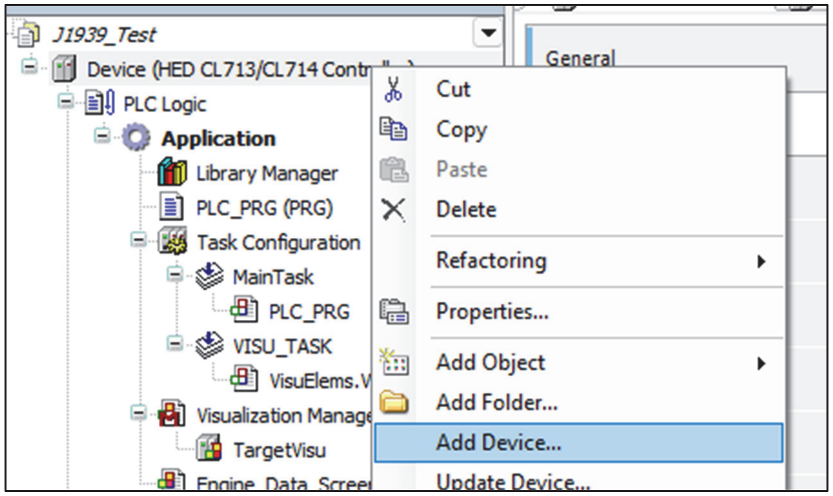
```
Pin17_Power_Supply.Pin17_Power_Supply_Activate := TRUE;
Pin18_Power_Supply.Pin18_Power_Supply_Activate := TRUE;
```

CAN – SENDING AND RECEIVING CAN MESSAGES/SIGNALS

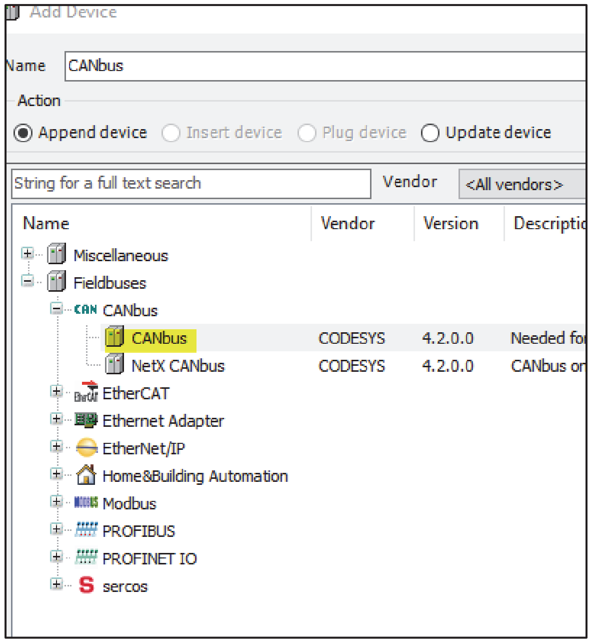
There are a few ways of sending and receiving CAN signals but the way we suggest will be through the use of a J1939 CAN library stack that is already included with the CL-713/714 Device.

Before sending or receiving CAN messages, you need to define the CAN bus devices.

1. Right click on the HED CL713/CL714 Controller and select Add Device



2. It will open a window where you can expand Fieldbuses. Under Fieldbuses you can select CANbus then select CANbus (not NetX CANbus).


Add Device

Name: CANbus

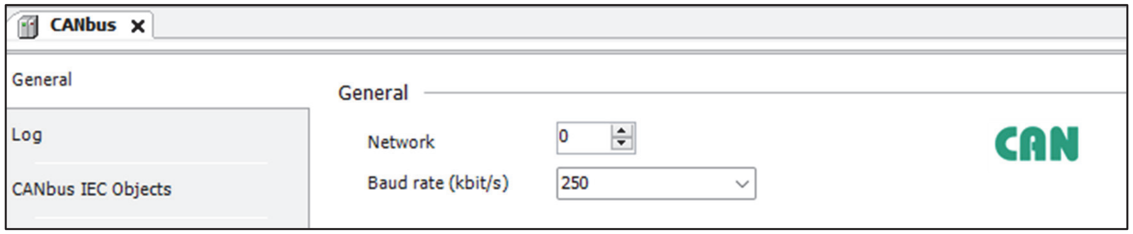
Action

Append device [✓] | Insert device | Plug device | Update device

String for a full text search | Vendor: <All vendors>

Name	Vendor	Version	Descriptic
Miscellaneous			
Fieldbuses			
CANbus			
CANbus	CODESYS	4.2.0.0	Needed fo
NetX CANbus	CODESYS	4.2.0.0	CANbus or
EtherCAT			
Ethernet Adapter			
EtherNet/IP			
Home&Building Automation			
Modbus			
PROFIBUS			
PROFINET IO			
sercos			

3. Double click on your CANbus device to bring up its properties editor. Under the General tab is where you select the CAN line and set the baud rate (CAN 1 is Network 0; CAN 2 is Network 1).


