neoECU-12

User's Guide

Version 1.0 – March 17, 2020

TABLE OF CONTENTS

1 Introduction and Overview
- 1.1 Introduction
- 1.2 Package Contents
- 1.3 Operational Overview
- 1.4 Block Diagram
- 1.5 Summary of Key Features
- 1.6 Hardware and Software Requirements

2 A Tour of neoECU-12 Hardware
- 2.1 Case and Overall Design
- 2.2 Front side of case
- 2.3 Back side of case

3 Hardware and Software Setup
- 3.1 Vehicle Spy and Driver Installation and Setup
- 3.2 Hardware Connections
- 3.3 Termination
- 3.4 Vehicle Network and Power Connections
- 3.5 PC Connection

4 Device Configuration
- 4.1 Starting and Using neoVI Explorer
- 4.2 Update Firmware
- 4.3 Hardware Setup
- 4.4 CAN Setup
- 4.5 LIN Setup
- 4.6 MISC IO Setup
- 4.7 Program CoreMini

5 Function Block Scripts
- 5.1 Setting MISC IO as inputs
- 5.2 Send MISC IO status in CAN message
- 5.3 Setting MISC IO as outputs
- 5.4 Setting the MISC IO as PWM outputs
- 5.5 Setting the Analog Inputs
- 5.6 Send Analog Input value in CAN message
- 5.7 Controlling the Tri-Color LEDs
- 5.8 LIN example
- 5.9 CAN to CAN-FD Gateway example

6 Support Contact Information
- 6.1 ICS United States Headquarters
- 6.2 ICS International Offices

1 Introduction and Overview

1.1 Introduction

Thank you for purchasing an Intrepid Control Systems neoECU-12 low-cost embedded ECU. The neoECU-12 is a scriptable, low cost node for CAN/CAN-FD, and LIN. The neoECU-12 provides two programmable CAN/CAN-FD channels. The CAN/CAN-FD channels can be selected for SW CAN, or LSFT CAN, or CAN/CAN-FD. The neoECU-12 also provides one channel of LIN, seven channels of MISC I/O (six at 3.3V, one at 5.0V), four channels of Analog Inputs, and five channels of PWM. The five channels of PWM are multiplexed with the first five channels of MISC I/O.

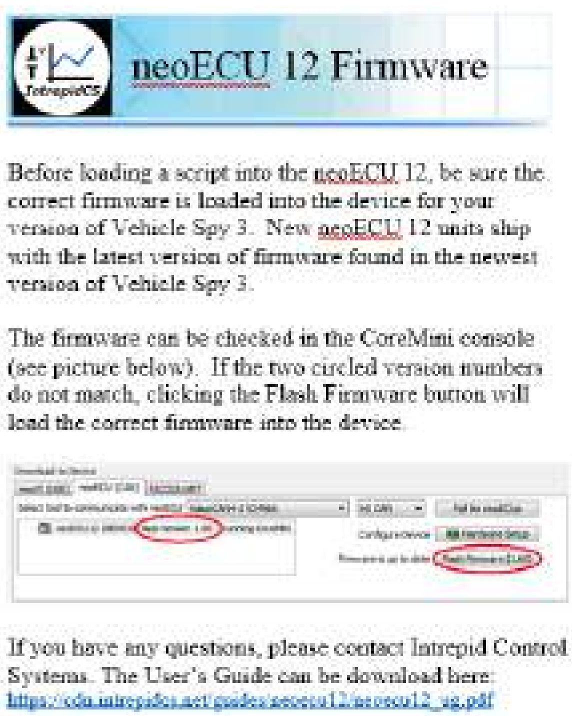
The neoECU-12 is the next generation of the neoECU-10 and adds CAN-FD capability.

1.2 Package Contents

Your neoECU-12 package includes both hardware and software.

Hardware

Upon opening the neoECU-12 box, you should find a neoECU-12 Firmware Guide on top with the device itself secured under a transparent plastic film in a cardboard holder. Remove the guide, the device and the cardboard, and you'll see the following items:


neoECU 12 Firmware

Before loading a script into the neoECU 12, be sure the correct firmware is loaded into the device for your version of Vehicle Spy 3. New neoECU 12 units ship with the latest version of firmware found in the newest version of Vehicle Spy 3.

The firmware can be checked in the CoreMini console (see picture below). If the two circled version numbers do not match, clicking the Flash Firmware button will load the correct firmware into the device.

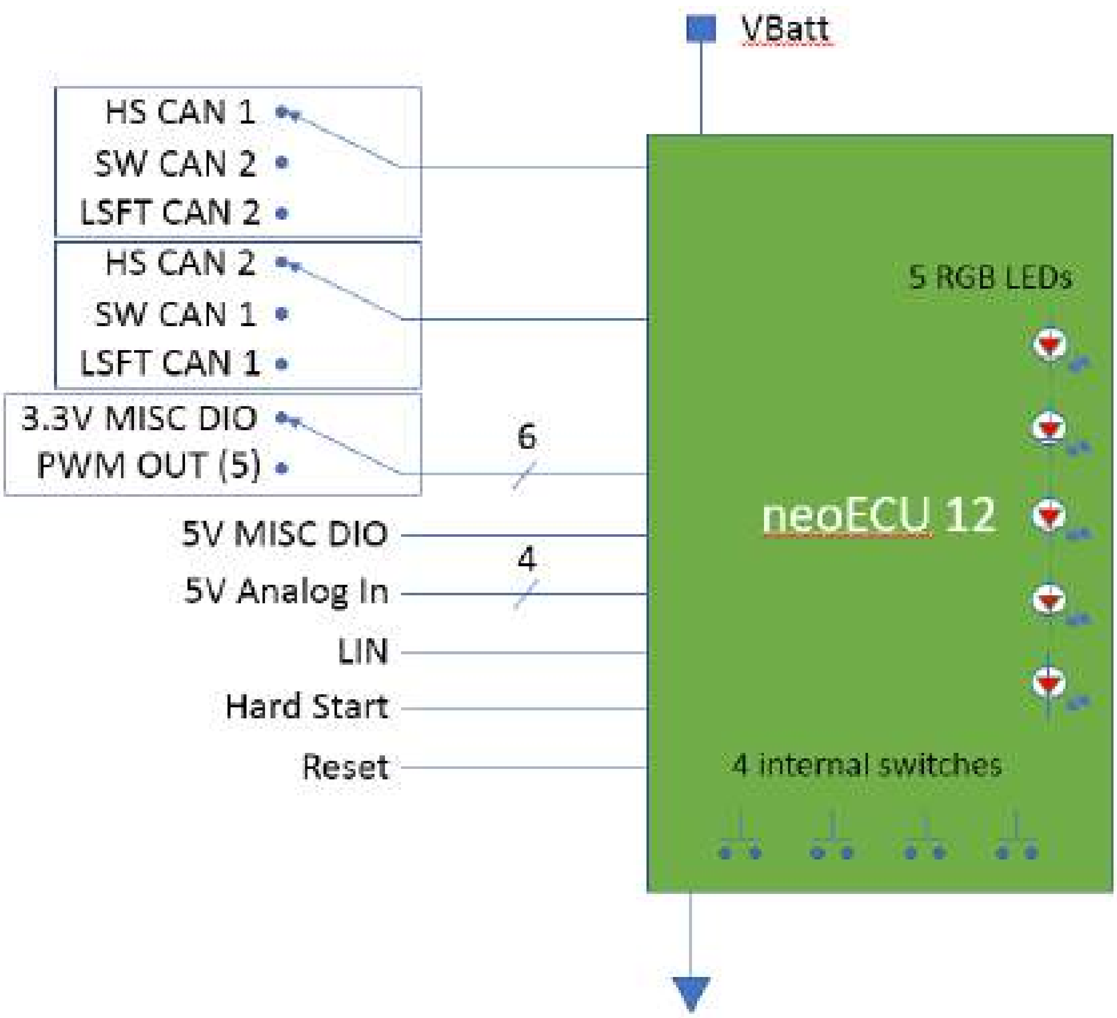
If you have any questions, please contact Intrepid Control Systems. The User's Guide can be download here: https://cdn.intrepidcs.net/guides/neoecu12/neoecu12_ug.pdf

1.3 Operational Overview

The neoECU-12 is a compact but powerful tool for working with vehicle networks. Its operation can be broken into multiple network communication, analog and digital inputs, and PWM and digital outputs. This device is ideal for CAN and/or LIN node communication and provides numerous connection points to interface to external circuits for command and control.

Using Vehicle Spy software you can define CAN and LIN transmit messages with custom data and send them on a periodic schedule or send them based on certain conditions. Using Intrepid's Scripting language built into Vehicle Spy you can create custom intelligent scripts that control the input and output ports. The data from the input and output ports can be inserted as signals into the CAN or LIN messages or used to trigger CAN or LIN messages. The custom scripts will be compiled and programmed into the internal memory (called CoreMini) of the neoECU-12.

1.4 Block Diagram



Please note HSCAN 1 H, LSFT CAN 2 H, and SW CAN 2 share the same pin (14) so only one can be selected. HSCAN 1 L and LSFT CAN 2 L share the same pin (15). You select which network you want to use in neoVI Explorer.

To force the neoECU-12 into bootloader mode put a "low" onto the Hard Start pin (before you power-up the neoECU-12). Otherwise leave it open since the neoECU-12 has an internal pull-up resistor keeping the pin "high".

To force the neoECU-12 into reset mode put a "low" onto the Reset pin. Otherwise leave it open since the neoECU-12 has an internal pull-up resistor keeping the pin "high".

1.5 Summary of Key Features

The neoECU-12 is a low-cost embedded ECU with CAN FD and LIN. It has two channels of CAN/CAN FD which can also be configured for SW CAN and LSFT CAN. It has digital inputs and outputs, analog inputs, and PWM outputs. There are four internal programmable buttons and five internal tri-color LEDs that are visible on the back side of the unit.

Construction

- Compact design: 2.2" x 3.7" x 1.0" (5.6 X 9.4 x 2.5 cm)
- Light weight: less than 2 oz. (57 g)
- Plastic casing
- 25-pin male DB 25 connector

Power and Performance

- Fourth generation neoVI architecture
- 32 MB memory for custom scripts
- Field upgradeable firmware
- 4.5V to 40V input power on VBatt
- Lower power sleep mode

Network Interfaces and Features

- Two selectable CAN channels: CAN/CAN-FD or SW CAN or LSFT CAN
  - Software-configurable CAN termination
  - Software enable/disable, baud rate and other parameters
- 1 LIN / K-Line channel
  - Software enable/disable, baud rate and other parameters
- Six selectable MISC DIO or PWM output channels. NOTE: PWM 3 waveform is mirrored on PWM 6. Frequency range is 5 Hz to 20 KHz. Maximum frequency is 100 KHz when using 50% duty cycle.
- Four Analog inputs. 12-bit A/D resolution (0-4095 counts), 5V maximum input voltage.
- Hard start pin. If a "low" voltage is detected on this pin then the neoECU-12 starts in bootloader mode.
- Reset pin. If a low voltage is inserted on this pin then the neoECU-12 is reset.

1.6 Hardware and Software Requirements

- A vehicle network, either within an actual vehicle or a test bench environment.
- A DC power supply capable of providing 4.5V (minimum) to 40V (maximum), with a nominal current of 70 mA at 12V. Your network setup must include wiring capable of providing this power on pin 25 and ground on pin 13.
- In order to program the neoECU-12 you will need a ValueCAN or neoVI device connected to HS CAN 1 and a licensed copy of Vehicle Spy (Pro or Enterprise version) 3.9.1.13 or higher running on a Windows-based PC or laptop.

2 A Tour of neoECU-12 Hardware

2.1 Case and Overall Design

The neoECU-12 is enclosed in a lightweight plastic case. The device has been designed and tested for in-vehicle use, and is operational in a temperature range from -40°C to +85°C.

The pinout for the DB25 male connector is on the bottom of the case.

DB 25 Pin Out

Pin	Signal	Pin	Signal
1	SW CAN 1	14	HS CAN 1 H LSFT CAN 2 H SW CAN 2
2	HARD START	15	HS CAN 1 L LSFT CAN 2 L
3	LSFT CAN 1 H		
4	LSFT CAN 1 L	16	HS CAN 2 H
5	MISC IO 1	17	HS CAN 2 L
6	MISC IO 2	18	MISC IO 3
7	MISC IO 7	19	MISC IO 4
8	LIN 1	20	MISC IO 5
9	-	21	MISC IO 6
10	AIN 1	22	AIN 3
11	AIN 2	23	AIN 4
12	-	24	RESET
13	GND	25	VBATT

2.2 Front side of Case

The front side of the case contains the DB25 male connector.

2.3 Back side of Case

The back side of the case contains the serial number and tri-color LEDs.

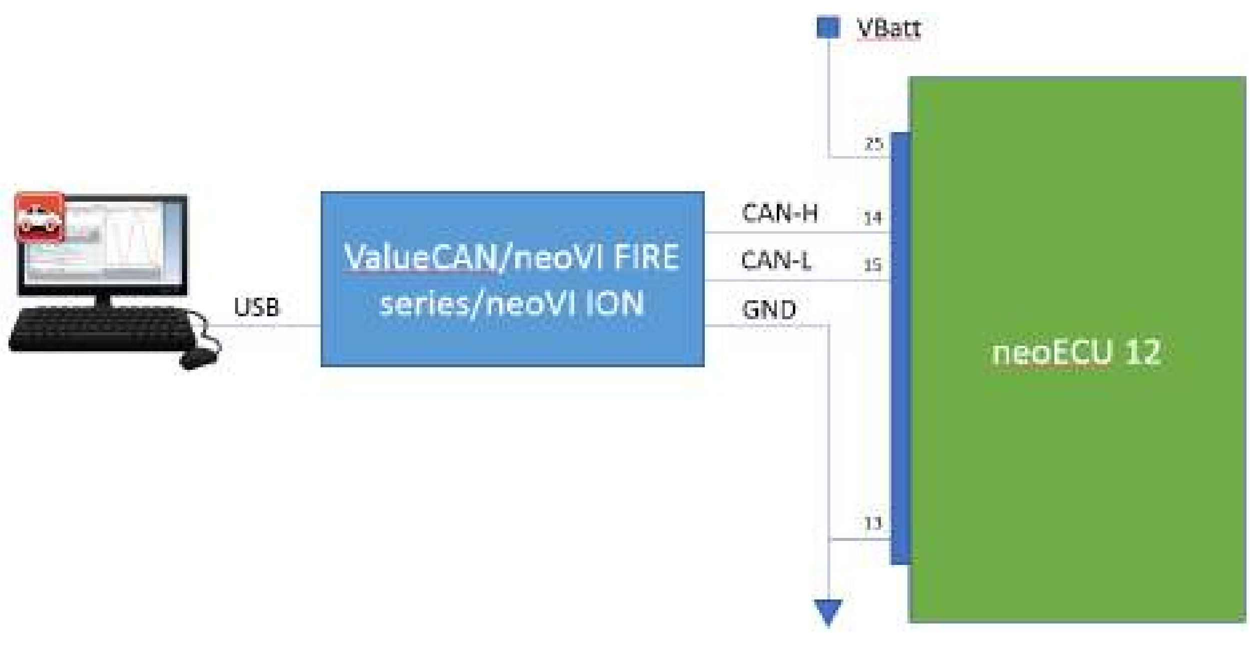
3 Hardware and Software Setup

3.1 Vehicle Spy Installation

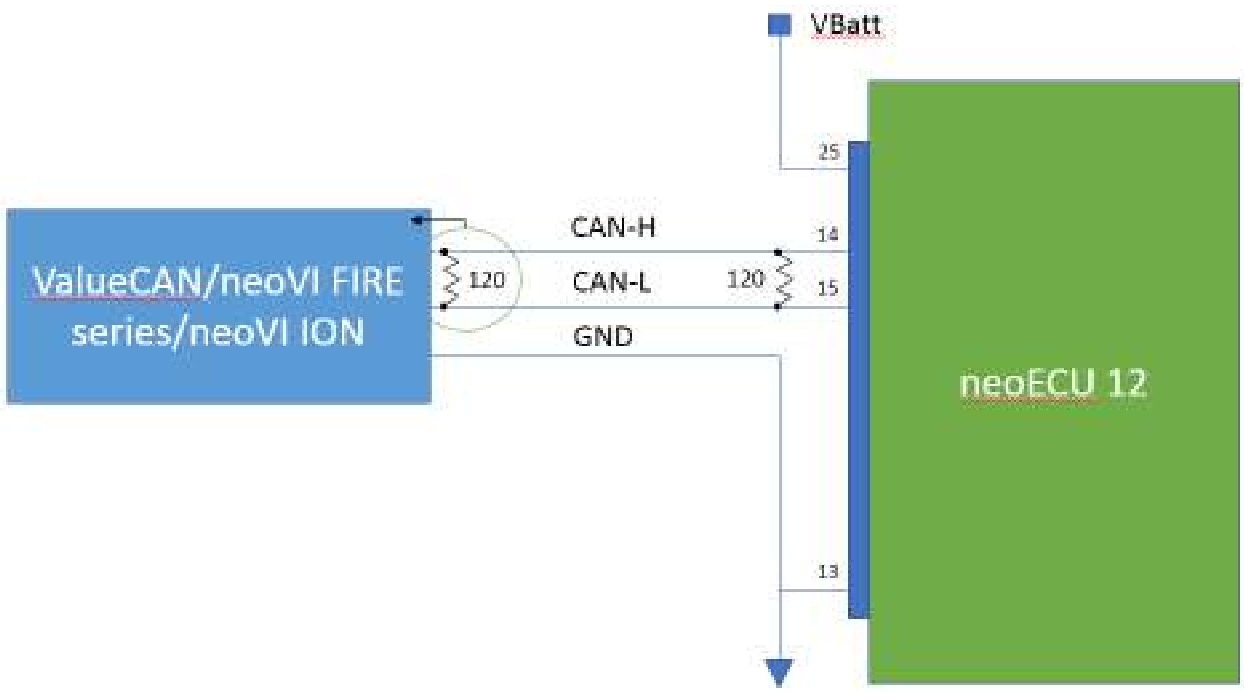
1. Run the Vehicle Spy installer from the CD-ROM or from the download link
2. Select Language
3. Start Vehicle Spy setup wizard
4. Review and accept license agreement
5. Select installation type (new or repair … most cases will be new)
6. Select destination location (we recommend the default location)
7. Select Start Menu folder (we recommend the default location)
8. Review installation options and begin installation
9. Install VCP Drivers
10. Install WinPcap
11. Install ICS port Drivers

3.2 Hardware Connections

The following diagram will show the minimum connection to the neoECU-12 in order to configure it and program the CoreMini of the device. Connect to HS CAN 1 of the neoECU-12.



Termination. Please note the neoECU-12 does not have built-in selectable termination resistors. Therefore, an external resistor (120 ohm) is required on the CAN-H and CAN-L pins as shown below. The device you are connecting to also needs the termination resistor. Most of the Intrepid Control System devices have built-in selectable termination resistors that can be enabled in via the hardware setup for that device. If this option is not available, then add the external resistor as shown below.



4 Device Configuration

4.1 Internal momentary pushbutton switches

In order to open the case use a small screwdriver and put into the slot near the front and twist. Do this for both sides and you will see the case is hinged in the back. Now you have access to the momentary pushbutton switches.

Press and hold one of these switches to get in that particular mode when applying power to the neoECU-12. After power is applied the LEDs will light red in sequence from 1 to 5 and repeat.

Switch	Description
Switch 1	Prevents neoECU from Automatically running the CoreMini Script
Switch 2	Sets application settings to default states
Switch 3	Forces neoECU into Boot-Loader Mode (This can also be done by holding Pin 2 of the 25 pin connector to ground)
Switch 4	Forces neoECU into Boot-Loader mode and sets application settings to default states

S1, S2, S3, S4

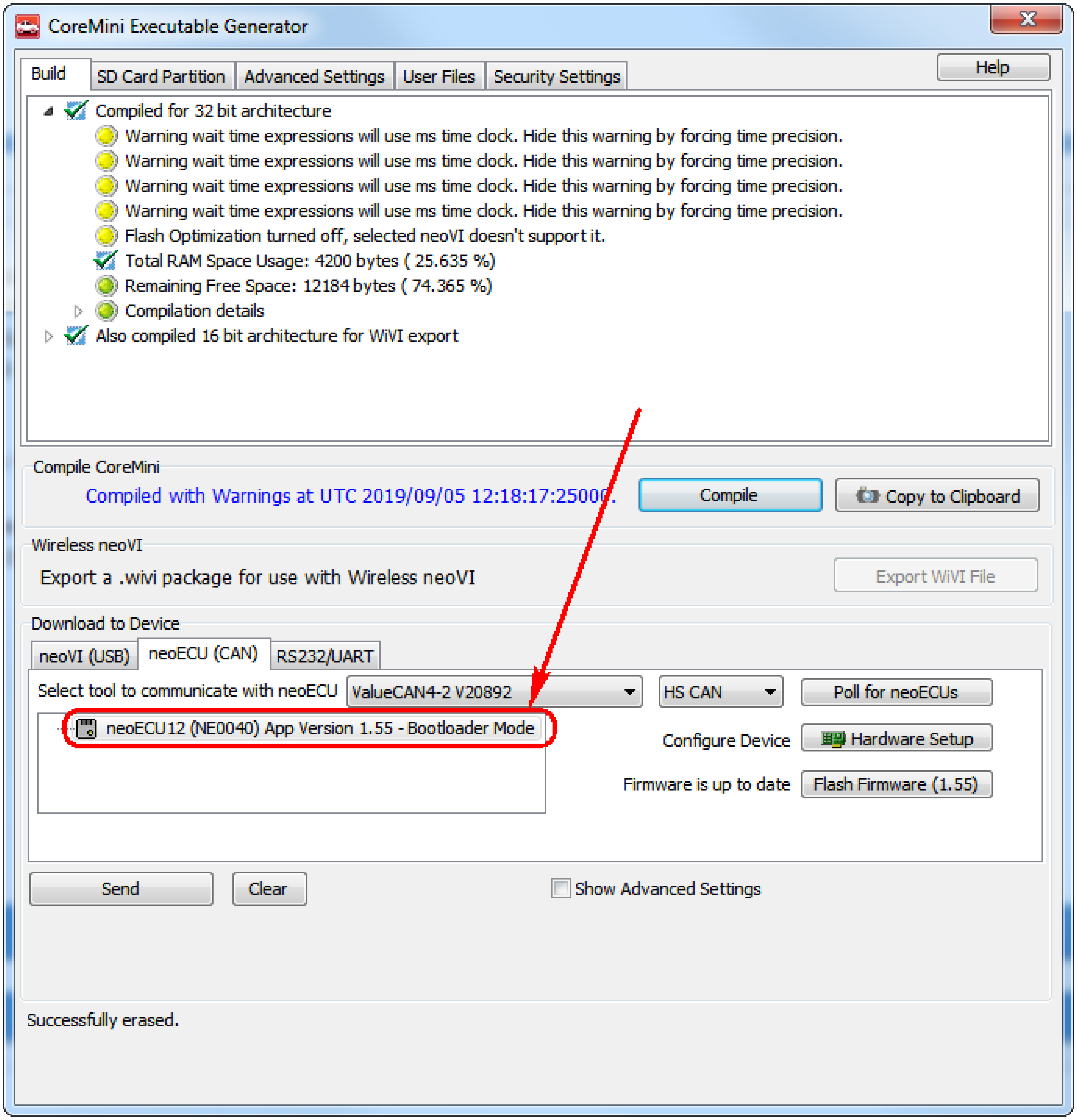
4.2 Update firmware

Open Vehicle Spy 3.

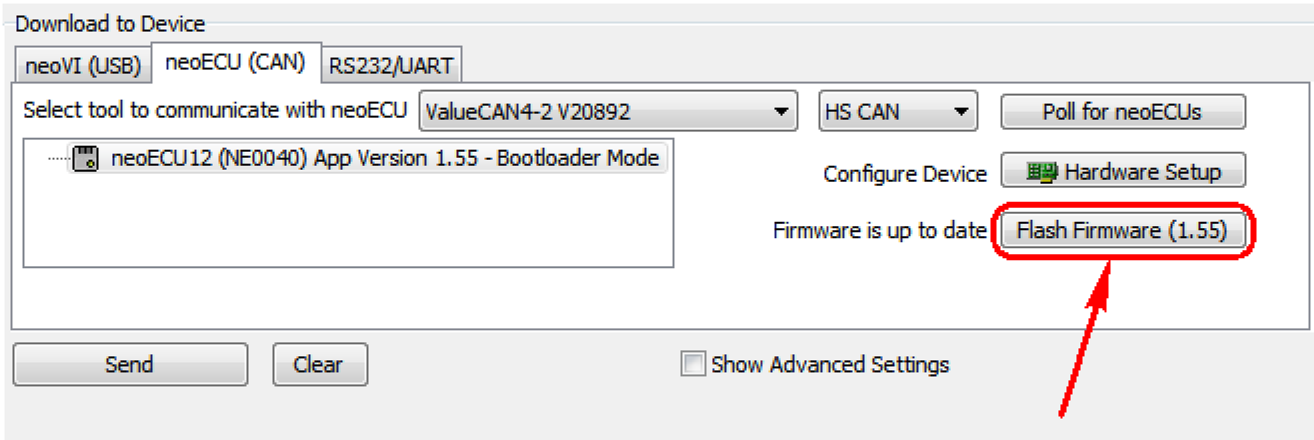
You will see the version number in the lower left corner (i.e. 3.9.1.15) and the level in red text (i.e. Basic, Pro, or Enterprise). This is the device connected to the computer via the USB cable that will program the neoECU-12 over the CAN Bus.

Next, select "CoreMini Console" from the Tools pull-down menu and wait for the CoreMini Console window to open.

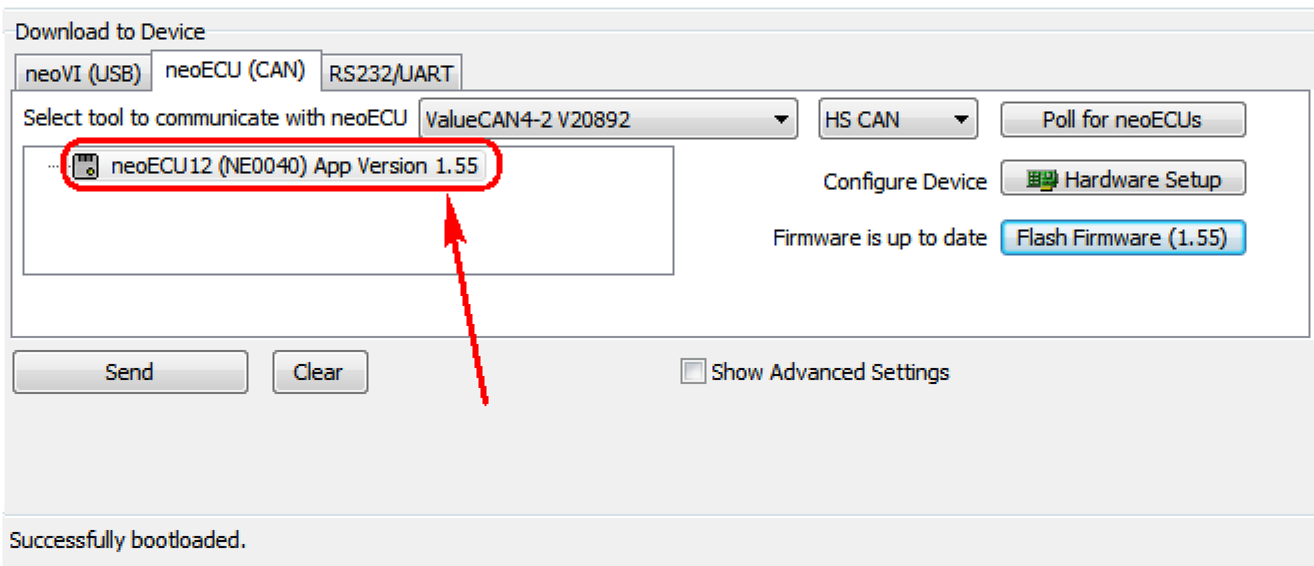
Click on the neoECU CAN tab and note the neoECU-12 is in bootloader mode.


CoreMini Executable Generator
Build
SD Card Partition
Advanced Settings
User Files
Security Settings
Help
Compiled for 32 bit architecture
Warning wait time expressions will use ms time clock. Hide this warning by forcing time precision.
Warning wait time expressions will use ms time clock. Hide this warning by forcing time precision.
Warning wait time expressions will use ms time clock. Hide this warning by forcing time precision.
Warning wait time expressions will use ms time clock. Hide this warning by forcing time precision.
Flash Optimization turned off, selected neoVI doesn't support it.
Total RAM Space Usage: 4200 bytes (25.635 %)
Remaining Free Space: 12184 bytes (74.365 %)
Compilation details
Also compiled 16 bit architecture for WiVI export
Compile CoreMini
Compiled with Warnings at UTC 2019/09/05 12:18:17:25000.
Compile
Copy to Clipboard
Wireless neoVI
Export a .wivi package for use with Wireless neoVI
Export WiVI File
Download to Device
neoVI (USB)
neoECU (CAN)
RS232/UART
Select tool to communicate with neoECU
ValueCAN4-2 V20892
HS CAN
Poll for neoECUs
neoECU12 (NE0040) App Version 1.55 - Bootloader Mode
Configure Device
Hardware Setup
Firmware is up to date
Flash Firmware (1.55)
Send
Clear
Show Advanced Settings
Successfully erased.

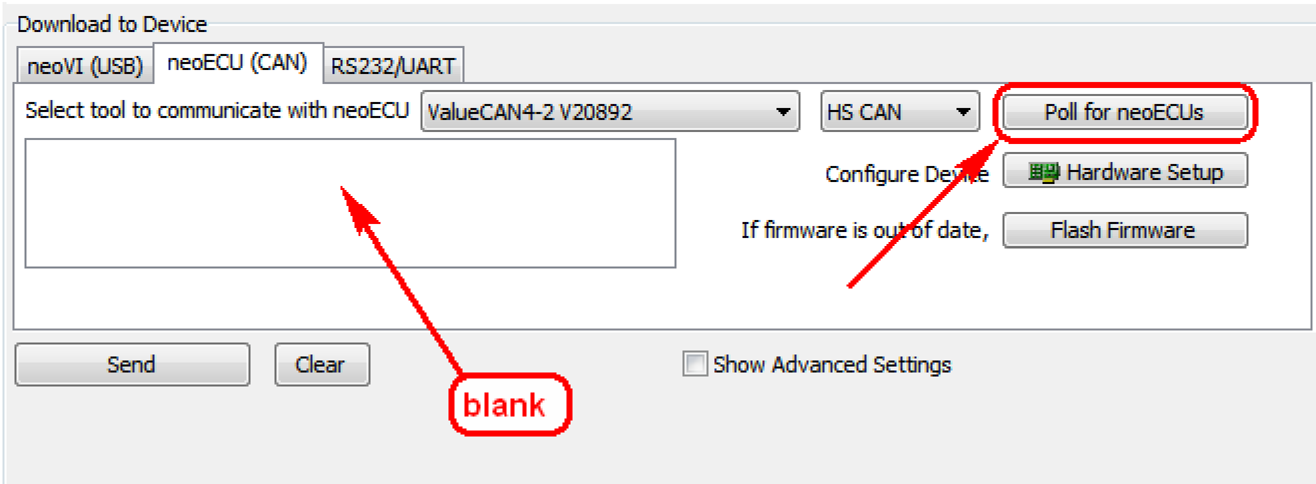
Click on the "Flash Firmware" button to update the firmware and get it out of bootloader mode.



After the device is successfully updated you will see that it no longer shows "Bootloader Mode" and LED 3 is blinking a magenta color. During the flash sequence LED 3 blinks an orange color.