CANbus

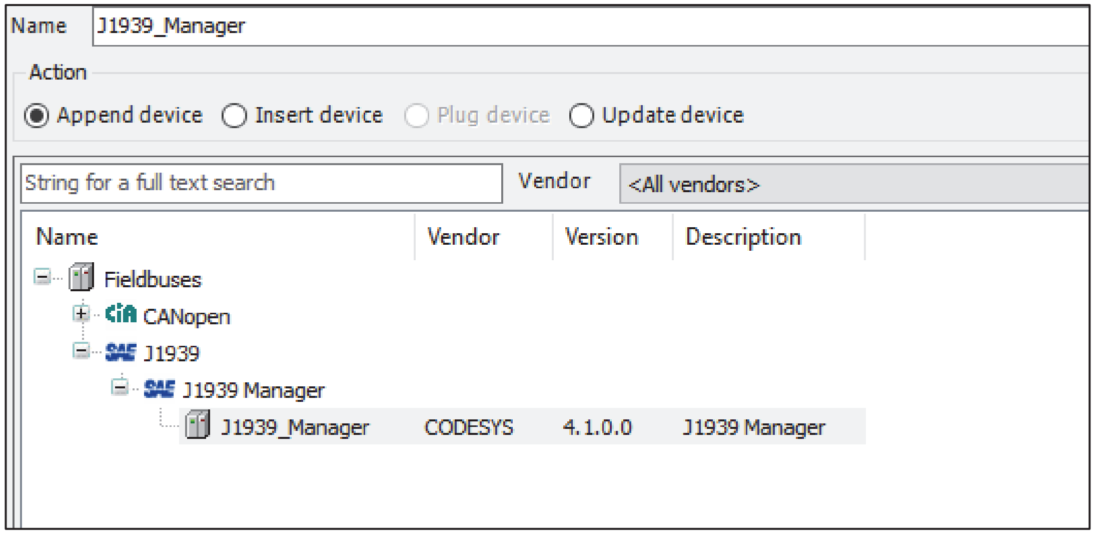
General | Log | CANbus IEC Objects

General

Network: 0

Baud rate (kbit/s): 250

4. Once configured, you will need to add the J1939 Device by right clicking on the CANbus Device and selecting Add Device. You now have the option to add a J1939 Manager or a CANopen Manager. If you have an EDS file, select CANopen. For engine/chassis interfaces select SAE J1939 Manager.

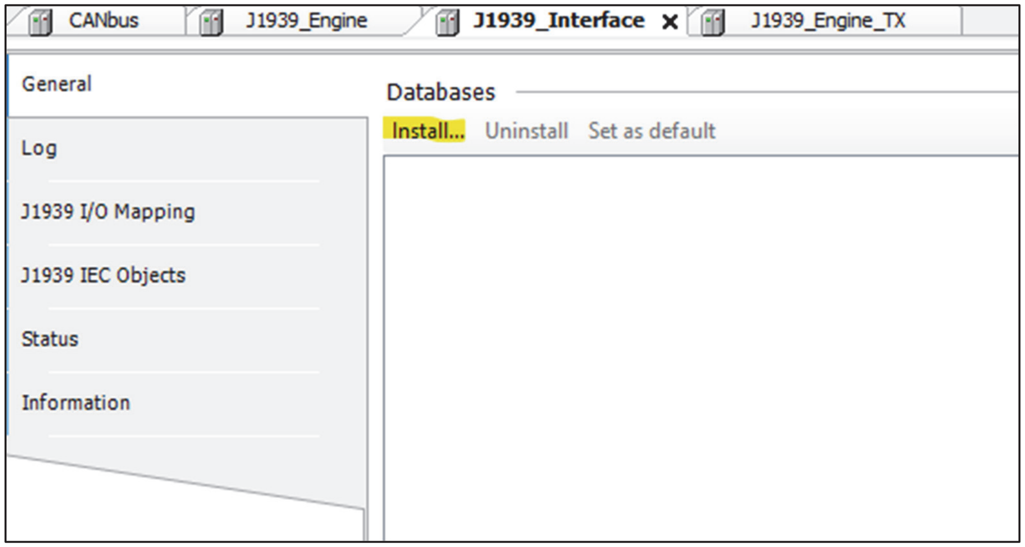

Name: J1939_Manager

Action: [✓ Append device] Insert device Plug device Update device

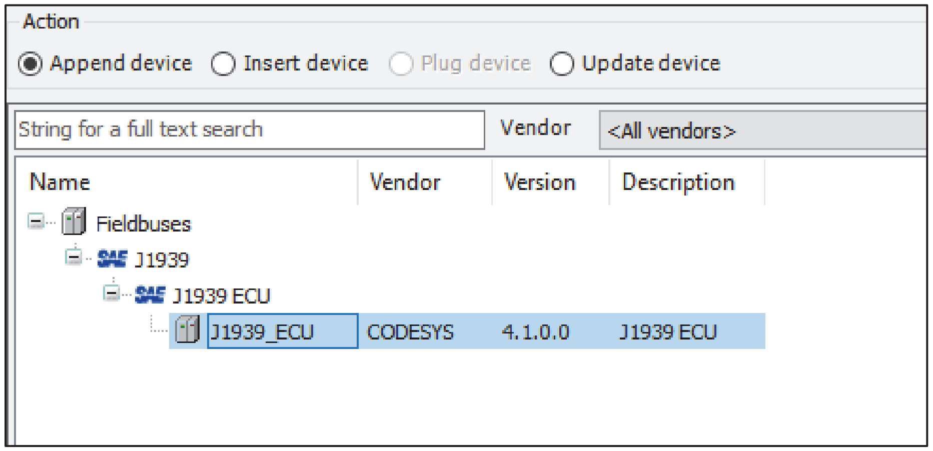
String for a full text search Vendor <All vendors>

Name	Vendor	Version	Description
Fieldbuses			
CANopen			
J1939			
J1939 Manager			
J1939_Manager	CODESYS	4.1.0.0	J1939 Manager

5. It is at the J1939 Manager that you can import DBC files to be used later by your J1939 devices. Under the General tab you can Install DBC files as databases to be used later.