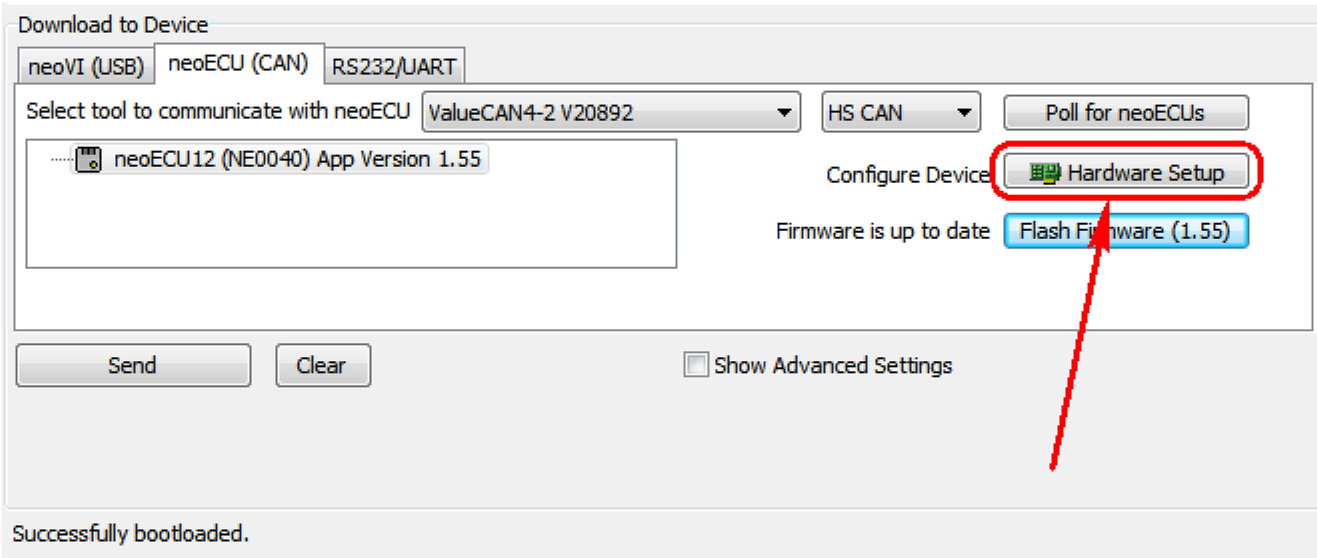


If the neoECU tab is blank as shown below click on the "Poll for neoECUs button. If the neoCU-12 does not show then there is probably a baud rate mismatch.

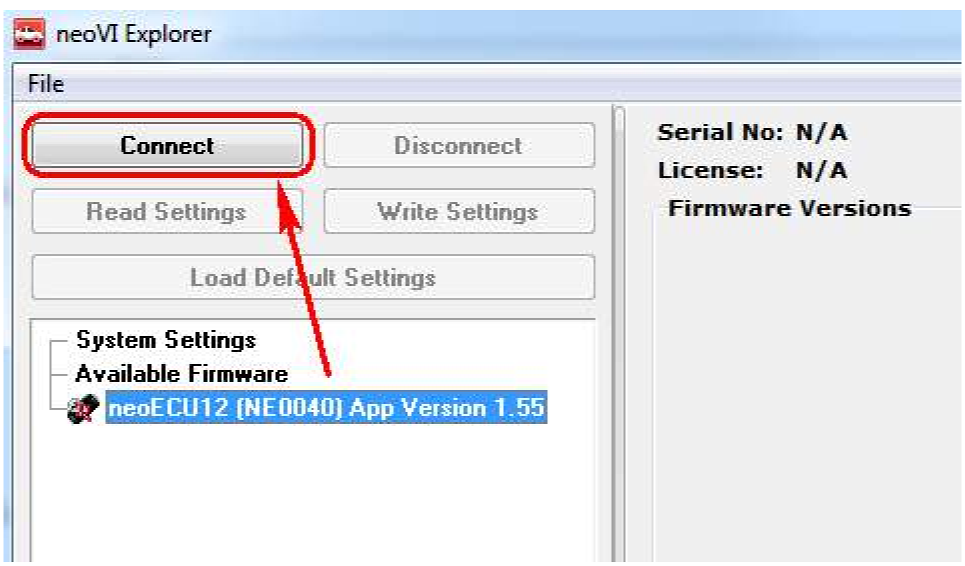


4.3 Hardware Setup

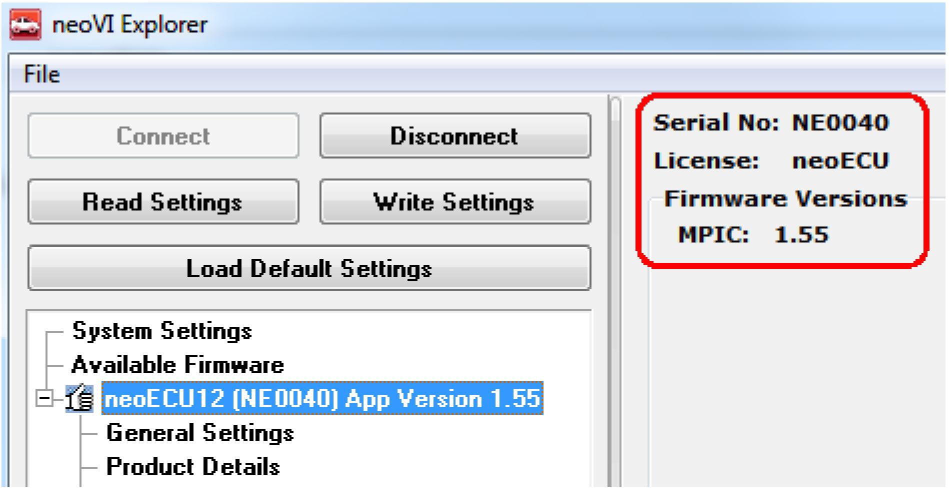
Next, click on the "Hardware Setup" button to set the parameters for the HS CAN 1, HS CAN 2, and LIN Busses and for all the peripherals on the neoECU-12.



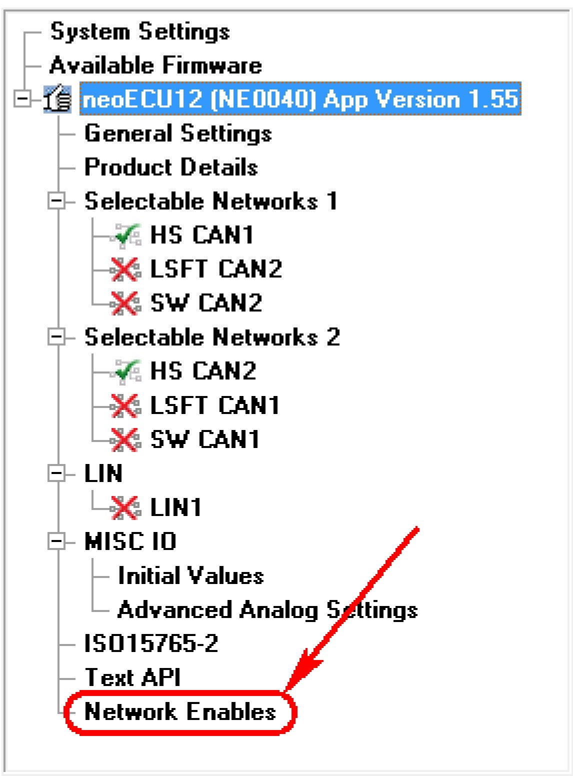
The neoVI Explorer window should pop-up. The neoECU-12 should be listed. If not, click on the "Search for Devices' button at the bottom of this window. Select the neoECU-12 and click the "Connect" button.



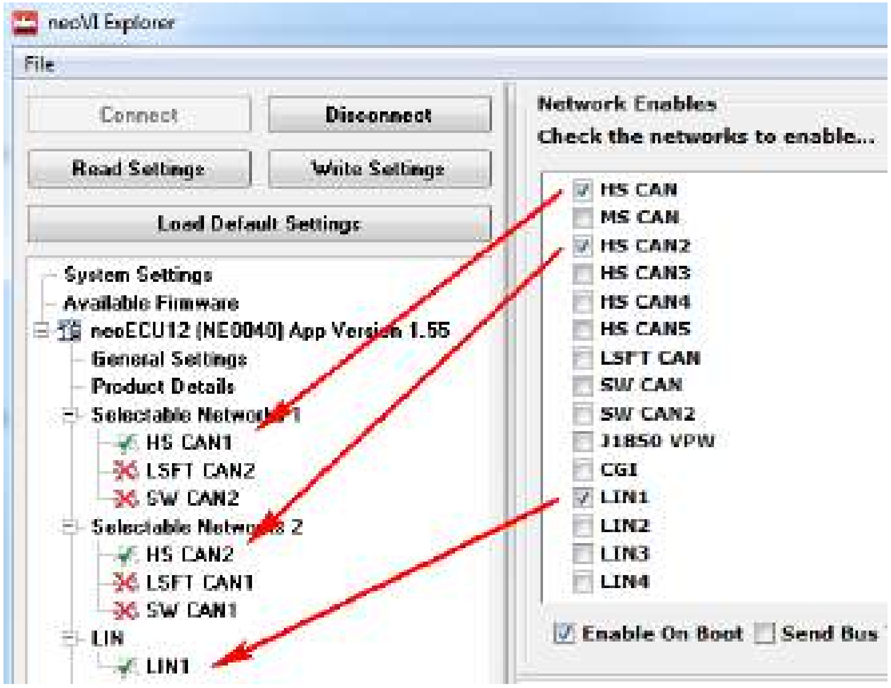
After connecting you will see the neoECU-12 Serial Number and Firmware version.



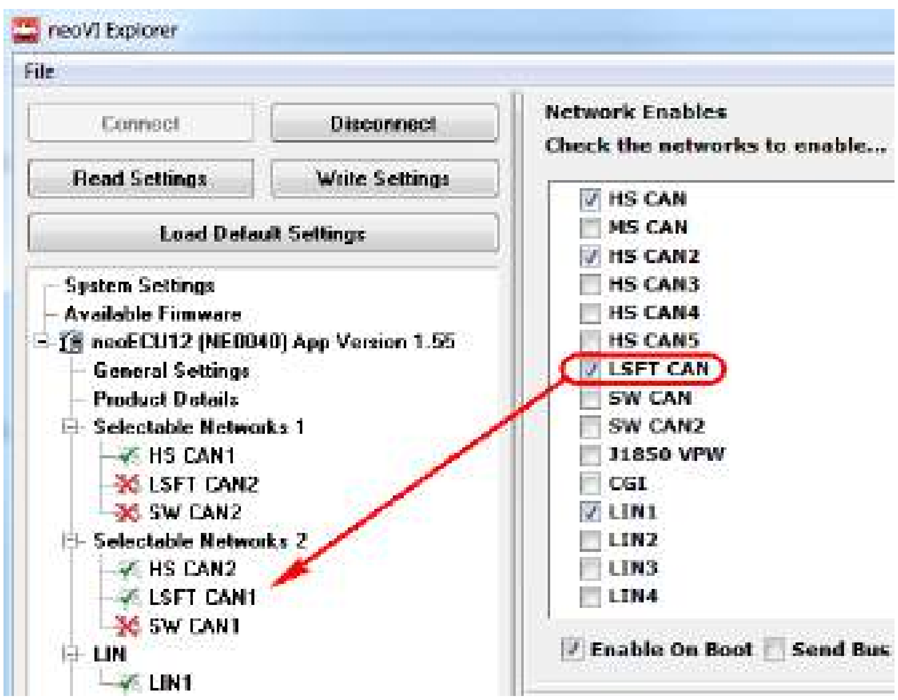
Next click on "Network Enables".



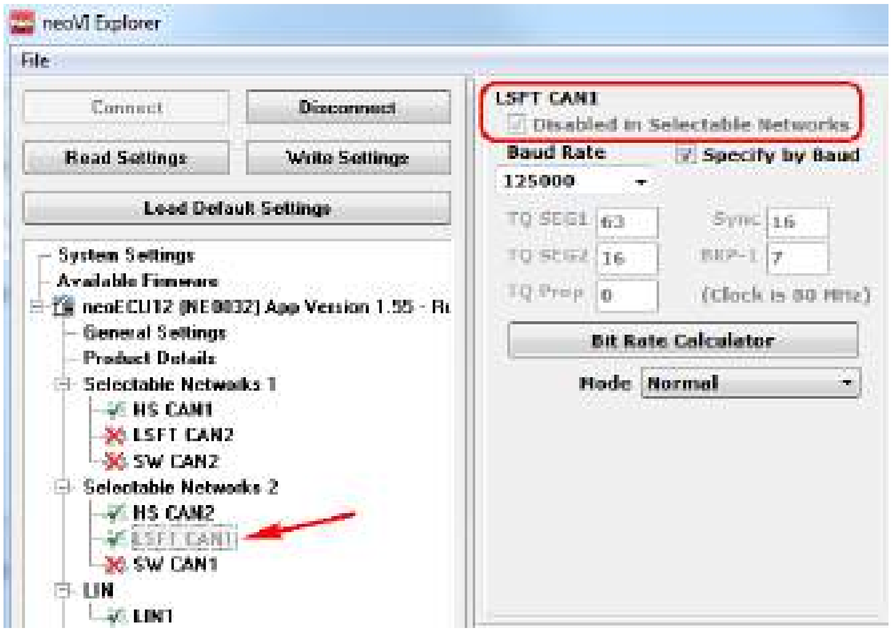
In this example you can see HS CAN 1, HS CAN 2, and LIN is enabled.



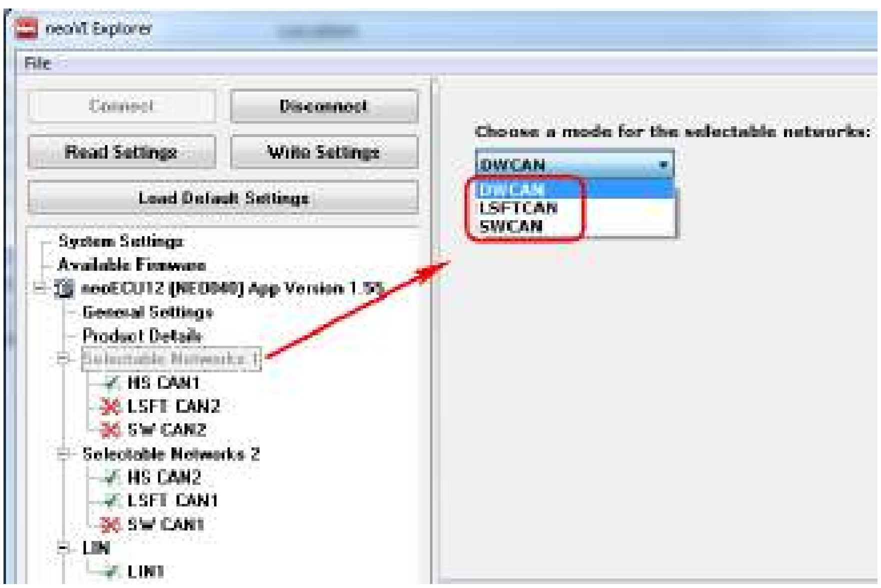
Now if you select “LSFT CAN” as well you can see it is enabled under the Selectable Networks 2 but HS CAN1 remains selected as well. Since they share the same pin the scenario is not desirable.



Now click on LSFT CAN1. You will see that it is actually disabled.

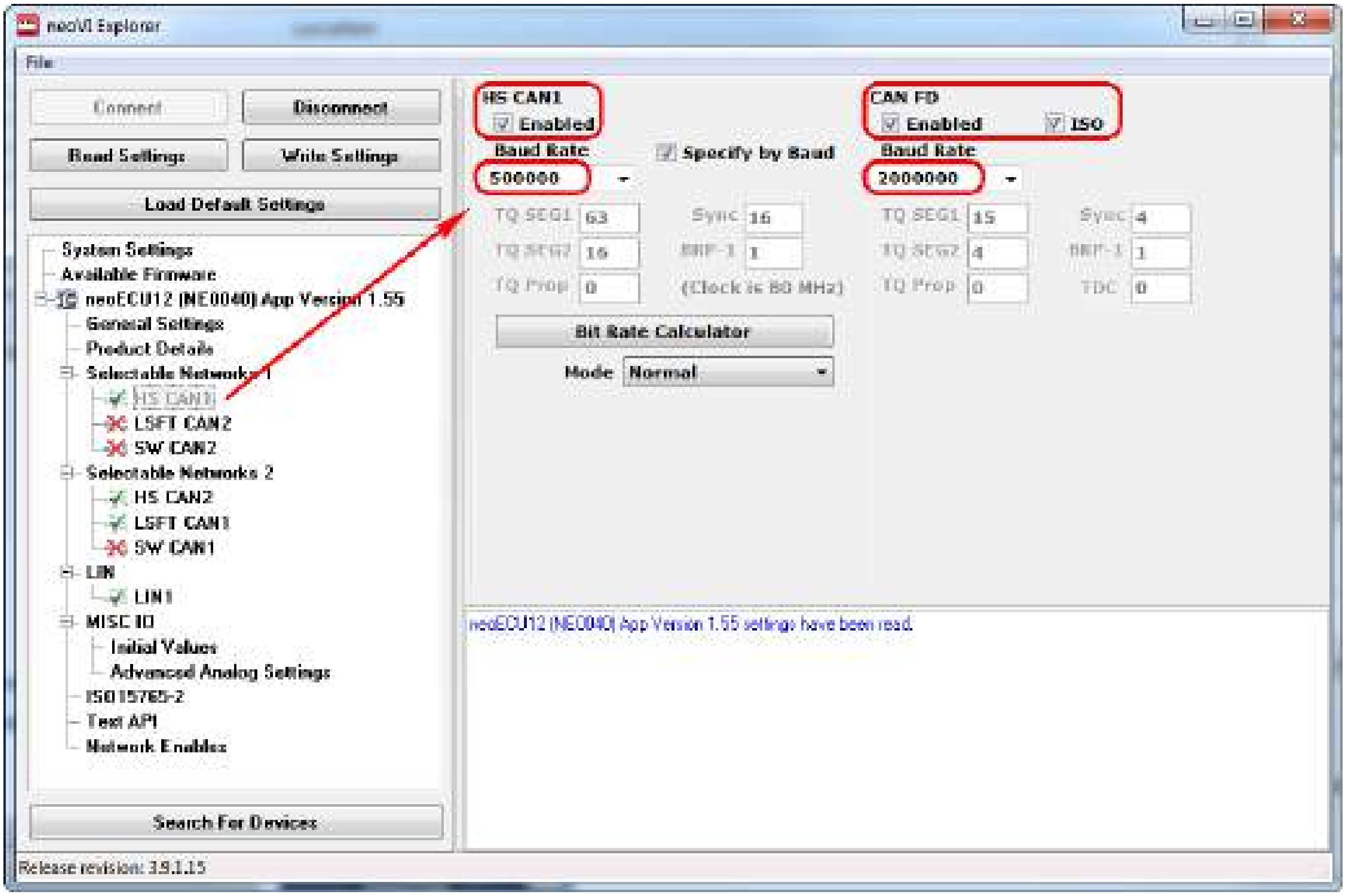


The best method for selecting the network is to click on "Selectable Networks 1" or "Selectable Networks 2" and from the pull-down menu make your selection.