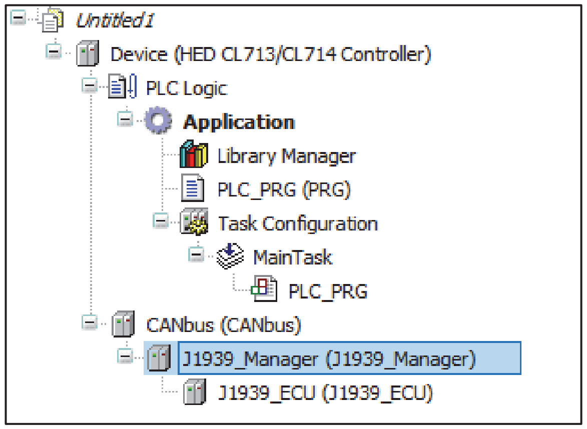
You are now ready to add a J1939 ECU device – you need to create a separate device for each source address your application will be sending or receiving messages.


Action

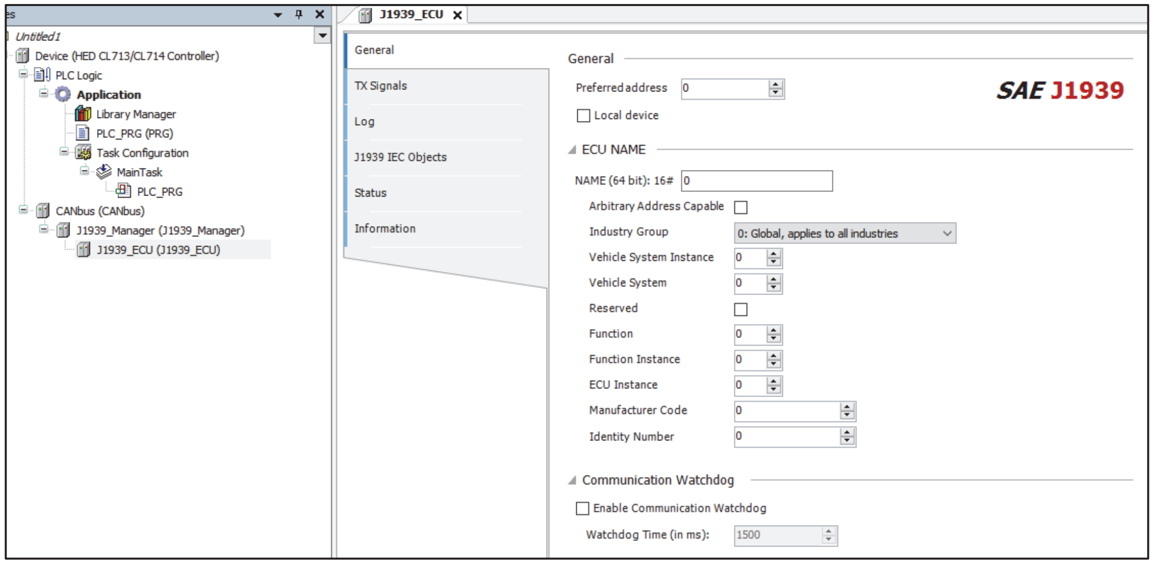
[✓ Append device] Insert device Plug device Update device

String for a full text search Vendor <All vendors>

Name	Vendor	Version	Description
Fieldbuses			
J1939			
J1939 ECU			
J1939_ECU	CODESYS	4.1.0.0	J1939 ECU


```
Untitled1
  Device (HED CL713/CL714 Controller)
    PLC Logic
      Application
        Library Manager
        PLC_PRG (PRG)
        Task Configuration
          MainTask
            PLC_PRG
    CANbus (CANbus)
      J1939_Manager (J1939_Manager)
        J1939_ECU (J1939_ECU)
```

To begin adding messages and/or signals you can double click on the J1939_ECU device to bring up a GUI that will allow you to begin configuring that CAN device.



General

Preferred address: 0

Local device: ☐

ECU NAME

NAME (64 bit): 16# 0

Arbitrary Address Capable	☐
Industry Group	0: Global, applies to all industries
Vehicle System Instance	0
Vehicle System	0
Reserved	☐
Function	0
Function Instance	0
ECU Instance	0
Manufacturer Code	0
Identity Number	0

Communication Watchdog

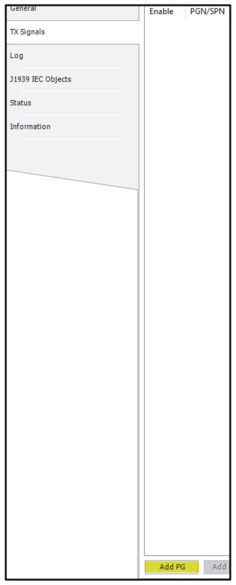
Enable Communication Watchdog: ☐

Watchdog Time (in ms): 1500

Preferred Address is the Source Address of this device, i.e. most engines would be 0 and some transmissions would be 3 or 11 etc.

Please note: As this is an external device (Local Device is not checked) the messages defined under Tx Signals are the messages that will be Received by the CL-713. If you want to use the CAN Device to send messages, you need to check Local Devices.

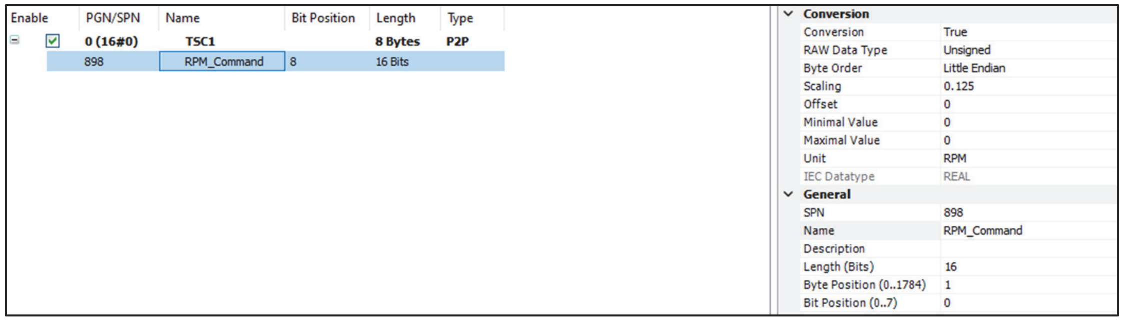
To add a message that would be transmitted by that remoted device to the display you can go into TX Signals and click Add PG



By default, there is no database loaded but if you had a database of messages you could choose directly from that, otherwise Custom needs to be chosen. Under Custom you can define the parameters of the message with a restriction that the PGN (PDU Specific and PDU Format) must remain within the standard J1939 defined space – you cannot define a proprietary message for the device configured as is. To define a proprietary message, you would need to set the device to a Local Device.

After adding a PGN you can then begin adding signals to that PGN, defining how long the signal will be, scaling, offset, name, byte order, etc. Once the signal is added you can then adjust the bit position in which it will start from.

- Length is the length of this data value in bits
- Byte Position is the starting bit within the 8 data bytes of the CAN message
- Bit Position is the starting bit
- Scaling is a multiplier to the value extracted from the CAN message.
- Offest is added to the value extracted from the CAN message



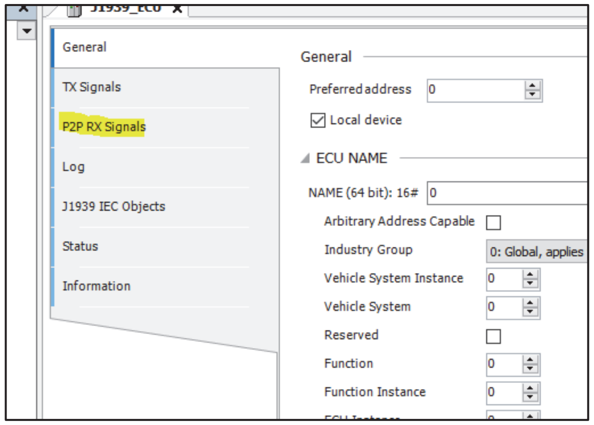
Enable	PGN/SPN	Name	Bit Position	Length	Type
⊟ ✓	**0 (16#0)**	**TSC1**		**8 Bytes**	**P2P**
	898	RPM_Command	8	16 Bits	

Conversion	
Conversion	True
RAW Data Type	Unsigned
Byte Order	Little Endian
Scaling	0.125
Offset	0
Minimal Value	0
Maximal Value	0
Unit	RPM
IEC Datatype	REAL
General	
SPN	898
Name	RPM_Command
Description	
Length (Bits)	16
Byte Position (0..1784)	1
Bit Position (0..7)	0

To utilize the data you are extracting from CAN messages, simply use the Signal name in your application. This line sets a local variable RPM_Command to the Engine_Speed read from a CAN message and adds 50 to that value.

```
RPM_Command := Engine_Speed + 50;
```

Choosing to click the box for Local Device this would designate the ECU device to be a simulated CAN device coming from the controller/display itself. You will see that a new tab appears, P2P Rx Signals