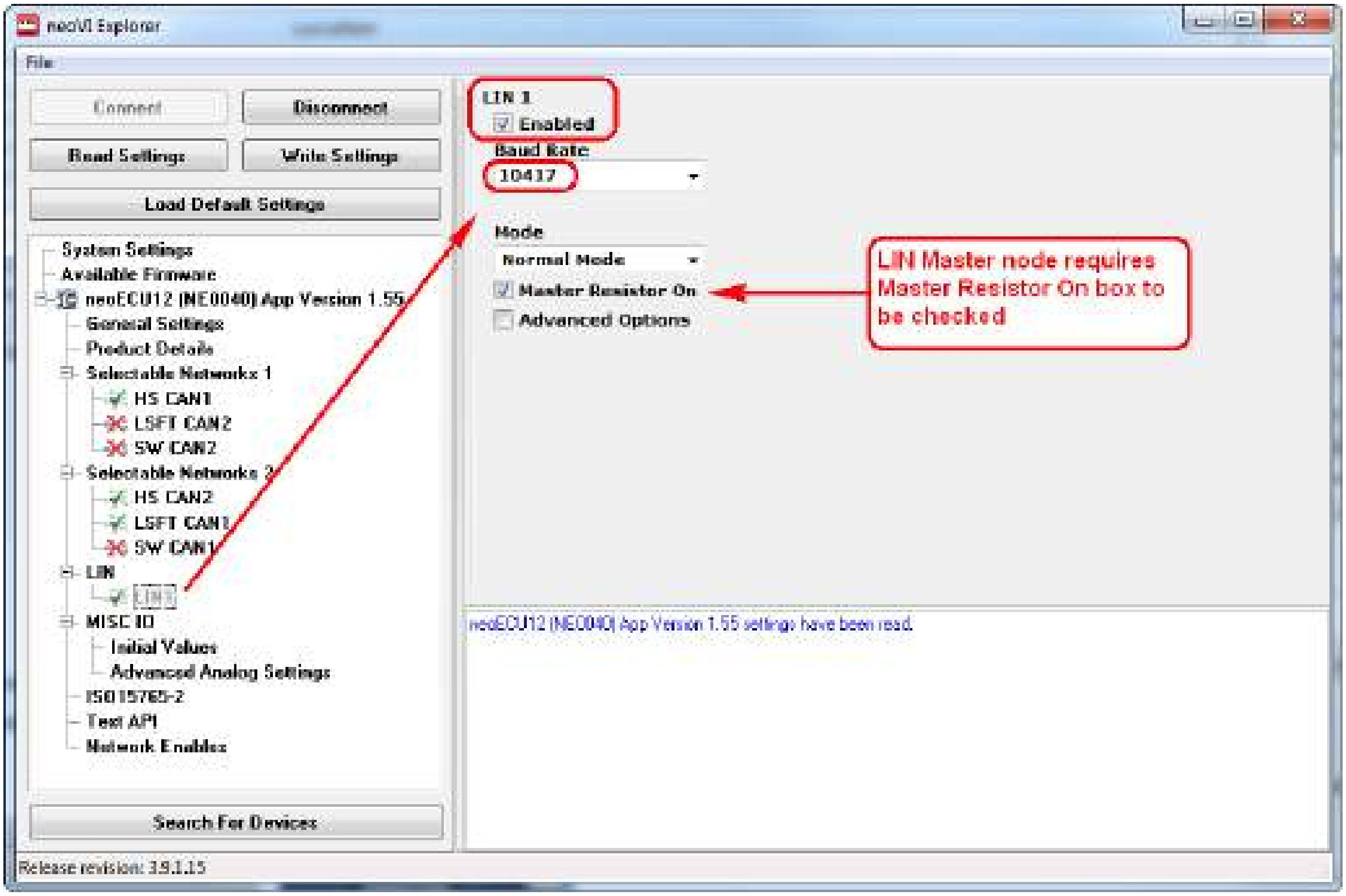
4.4 CAN Setup

Click on HS CAN1 (or HS CAN2) then click on the Enabled box to enable CAN and click on the CAN FD Enabled box to enable CAN FD. Set the baud rate appropriately. Click on the Write Settings button after you make any changes.



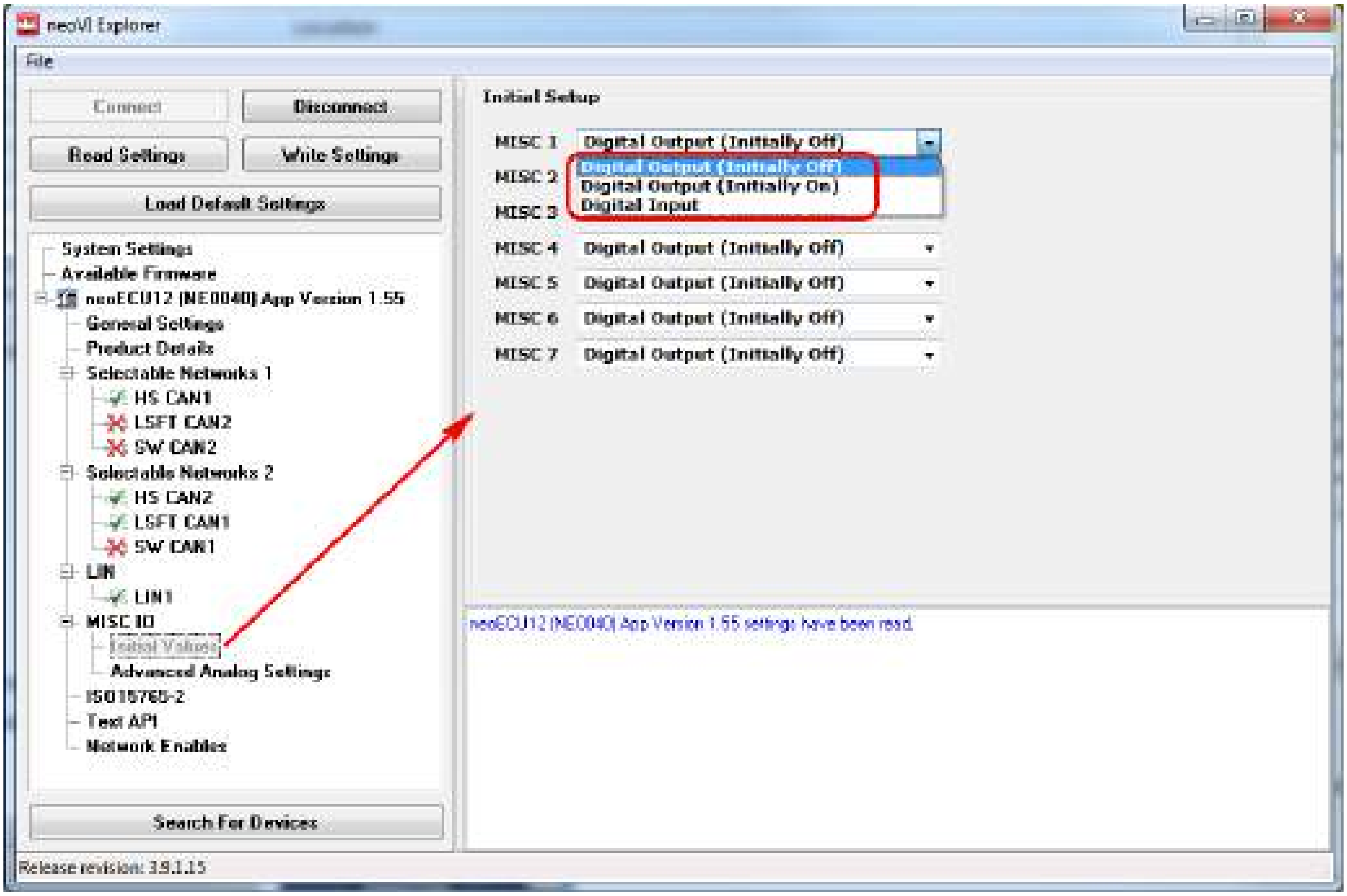
4.5 LIN Setup

Click on LIN1 then click on the Enabled box to enable LIN. Set the baud rate appropriately. Most cases the baud rate is 10417 or 19200. If the neoECU-12 is going to act as the LIN Master click on the Master Resistor On box. Click on the Write Settings button after you make any changes.



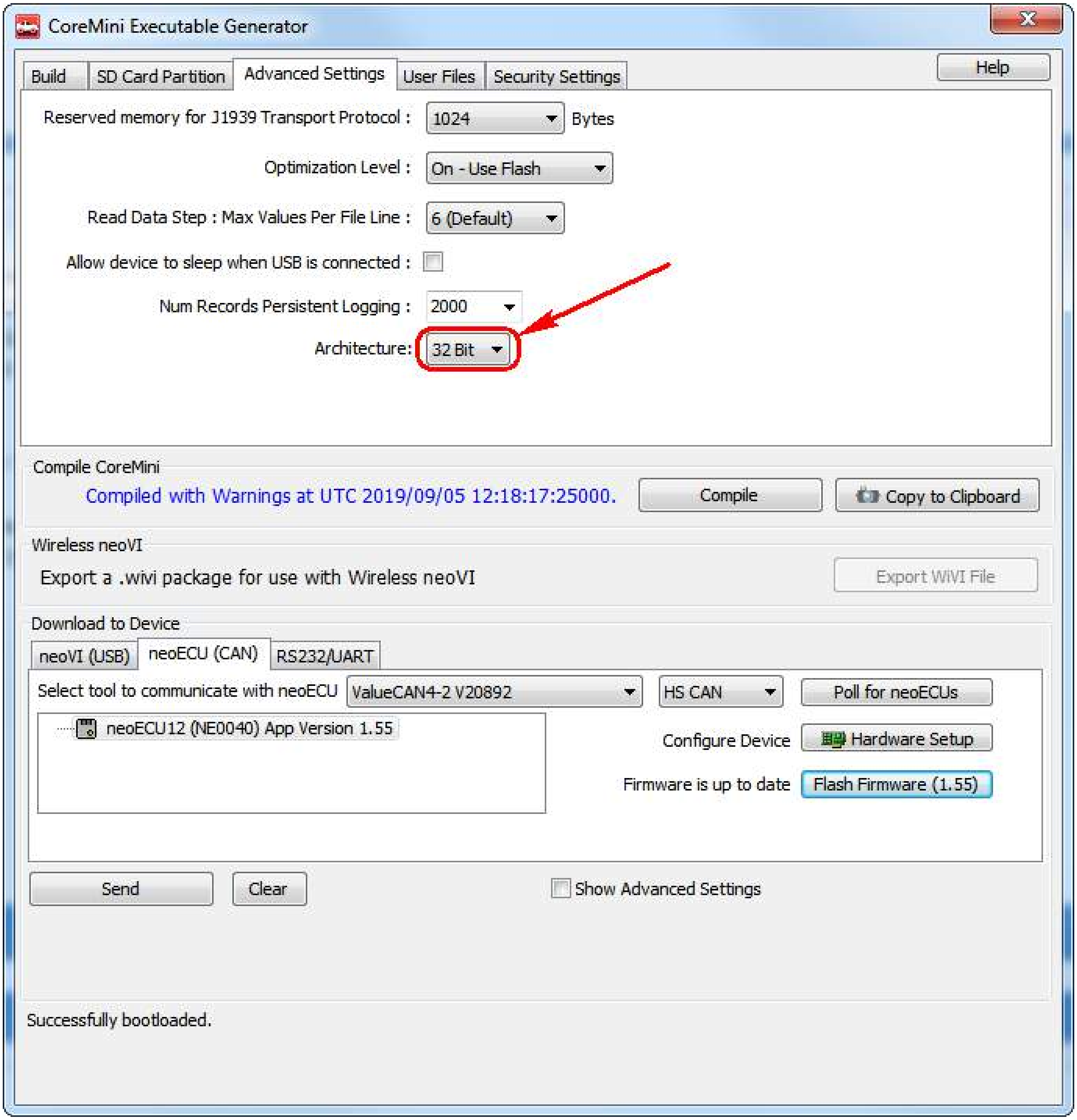
4.6 MISC IO Setup

Click on Initial Values then click on the pull-down menu to select Digital Output (Initially Off), Digital Output (Initially On), or Digital Input.

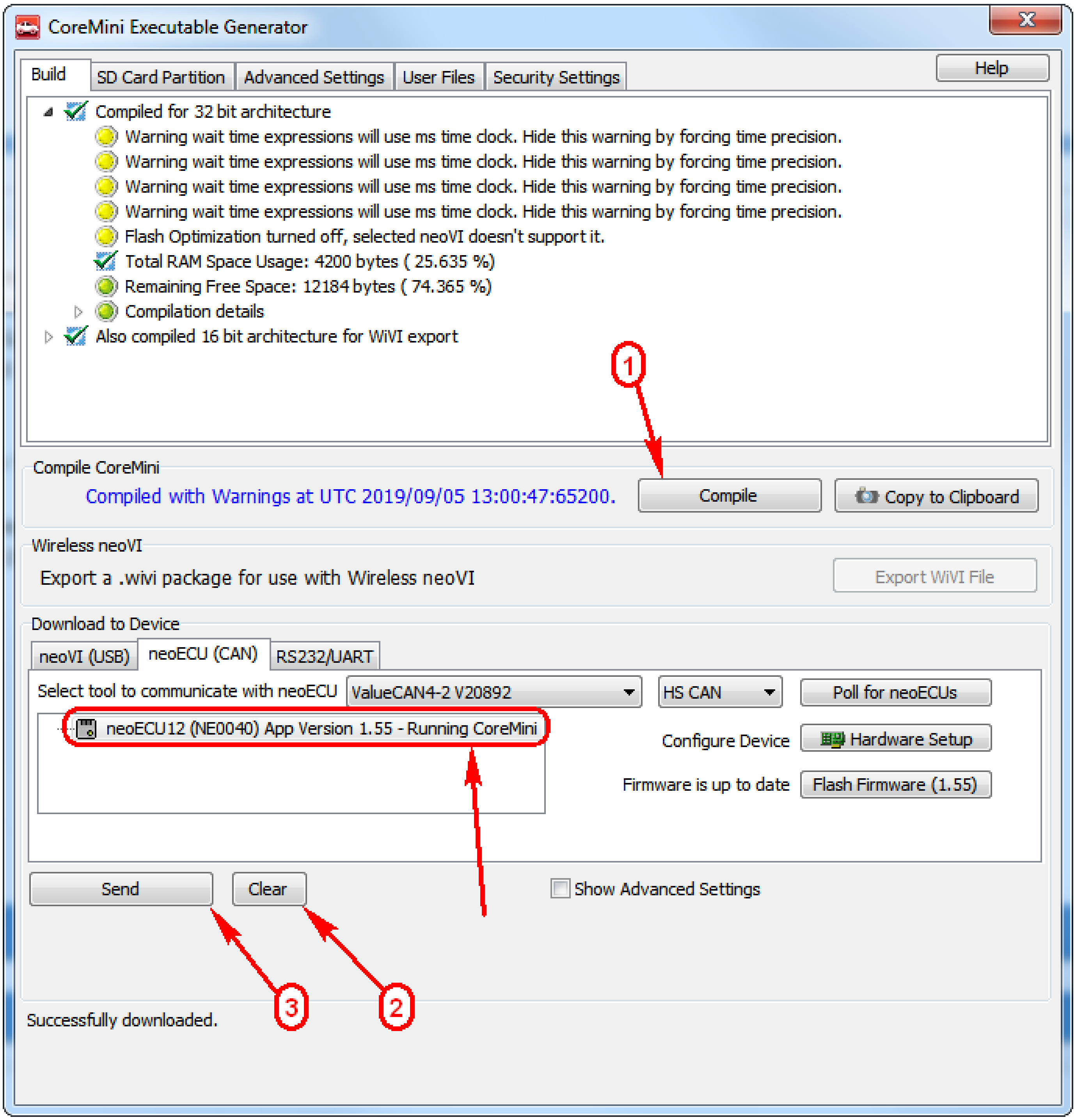


4.7 Program CoreMini

Next you will program the neoECU-12 with your custom script(s) that you created. But first click on the "Advanced Settings" tab and make sure the Architecture is set to **32 Bit** and not 16 Bit.