General

Preferred address: 0

☑ Local device

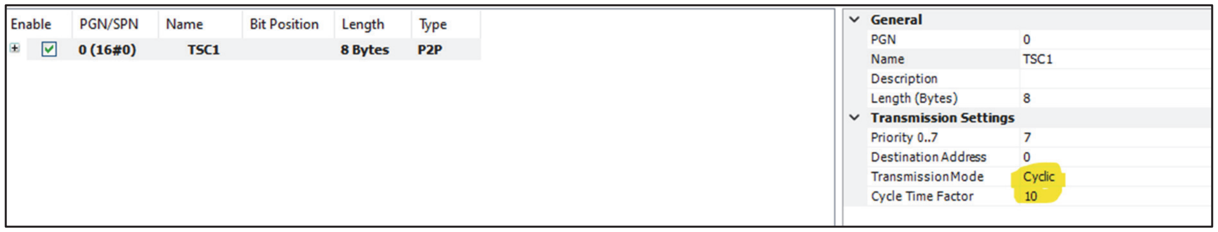
ECU NAME

NAME (64 bit): 16# 0

Arbitrary Address Capable	☐
Industry Group	0: Global, applies
Vehicle System Instance	0
Vehicle System	0
Reserved	☐
Function	0
Function Instance	0

When set as a local device the Tx Signals are the defined messages that the display will transmit out and the P2P RX Signals are the messages it would expect to receive. The Preferred Address in regard to Tx Signals is the Source Address that the message will be transmitted with, but in regard to P2P RX Signals it would be the Destination Address of the message ID. Adjusting the P2P RX Signal source address will be done within that tab directly under the message settings of the PG you have defined.

To define a message to be transmitted from the display go under Tx Signals, like before, and click the Add PG and then define your message under the Custom tab. With the device now selected as local, you will be able to define a PGN within the proprietary realm of Identifiers. For these transmitted messages, selected the PGN allows you to configure how the message is sent: On Change, Cyclic, etc.



Enable	PGN/SPN	Name	Bit Position	Length	Type
✓	**0 (16#0)**	**TSC1**		**8 Bytes**	**P2P**

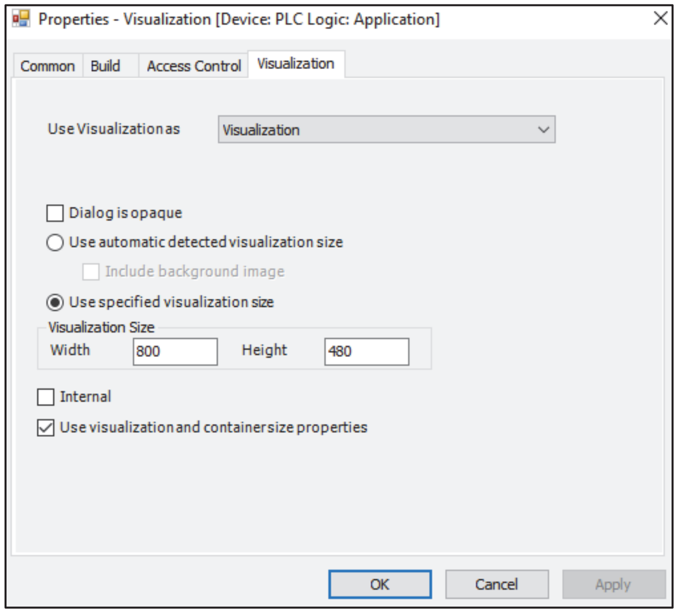
General	
PGN	0
Name	TSC1
Description	
Length (Bytes)	8
Transmission Settings	
Priority 0..7	7
Destination Address	0
TransmissionMode	Cyclic
Cycle Time Factor	10

To define a message to be received by the display, you would click the P2P RX Signals and add a PG there. Or you can add a new device to receive those messages.

CREATING SCREENS

Screens are added through the use of visualizations within CODESYS.

1. Right click on Application and then Add Object->Visualization
2. It is recommended after the visualization is created to right click on it and adjust the properties such that it has a fixed size compatible with the display screen size of 800x480 pixels.



Properties - Visualization [Device: PLC Logic: Application]

Common | Build | Access Control | Visualization

Use Visualization as: Visualization

☐ Dialog is opaque
○ Use automatic detected visualization size
☐ Include background image
● Use specified visualization size

Visualization Size
Width: 800
Height: 480

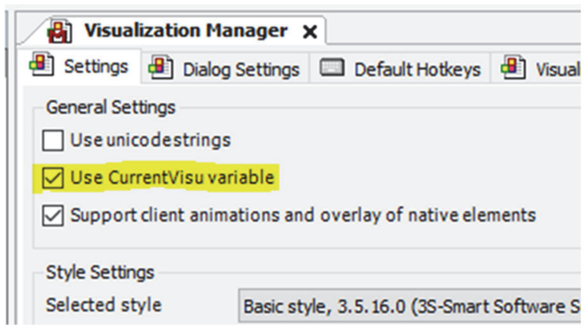
☐ Internal
☑ Use visualization and container size properties

3. From here create your HMI as you normally would within CODESYS

SWITCHING BETWEEN SCREENS IN YOUR APPLICATION

If your application has more than one screen, you will need a way to change which screen is being shown on the display. There are many methods to do this, and most can be found by doing a quick search on the internet or using one of the many CODESYS examples, but here is one of those methods.

1. Open the Visualization Manager to find the “Use CurrentVisu variable” and ensure that is checked



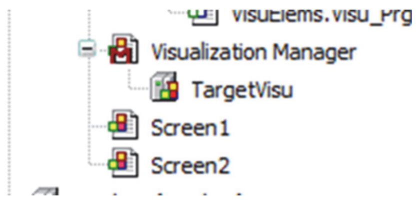
General Settings

- [ ] Use unicodestrings
- [x] Use CurrentVisu variable
- [x] Support client animations and overlay of native elements

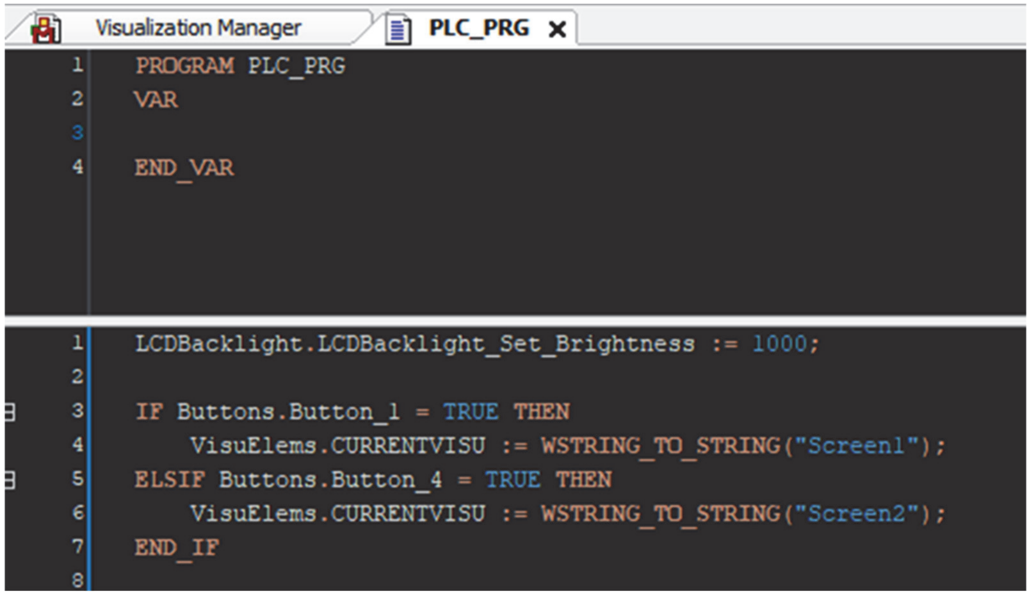
Style Settings

Selected style: Basic style, 3.5.16.0 (3S-Smart Software S

2. Create your separate visualizations



3. To change what visualization will be visible you will need to make use of that CurrentVisu variable by setting it equal to the visualization you wish to show.



```
PROGRAM PLC_PRG
VAR

END_VAR
```

```
LCDBacklight.LCDBacklight_Set_Brightness := 1000;

IF Buttons.Button_1 = TRUE THEN
    VisuElems.CURRENTVISU := WSTRING_TO_STRING("Screen1");
ELSIF Buttons.Button_4 = TRUE THEN
    VisuElems.CURRENTVISU := WSTRING_TO_STRING("Screen2");
END_IF

```

This small example is making use of the buttons to change to a specific screen which sets the CurrentVisu variable equal to the name of the visualization you wish to have shown.

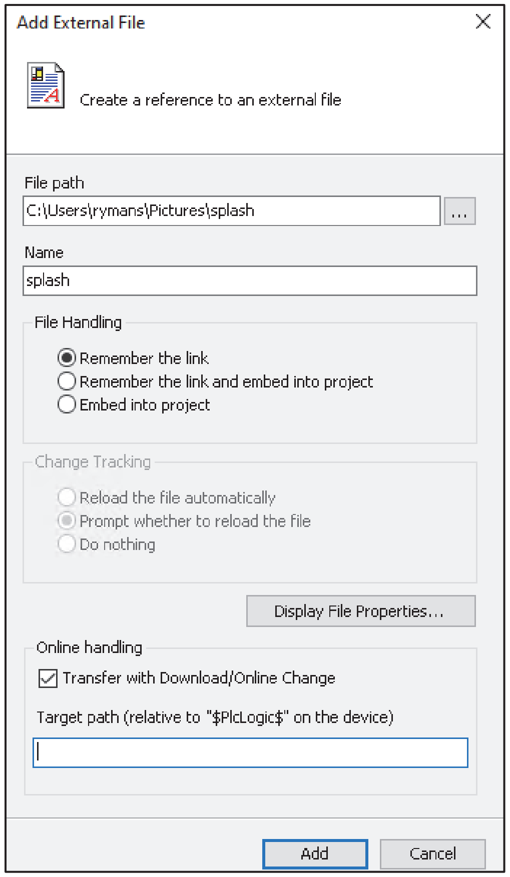
CHANGING OF SPLASH SCREEN

The splash file is a special file that represents the image used for the boot time splash screen. This file cannot be made within CODESYS, but it can be transferred/downloaded from within the CODESYS environment.

To do this you will need to create an image outside of the CODESYS environment following a few simple rules:

1. The image type is one of: PNG, BMP, JPG, or GIF
2. The file name is named splash with no file extension.
3. The resolution of the file must be exactly 800x480.