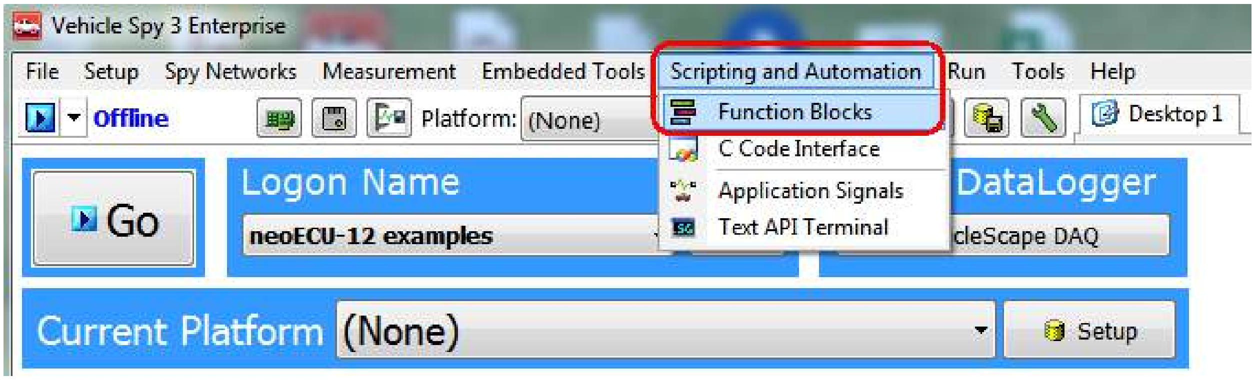


After selecting 32-bit click back to the Build tab. Click on the "Compile" button, followed by clicking on the "Clear" button, followed by clicking on the "Send" button. Once the progress bar is complete you have programmed the CoreMini of the device. The message at the bottom of the screen will show "Successfully downloaded" and the neoECU 12 shows "Running CoreMini".

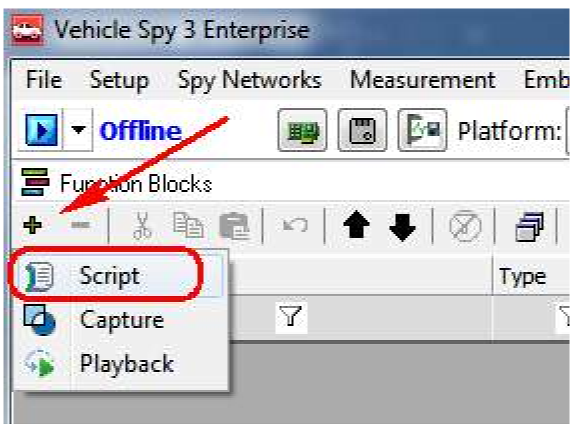


5.0 Function Block Scripts

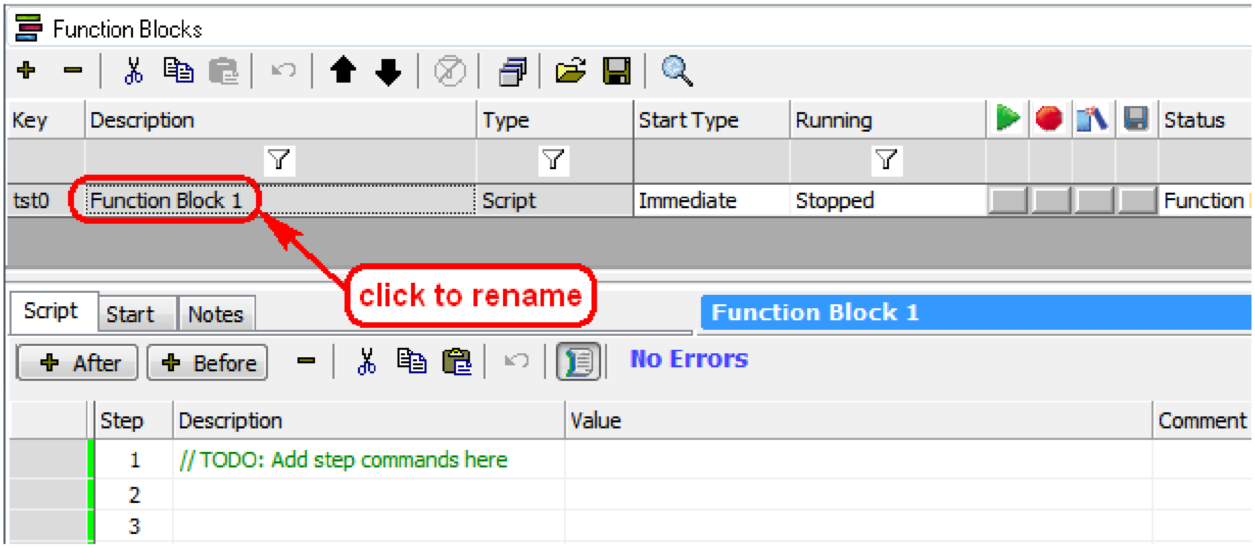
Next you will learn to program the neoECU-12 Function Block Scripts. From the Scripting and Automation pull-down menu select Function Blocks.



Next click on the "+" symbol and select "Script".

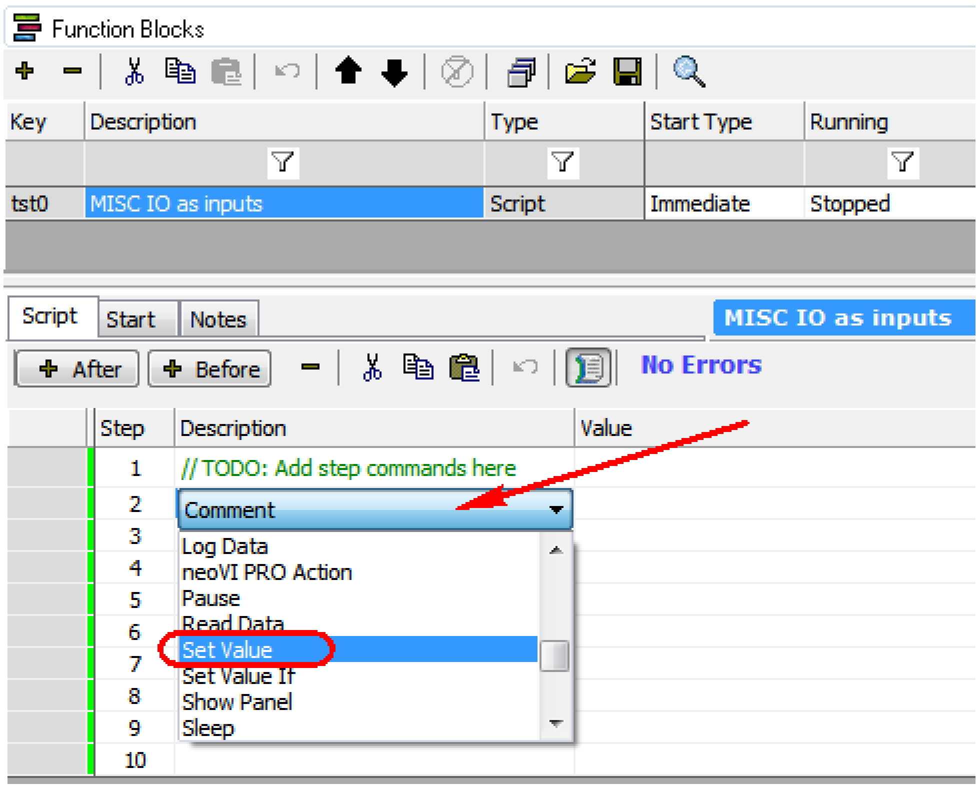


Click on the "Function Block 1" Description to rename it. For example, rename it to "MISC IO as inputs". This is not mandatory but it is good coding practice to name your scripts that are meaningful for debugging and code re-use.

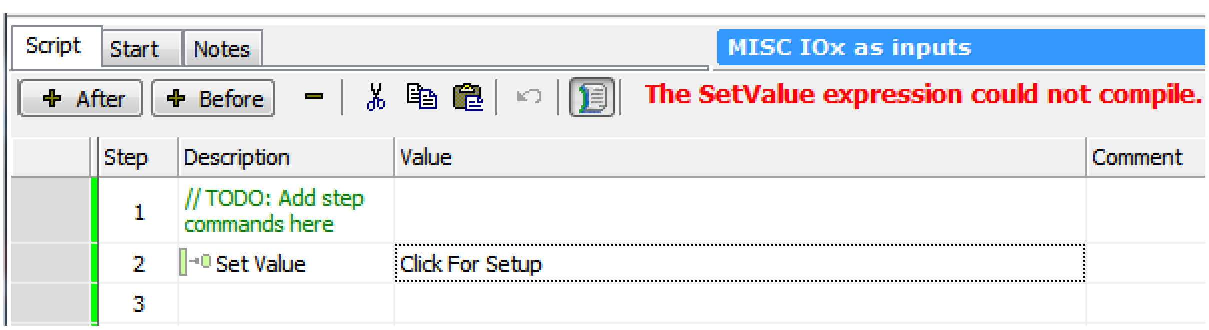


5.1 MISC IO as input(s)

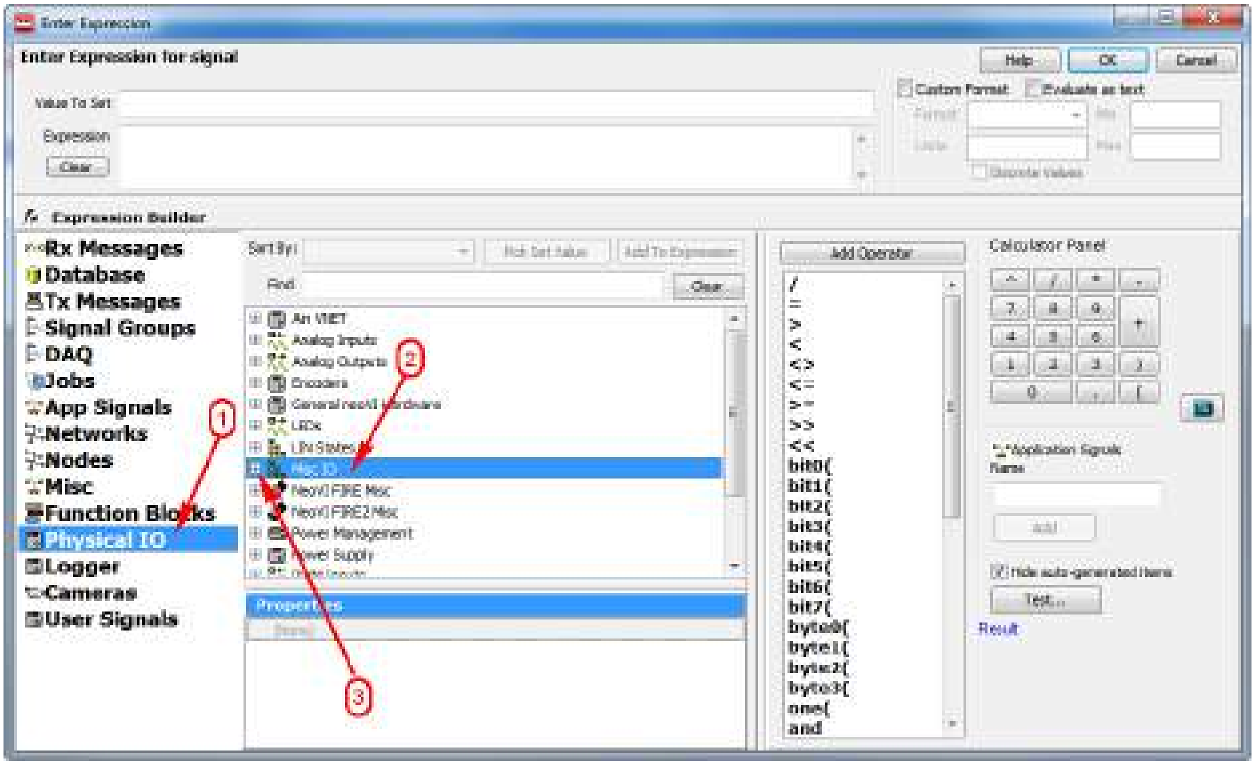
Next double-click on the empty cell row 2 column Description to bring up a pull-down menu of all the scripting commands. Select "Set Value".



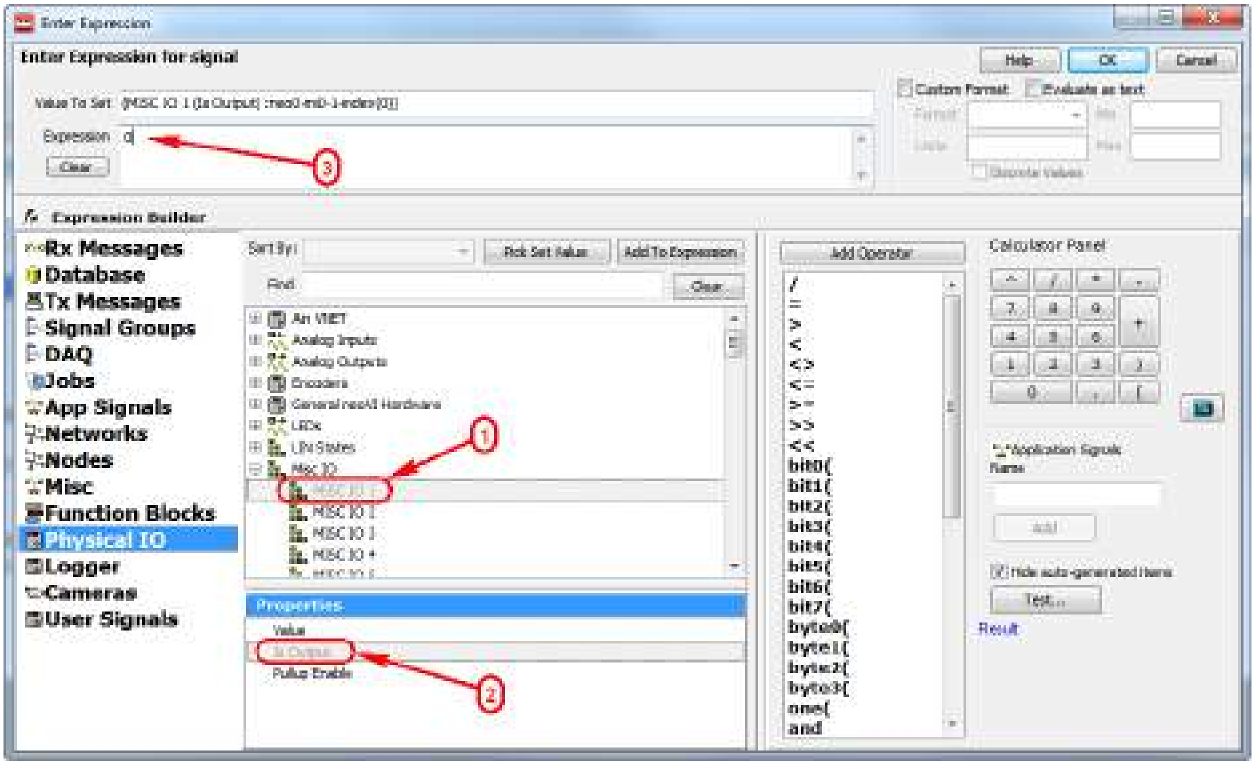
Next click on the empty cell row 2 column Value. Note the red text "The SetValue expression could not compile" message. Please note all error messages needs to be corrected before the script can be compiled and programmed into CoreMini. Once we set the value correctly the error message will go away.