Once the file is made you will be able to add it to your project by right clicking on Application -> Add Object -> External File which will bring up a window for you to navigate to the file and select an option:


Add External File

Create a reference to an external file

File path: C:\Users\rymans\Pictures\splash

Name: splash

File Handling
- [✓ Remember the link]
- Remember the link and embed into project
- Embed into project

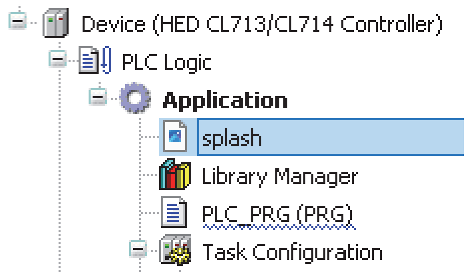
Change Tracking
- Reload the file automatically
- [✓ Prompt whether to reload the file]
- Do nothing

Online handling
- [✓ Transfer with Download/Online Change]

Target path (relative to "$PlcLogic$" on the device):

Please ensure to keep the Target Path empty, there are internal scripts within the display that will properly place the file on the target. You may select “Embed into project” option if you wish to keep the image saved/contained within the project itself, though it is not required.

Once added to the project you should be able to see the splash within your Application tree structure:

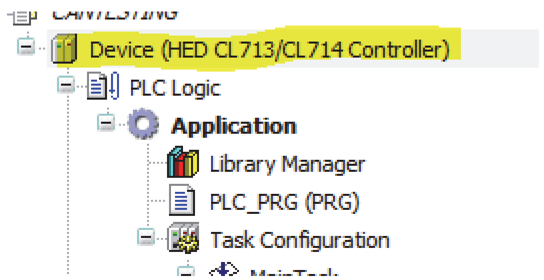


After the image is added to the application you may login to your device and download as you normally would and application (following section below). That will get the file onto the device, but the splash will not be immediately available. The device will need to run some internal scripts on a reboot which will relocate the needed file and force another reboot of the device. Upon that second reboot, your splash image should appear as the splash going forward.

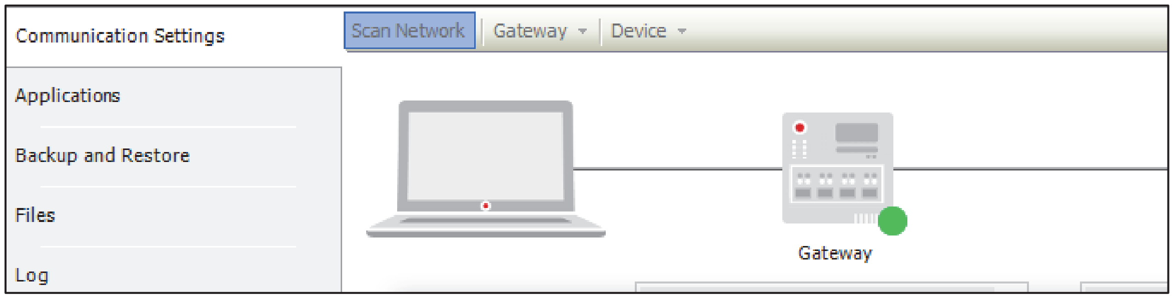
PROGRAMMING DISPLAY THROUGH CODESYS

By default, your display should come with the CODESYS runtime installed on it from the factory. To download your CODESYS application to it you will need the M12 USB cable connected to the USB port on your display, and you will need to have the proper drivers installed in order for the device to be found/recognized. To obtain the drivers you will need to download HED’s Linux Device Downloader (LDD). The tool, while not directly used with a CODESYS display, will install all the necessary drivers for interfacing with the display. You can contact HED’s Technical Support (technicalsupport@hedcontrols.com) to obtain the download if needed.

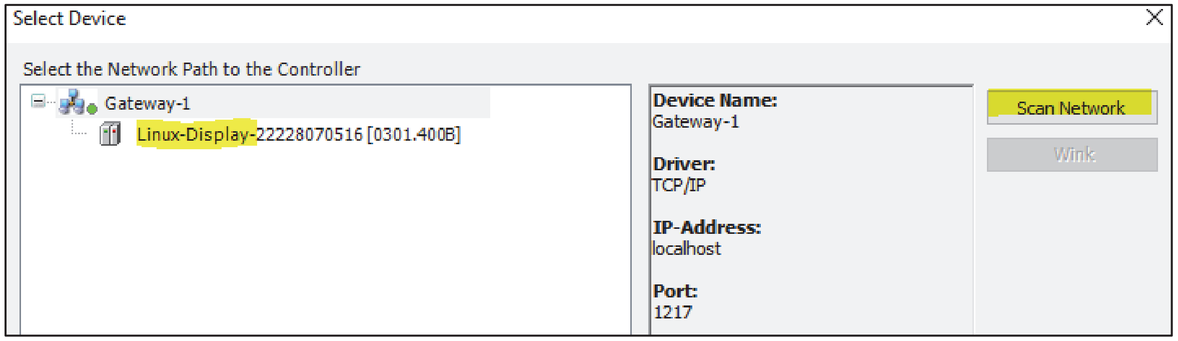
1. Power the display up and then plug the USB into your PC. From there you can double-click on the top level device in your project.