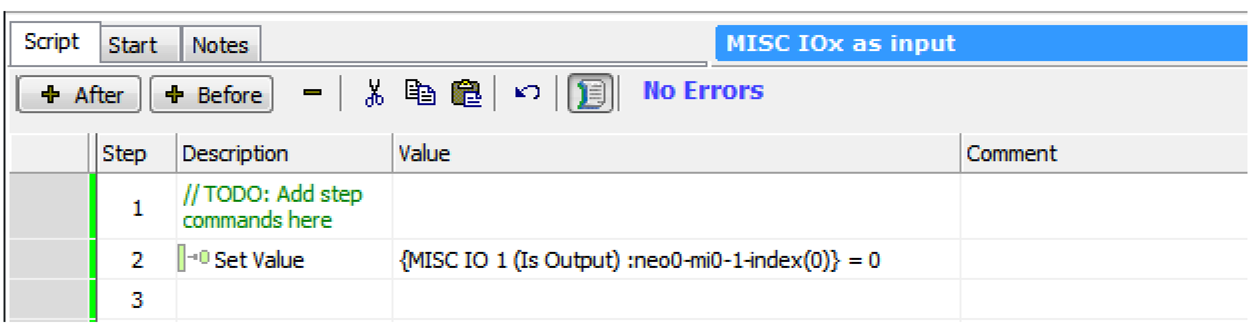
Next double-click where it shows "Click For Setup" The Expression Builder pop-up window will show. Click on "Physical IO" followed by "Misc IO" followed by the "+" symbol.



Next click on “MISC IO 1” followed by double-clicking on the “IS Output” property. This adds MISC IO 1 (Is Output) to the “Value to Set” box. Then put “0” for the Expression and click the OK button. “0” = input and “1” = output.



Note the Value column is now filled and the error message was replaced by “No Errors”.



Next add the appropriate comment by double-clicking in the comment box to the right.

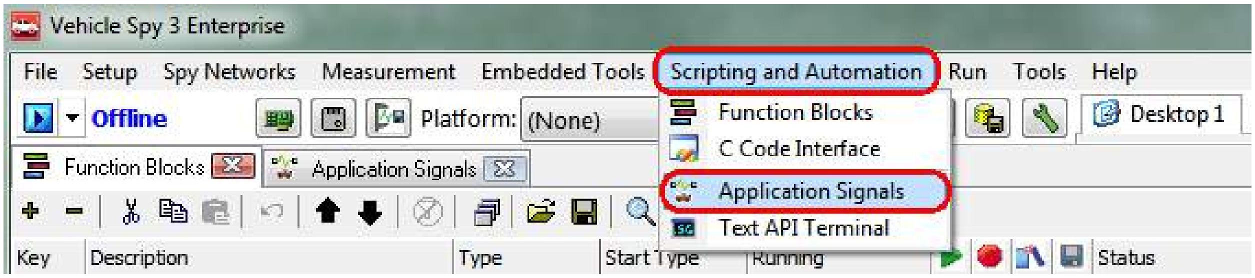
1	// TODO: Add step commands here		
2	Set Value	{MISC IO 1 (Is Output) :neo0-mi0-1-index(0)} = 0	// MISC IO 1 set as input
3			

Use the same format to setup the remaining MISC IO as inputs:

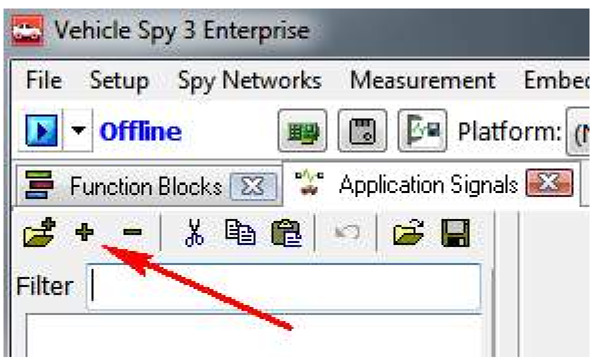
1	// TODO: Add step commands here		
2	Set Value	{MISC IO 1 (Is Output) :neo0-mi0-1-index(0)} = 0	// MISC IO 1 set as input
3	Set Value	{MISC IO 2 (Is Output) :neo0-mi1-1-index(0)} = 0	// MISC IO 2 set as input
4	Set Value	{MISC IO 3 (Is Output) :neo0-mi2-1-index(0)} = 0	// MISC IO 3 set as input
5	Set Value	{MISC IO 4 (Is Output) :neo0-mi3-1-index(0)} = 0	// MISC IO 4 set as input
6	Set Value	{MISC IO 5 (Is Output) :neo0-mi4-1-index(0)} = 0	// MISC IO 5 set as input
7	Set Value	{MISC IO 6 (Is Output) :neo0-mi5-1-index(0)} = 0	// MISC IO 6 set as input
8	Set Value	{MISC IO 7 (Is Output) :neo0-mi6-1-index(0)} = 0	// MISC IO 7 set as input

The next section is optional.

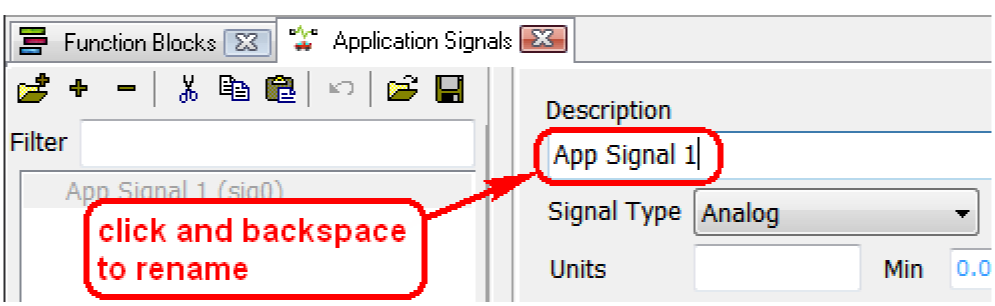
Add variables (Application Signals) to your script. From the Scripting and Automation pull-down menu select Application Signals.



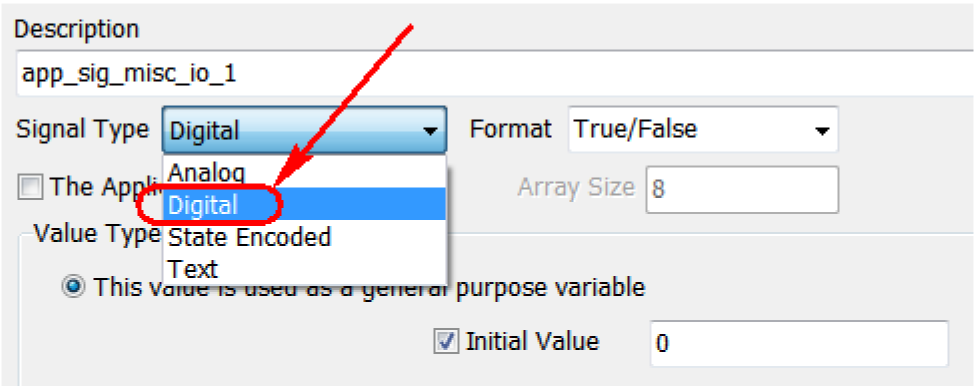
Next click on the "+" symbol to add "Application Signals". These are like variables in C code. Clicking on the "+" symbol again to add additional application signals and clicking on the "-" symbol will delete the application signal that is highlighted.



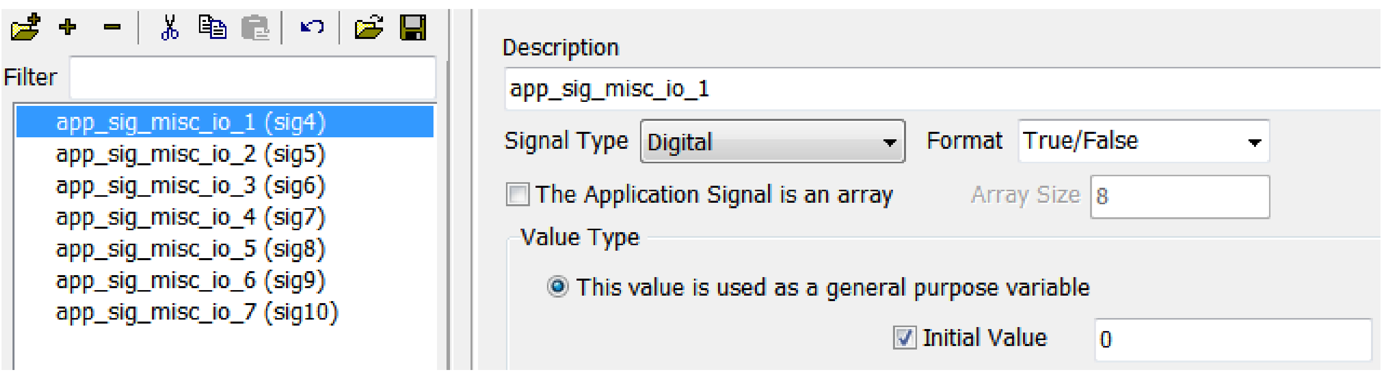
Next rename the variable to something that is more meaningful. Click on the description and backspace to rename. Since we are creating a variable for MISC IO 1 input let's name it "var_misc_io_1_input" for example.



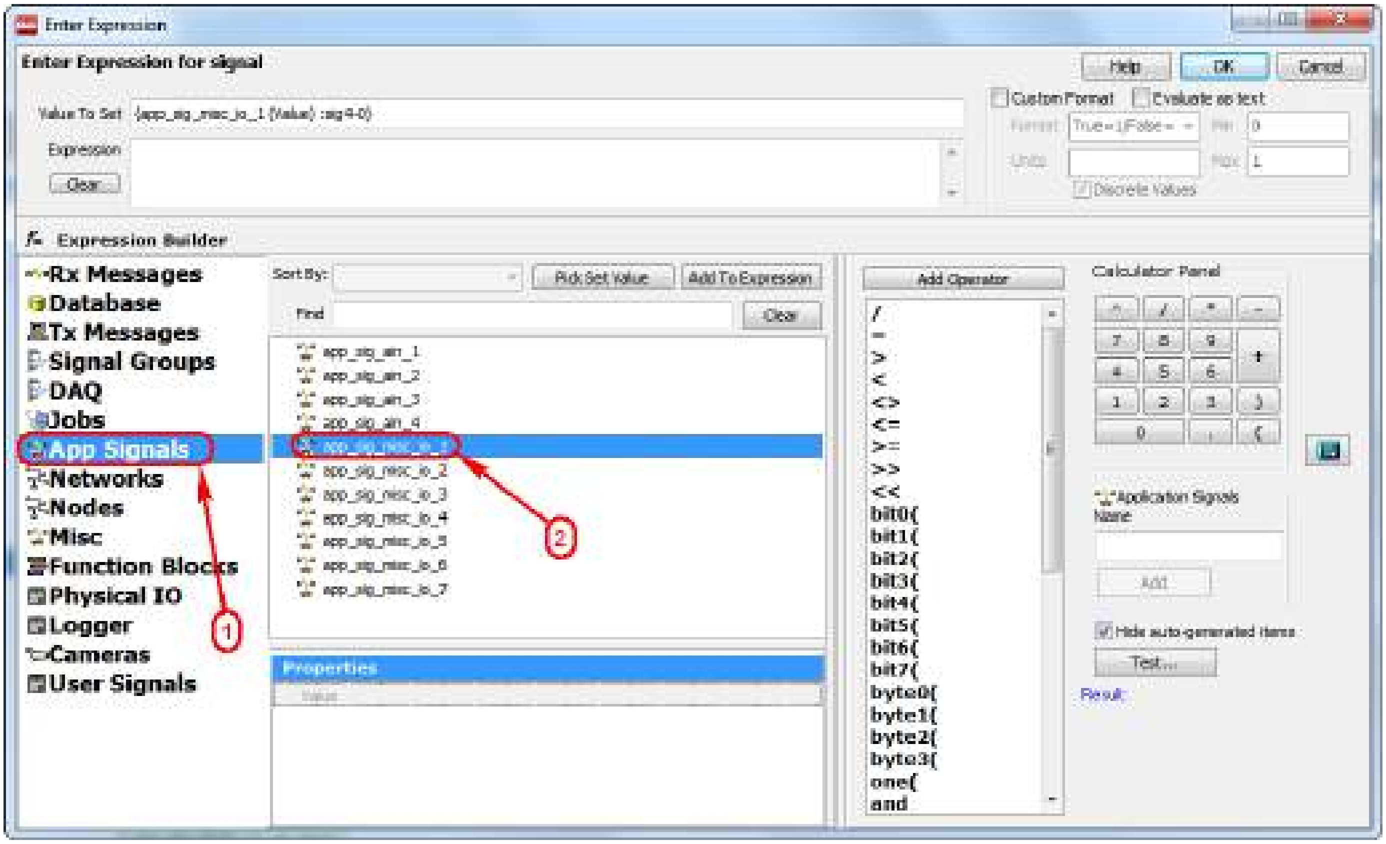
Next set the Signal Type to "Digital". When we read the port it will be "low" or "high".



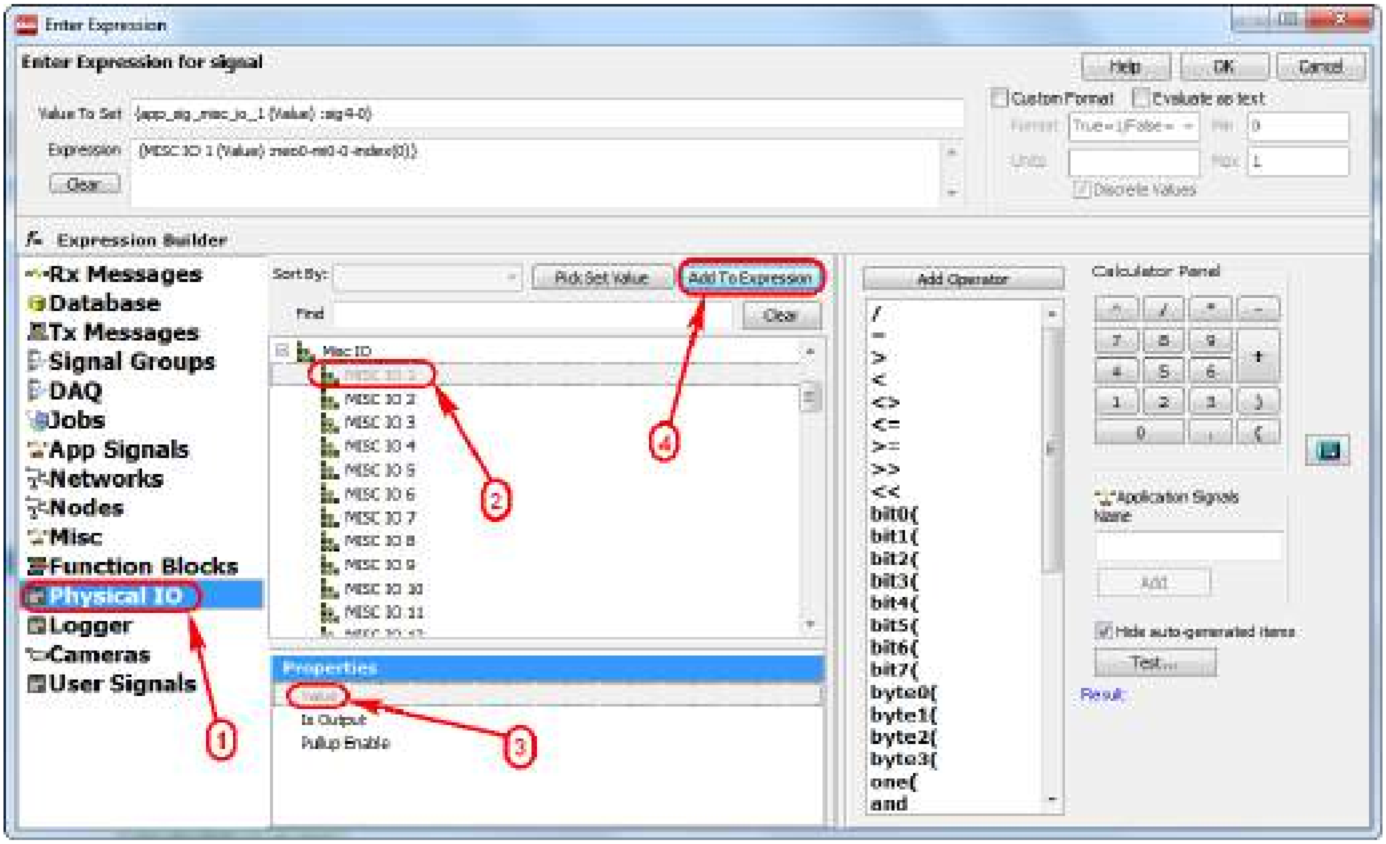
Do this for the remaining MISC IO inputs. Note the Format is defaulted to "True/False" but you can switch to "On/Off", "Yes/No", "Passed/Failed", "Open/Closed", etc…



Next, let's read the ports and assign the status to their respective variables. Select "Set Value" as the command double-click in the Value column. Click on App Signals then double-click on "app_sig_misc_io_1".



Next, click on "Physical IO" followed by "MISC IO 1" followed by "Value" followed by "Add to Expression". Then click "OK".



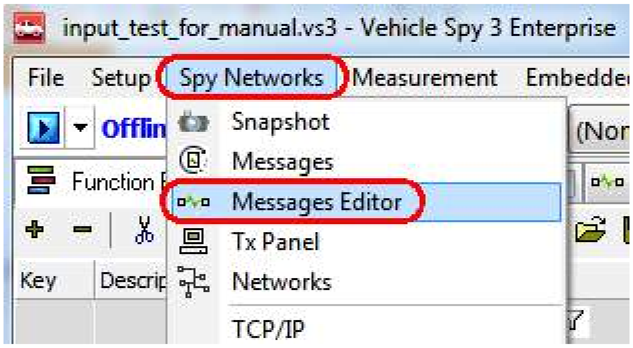
Repeat for all MISC IO. Below shows the MISC IO ports to be read into their respective variable.

10	Set Value	{app_sig_misc_io_1 (Value) :sig4-0} = {MISC IO 1 (Value) :neo0-mi0-0-index(0)}	// read misc io 1 into app signal 1
11	Set Value	{app_sig_misc_io_2 (Value) :sig5-0} = {MISC IO 2 (Value) :neo0-mi1-0-index(0)}	// read misc io 2 into app signal 2
12	Set Value	{app_sig_misc_io_3 (Value) :sig6-0} = {MISC IO 3 (Value) :neo0-mi2-0-index(0)}	// read misc io 3 into app signal 3
13	Set Value	{app_sig_misc_io_4 (Value) :sig7-0} = {MISC IO 4 (Value) :neo0-mi3-0-index(0)}	// read misc io 4 into app signal 4
14	Set Value	{app_sig_misc_io_5 (Value) :sig8-0} = {MISC IO 5 (Value) :neo0-mi4-0-index(0)}	// read misc io 5 into app signal 5
15	Set Value	{app_sig_misc_io_6 (Value) :sig9-0} = {MISC IO 6 (Value) :neo0-mi5-0-index(0)}	// read misc io 6 into app signal 6
16	Set Value	{app_sig_misc_io_7 (Value) :sig10-0} = {MISC IO 7 (Value) :neo0-mi6-0-index(0)}	// read misc io 7 into app signal 7

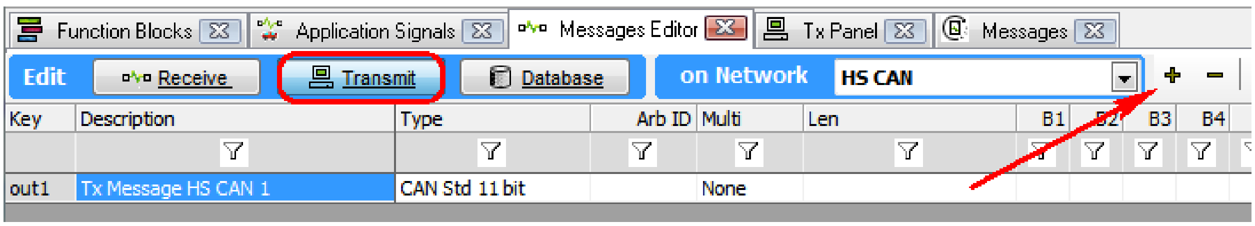
End of optional section.

5.2 Send MISC IO port status in CAN message.

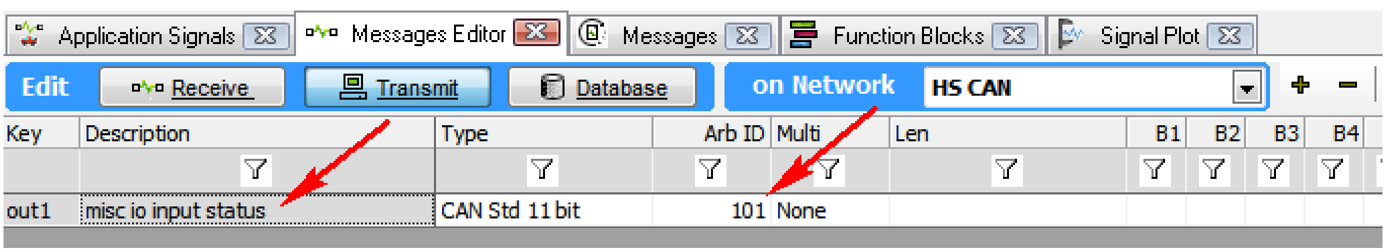
From the Spy Networks pull-down menu select Message Editor.



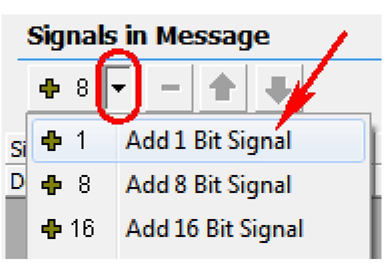
Click on the Transmit button then click on the "+" symbol to create a new HS CAN message.



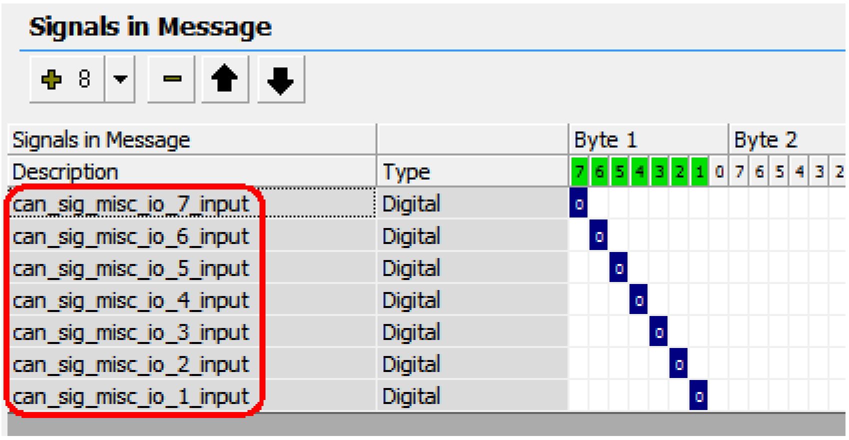
Click on "Tx Message HS CAN 1" to rename it (i.e. misc_io_input_status) and set the Arb ID to a value not being used (i.e. 101).



Add seven 1-bit CAN signals to this message. Click on the down arrow next to "+8" and select "+1". Then click the "+1" six more times.



Next rename the signals to something more meaningful (i.e. can_sig_misc_io_1_input).



Next modify the script to get the MISC IO port value and copy it into its respective CAN signal. In this example the port is read and copied into the CAN signals then the CAN message is sent every 50 ms.

*NOTE – If you had done the optional section above and created application signals and read the MISC IO into the application signals then you would copy the application signals to their respective can signals. The image shown below is copying the MISC IO directly to their respective can signals. Either way works.

26	Set Value	{can_sig_misc_io_1_input (Value) :out1-sig6-0} = {MISC IO 1 (Value) :neo0-mi0-0-index(0)}	// copy misc io 1 to can signal 1
27	Set Value	{can_sig_misc_io_2_input (Value) :out1-sig5-0} = {MISC IO 2 (Value) :neo0-mi1-0-index(0)}	// copy misc io 2 to can signal 2
28	Set Value	{can_sig_misc_io_3_input (Value) :out1-sig4-0} = {MISC IO 3 (Value) :neo0-mi2-0-index(0)}	// copy misc io 3 to can signal 3
29	Set Value	{can_sig_misc_io_4_input (Value) :out1-sig3-0} = {MISC IO 4 (Value) :neo0-mi3-0-index(0)}	// copy misc io 4 to can signal 4
30	Set Value	{can_sig_misc_io_5_input (Value) :out1-sig2-0} = {MISC IO 5 (Value) :neo0-mi4-0-index(0)}	// copy misc io 5 to can signal 5
31	Set Value	{can_sig_misc_io_6_input (Value) :out1-sig1-0} = {MISC IO 6 (Value) :neo0-mi5-0-index(0)}	// copy misc io 6 to can signal 6
32	Set Value	{can_sig_misc_io_7_input (Value) :out1-sig0-0} = {MISC IO 7 (Value) :neo0-mi6-0-index(0)}	// copy misc io 7 to can signal 7
33	Wait For	= 50 ms	// delay 50 ms
34	Transmit	misc io input status (out1)	// transmit status message
35	Jump To	Step 10	// read misc io inputs again

Program CoreMini with this script. Connect another Intrepid tool to the neoECU-12 and monitor the HS CAN bus. You should see the status of all seven input ports.

*NOTE – If you do not connect a voltage to the pins they tend to float high (3.3V) except for IO7 which is a 5V tolerant IO pin. It floats low as you cen see from the image below. All of the other pins floated high except IO3 which was connected to GND.

	Count	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes
Filter								
	267	56.006 ms			misc io input status	101	1	76
	can_sig_misc_io_7_input				= False [0]			
	can_sig_misc_io_6_input				= True [1]			
	can_sig_misc_io_5_input				= True [1]			
	can_sig_misc_io_4_input				= True [1]			
	can_sig_misc_io_3_input				= False [0]			
	can_sig_misc_io_2_input				= True [1]			
	can_sig_misc_io_1_input				= True [1]			

5.3 MISC IO as output(s)

Setting the MISC IO as outputs is the same procedure as setting the port to an input but the expression is set to "1" instead of "0".

1	// TODO: Add step commands here		
2	Set Value	{MISC IO 1 (Is Output) :neo0-mi0-1-index(0)} = 1	// MISC IO 1 set as output
3	Set Value	{MISC IO 2 (Is Output) :neo0-mi1-1-index(0)} = 1	// MISC IO 2 set as output
4	Set Value	{MISC IO 3 (Is Output) :neo0-mi2-1-index(0)} = 1	// MISC IO 3 set as output
5	Set Value	{MISC IO 4 (Is Output) :neo0-mi3-1-index(0)} = 1	// MISC IO 4 set as output
6	Set Value	{MISC IO 5 (Is Output) :neo0-mi4-1-index(0)} = 1	// MISC IO 5 set as output
7	Set Value	{MISC IO 6 (Is Output) :neo0-mi5-1-index(0)} = 1	// MISC IO 6 set as output
8	Set Value	{MISC IO 7 (Is Output) :neo0-mi6-1-index(0)} = 1	// MISC IO 7 set as output

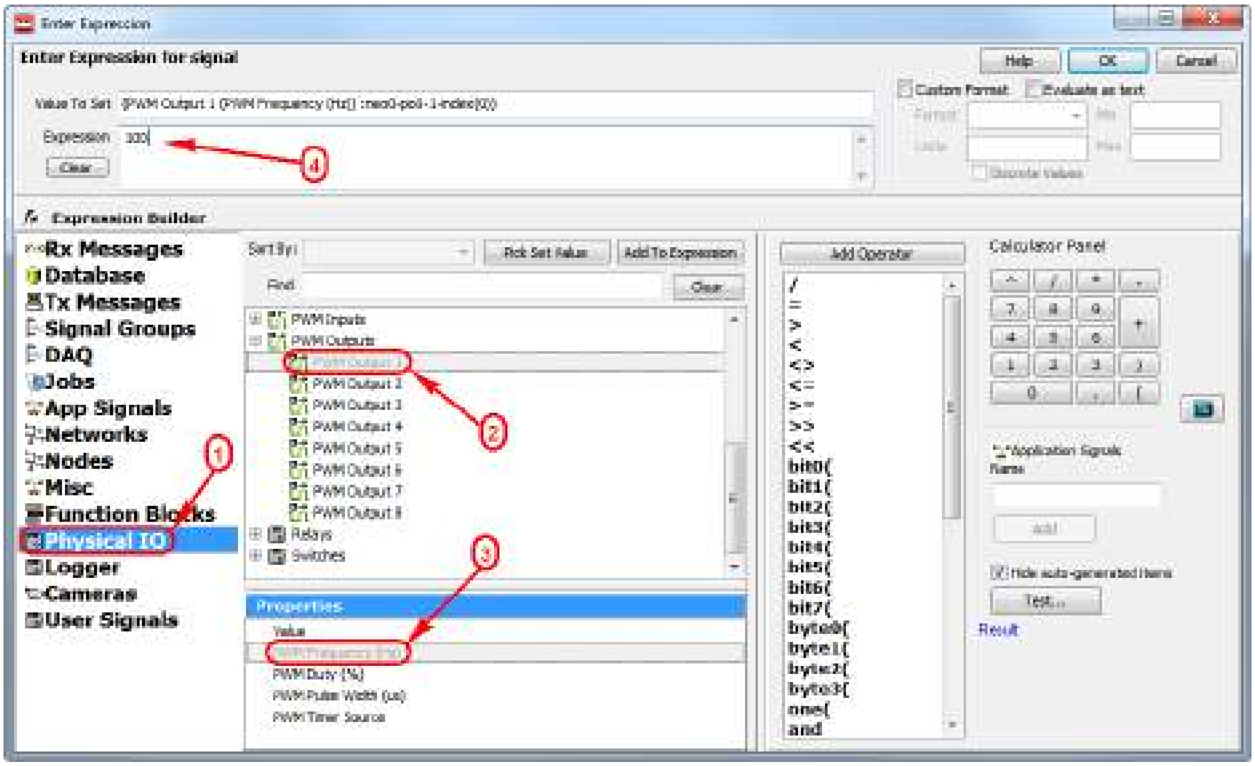
To write to the port set the MISC IO value property. In the example below the ports are alternated "high" and "low".

9			
10	Set Value	{MISC IO 1 (Value) :neo0-mi0-0-index(0)} = 1	// MISC IO 1 is set high
11	Set Value	{MISC IO 2 (Value) :neo0-mi1-0-index(0)} = 0	// MISC IO 2 is set low
12	Set Value	{MISC IO 3 (Value) :neo0-mi2-0-index(0)} = 1	// MISC IO 3 is set high
13	Set Value	{MISC IO 4 (Value) :neo0-mi3-0-index(0)} = 0	// MISC IO 4 is set low
14	Set Value	{MISC IO 5 (Value) :neo0-mi4-0-index(0)} = 1	// MISC IO 5 is set high
15	Set Value	{MISC IO 6 (Value) :neo0-mi5-0-index(0)} = 0	// MISC IO 6 is set low
16	Set Value	{MISC IO 7 (Value) :neo0-mi6-0-index(0)} = 1	// MISC IO 7 is set high
17			

5.4 MISC IO as PWM outputs

There are five PWM Outputs. Please note channel 3 PWM is mirrored on channel 6 PWM.

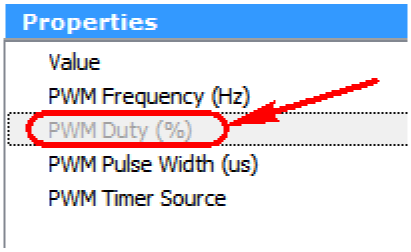
Setting the MISC IO as PWM outputs is like setting the MISC IO as outputs. After selecting the “Set Value” command double-click where it shows “Click For Setup”. The Expression Builder pop-up window will show. Click on “Physical IO” followed by “PWM Outputs” followed by the “+” symbol. Select PWM Output 1 and then double-click on “PWM Frequency”. Set the frequency in the Expression box. In this example, the frequency is set to 100 Hz.



Continue setting the PWM Output 2 thru 5 to 100Hz as shown below.