2. Next, under Communication Settings click Scan Network.



3. Once the scanning window comes up you can click the scan network and look for the Linux-Display.


Select Device
Select the Network Path to the Controller
Gateway-1
Linux-Display-22228070516 [0301.400B]
Scan Network
Wink

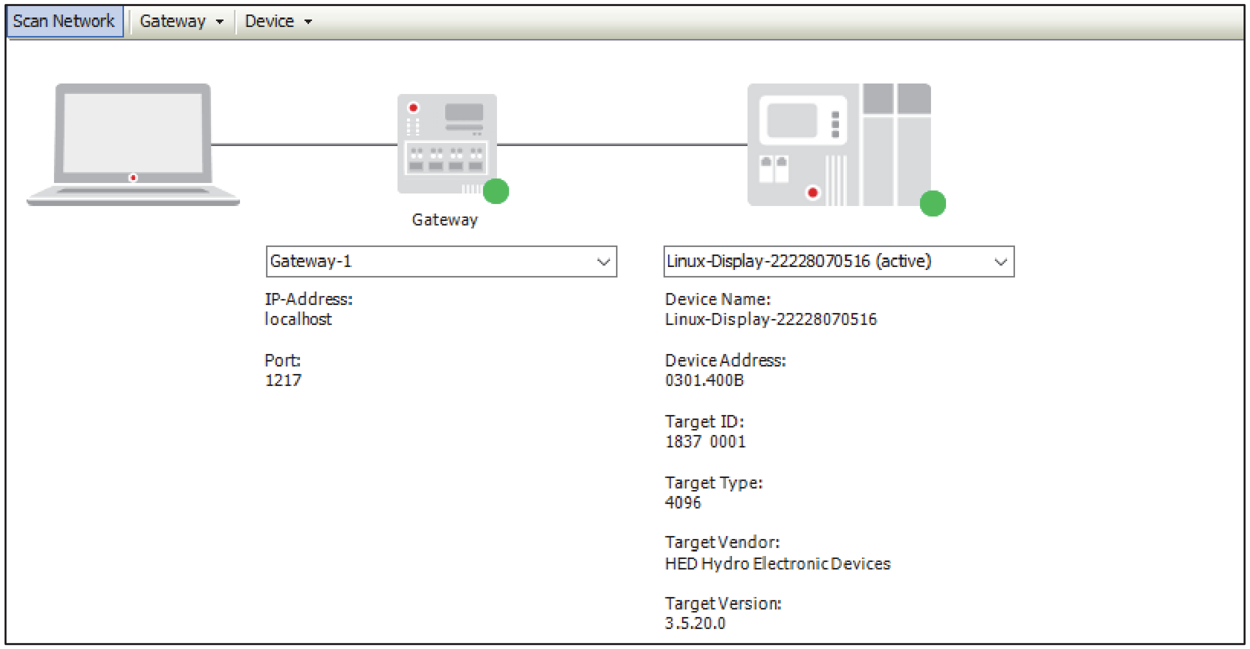
Device Name:
Gateway-1

Driver:
TCP/IP

IP-Address:
localhost

Port:
1217

4. Double-click on the display, you may be asked to create a user management profile for that display if you have not done so already, if you have then proceed logging into the user account. After logging in your target should show as the Linux Device.



Gateway-1

IP-Address:
localhost

Port:
1217

Linux-Display-22228070516 (active)

Device Name:
Linux-Display-22228070516

Device Address:
0301.400B

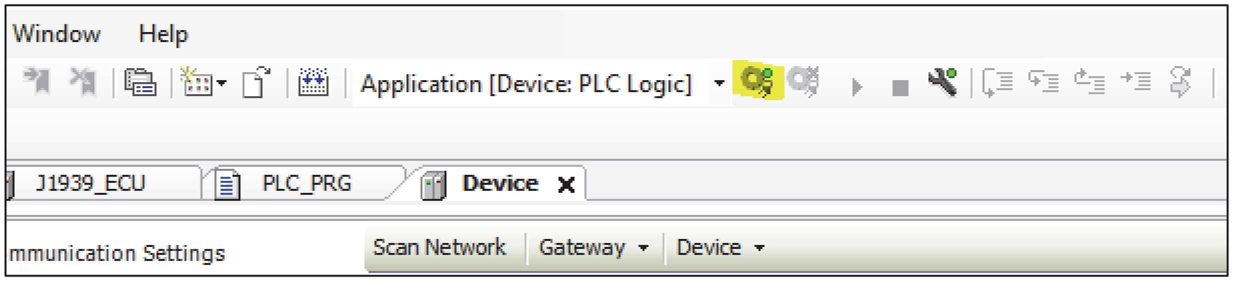
Target ID:
1837 0001

Target Type:
4096

Target Vendor:
HED Hydro Electronic Devices

Target Version:
3.5.20.0

5. Once the target is set you can then load your application by selecting the Online->Login, Alt+F8, or clicking the log in icon at the top. This will prompt the download of your application.



PRE-PROGRAMMING OPTION

We offer a pre-programming option where you provide HED the necessary files and we place your application on the display(s) in production so they ship ready to go from the factory. To take advantage of this, please work with your HED Sales Representative.

The file(s) we would need would be the .projecarchive file. To create a project archive: in your project, select File -> Project Archive -> Save Archive. Once the file is created send that to us along with any permissions information we may need, pending your permissions and user management within the project, to open the project to download to a display.