2	Set Value	{PWM Output 1 (PWM Frequency (Hz)) :neo0-po0-1-index(0)} = 100	// PWM 1 freq set to 100 Hz
3	Set Value	{PWM Output 2 (PWM Frequency (Hz)) :neo0-po1-1-index(0)} = 100	// PWM 2 freq set to 100 Hz
4	Set Value	{PWM Output 3 (PWM Frequency (Hz)) :neo0-po2-1-index(0)} = 100	// PWM 3 freq set to 100 Hz
5	Set Value	{PWM Output 4 (PWM Frequency (Hz)) :neo0-po3-1-index(0)} = 100	// PWM 4 freq set to 100 Hz
6	Set Value	{PWM Output 5 (PWM Frequency (Hz)) :neo0-po4-1-index(0)} = 100	// PWM 5 freq set to 100 Hz
7			

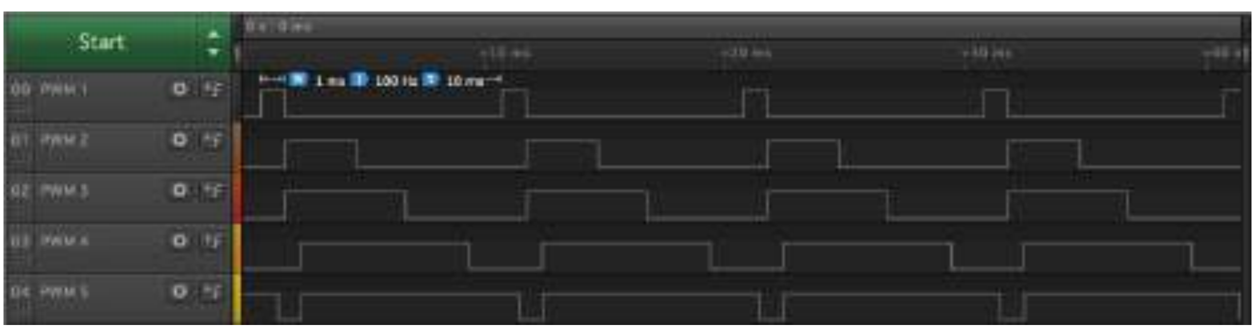
Repeat the same steps for setting the Duty Cycle for all five PWM channels.



In this example the duty cycle is 10%, 30%, 50%, 70%, and 90%, respectively for the five PWM channels.

2	Set Value	{PWM Output 1 (PWM Frequency (Hz)) :neo0-po0-1-index(0)} = 100	// PWM 1 freq set to 100 Hz
3	Set Value	{PWM Output 2 (PWM Frequency (Hz)) :neo0-po1-1-index(0)} = 100	// PWM 2 freq set to 100 Hz
4	Set Value	{PWM Output 3 (PWM Frequency (Hz)) :neo0-po2-1-index(0)} = 100	// PWM 3 freq set to 100 Hz
5	Set Value	{PWM Output 4 (PWM Frequency (Hz)) :neo0-po3-1-index(0)} = 100	// PWM 4 freq set to 100 Hz
6	Set Value	{PWM Output 5 (PWM Frequency (Hz)) :neo0-po4-1-index(0)} = 100	// PWM 5 freq set to 100 Hz
7			
8	Set Value	{PWM Output 1 (PWM Duty (%)) :neo0-po0-2-index(0)} = 10	// PWM 1 duty cycle set to 10%
9	Set Value	{PWM Output 2 (PWM Duty (%)) :neo0-po1-2-index(0)} = 30	// PWM 2 duty cycle set to 30%
10	Set Value	{PWM Output 3 (PWM Duty (%)) :neo0-po2-2-index(0)} = 50	// PWM 3 duty cycle set to 50%
11	Set Value	{PWM Output 4 (PWM Duty (%)) :neo0-po3-2-index(0)} = 70	// PWM 4 duty cycle set to 70%
12	Set Value	{PWM Output 5 (PWM Duty (%)) :neo0-po4-2-index(0)} = 90	// PWM 5 duty cycle set to 90%
13			

Below is a screenshot of the five PWM channels. Each measured 10%, 30%, 50%, 70%, 90% duty cycles, respectively.



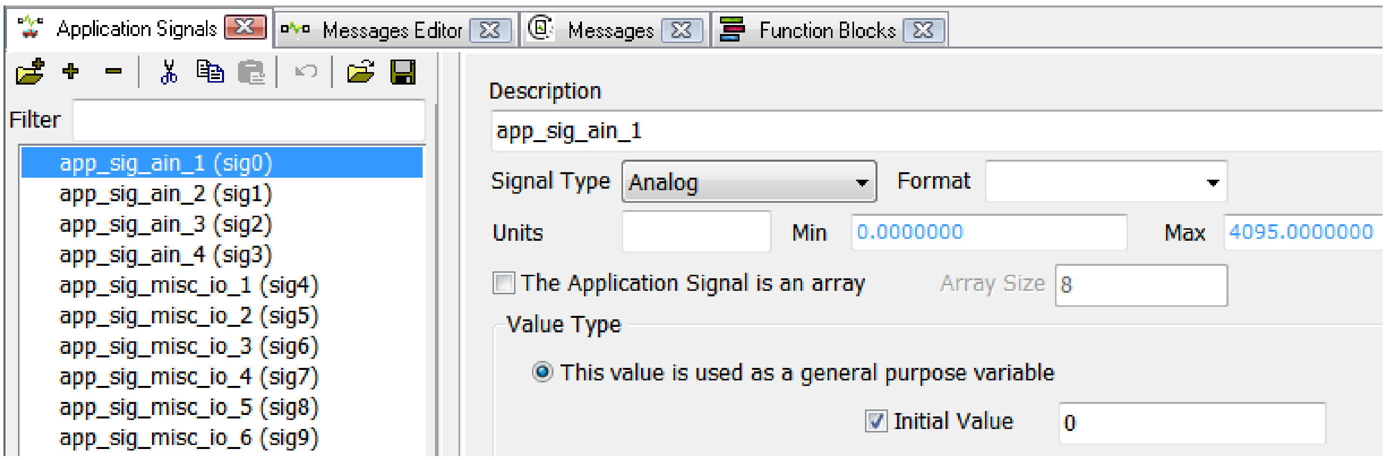
Note: The PWM output is 3.3V signal driven by a series 10K resistor. Please buffer accordingly to drive higher current loads.

5.5 MISC IO as Analog Inputs

There are four 0-5V Analog inputs. These are connected to 12-bit A/D Converters so the maximum value read from the A/D Converter is 4095 counts.

The next section is optional (just like it was for the MISC IO as inputs).

Add variables (Application Signals) to your script. From the Scripting and Automation pull-down menu select Application Signals. Add 4 variables and rename them appropriately. Note the Signal Type is set to "Analog" and the Max Value is set to "4095".



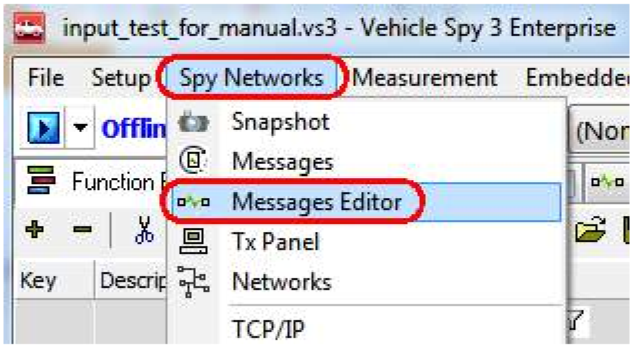
Now read the analog inputs into the variables.

2	Set Value	{app_sig_ain_1 (Value) :sig0-0} = {Analog Input 1 (Value) :neo0-ai0-0-index(0)}	// read ain 1 into app signal 1
3	Set Value	{app_sig_ain_2 (Value) :sig1-0} = {Analog Input 2 (Value) :neo0-ai1-0-index(0)}	// read ain 2 into app signal 2
4	Set Value	{app_sig_ain_3 (Value) :sig2-0} = {Analog Input 3 (Value) :neo0-ai2-0-index(0)}	// read ain 3 into app signal 3
5	Set Value	{app_sig_ain_4 (Value) :sig3-0} = {Analog Input 4 (Value) :neo0-ai3-0-index(0)}	// read ain 4 into app signal 4

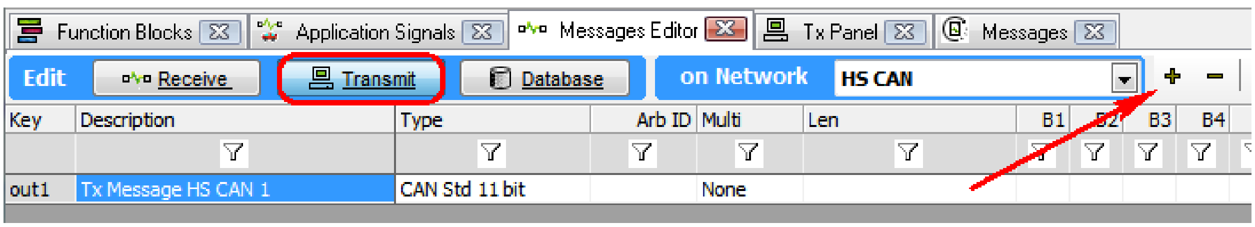
End of optional section.

5.6 Send Analog input values in CAN message

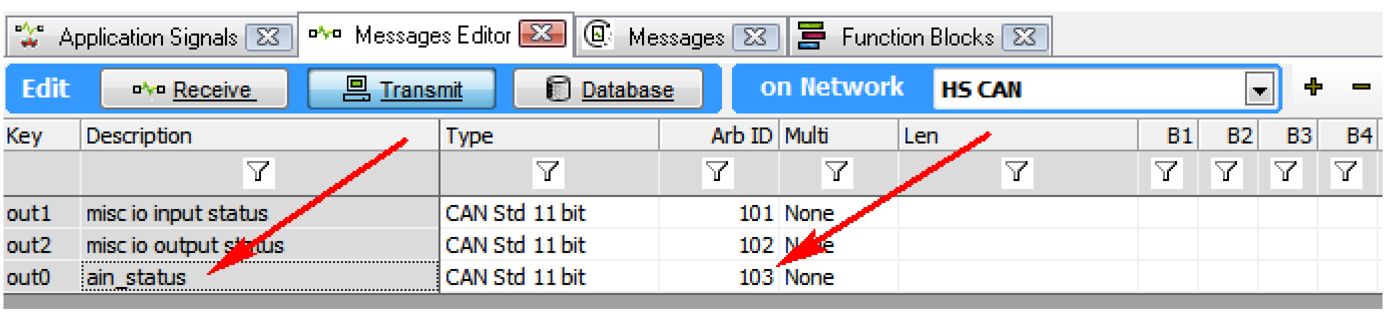
From the Spy Networks pull-down menu select Message Editor.



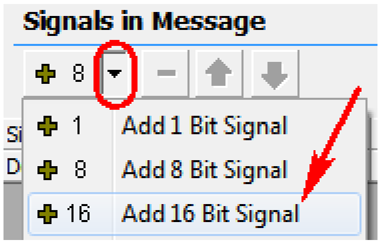
Click on the Transmit button then click on the "+" symbol to create a new HS CAN message.



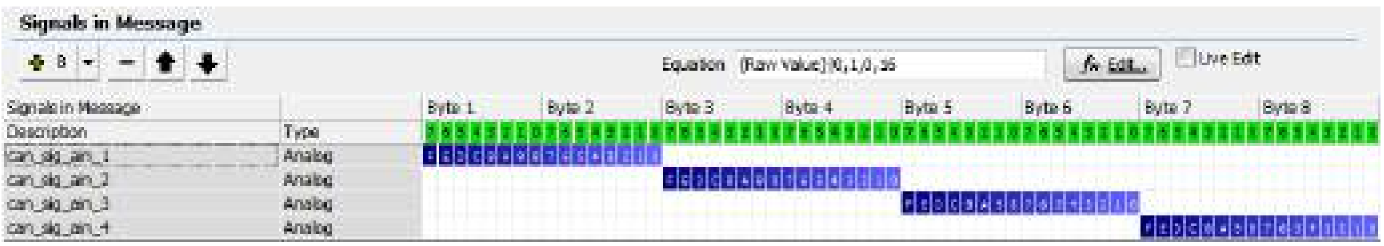
Click on "Tx Message HS CAN 3" to rename it (i.e. ain_status) and set the Arb ID to a value not being used (i.e. 103).



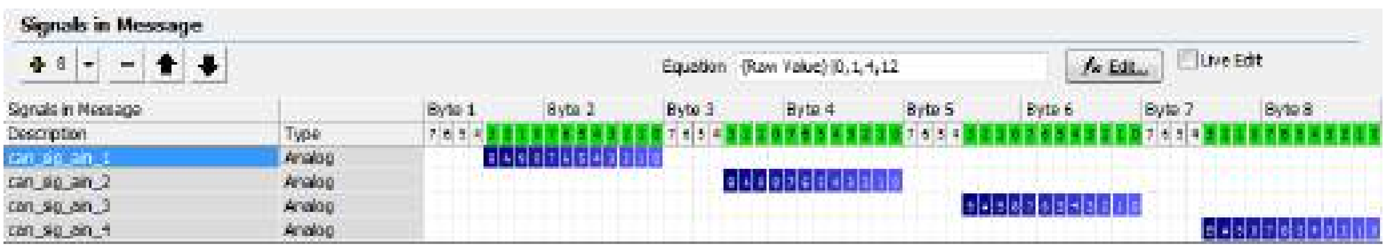
Add four 16-bit CAN signals to this message. Click on the down arrow next to "+8" and select "+16". Then click the "+1" three more times.



Next rename the signals to something more meaningful (i.e. can_sig_ain_1).



Or re-size the 16-bit signal values to 12-bit values by dragging the edge of the blue box inwards.



Next modify the script to get the analog in value and copy it into its respective CAN signal. In this example the port is read and directly copied into the CAN signals then the CAN message is sent every 10 ms.

2	Set Value	{can_sig_ain_1 (Value) :out0-sig0-0} = {Analog Input 1 (Value) :neo0-ai0-0-index(0)}	// copy ain 1 to can signal 1
3	Set Value	{can_sig_ain_2 (Value) :out0-sig1-0} = {Analog Input 2 (Value) :neo0-ai1-0-index(0)}	// copy ain 2 to can signal 2
4	Set Value	{can_sig_ain_3 (Value) :out0-sig2-0} = {Analog Input 3 (Value) :neo0-ai2-0-index(0)}	// copy ain 3 to can signal 3
5	Set Value	{can_sig_ain_4 (Value) :out0-sig3-0} = {Analog Input 4 (Value) :neo0-ai3-0-index(0)}	// copy ain 4 to can signal 4
6	Transmit	ain_status (out0)	
7	Wait For	= 10 ms	

Optionally, if you used a variable and read the analog input port into a variable then you would assign the variable to the appropriate CAN signal as shown below.

7	Set Value	{can_sig_ain_1 (Value) :out0-sig0-0} = {app_sig_ain_1 (Value) :sig0-0}	// copy app sig 1 to can signal 1
8	Set Value	{can_sig_ain_2 (Value) :out0-sig1-0} = {app_sig_ain_2 (Value) :sig1-0}	// copy app sig 2 to can signal 2
9	Set Value	{can_sig_ain_3 (Value) :out0-sig2-0} = {app_sig_ain_3 (Value) :sig2-0}	// copy app sig 3 to can signal 3
10	Set Value	{can_sig_ain_4 (Value) :out0-sig3-0} = {app_sig_ain_4 (Value) :sig3-0}	// copy app sig 4 to can signal 4
11	Transmit	ain_status (out0)	
12	Wait For	= 10 ms	

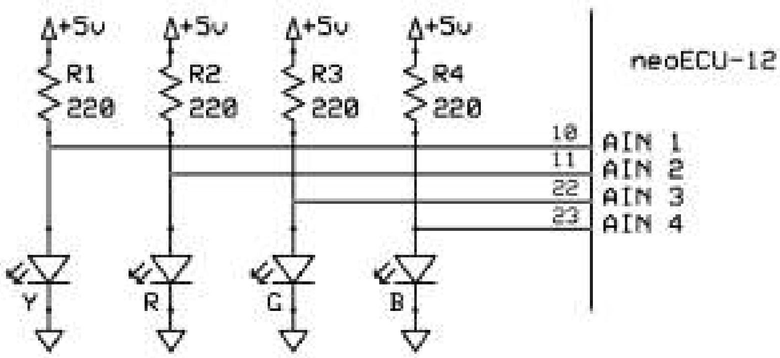
Program CoreMini with this script. Connect another Intrepid tool to the neoECU-12 and monitor the HS CAN bus. You should see the status of all four analog input ports as shown in the example below.

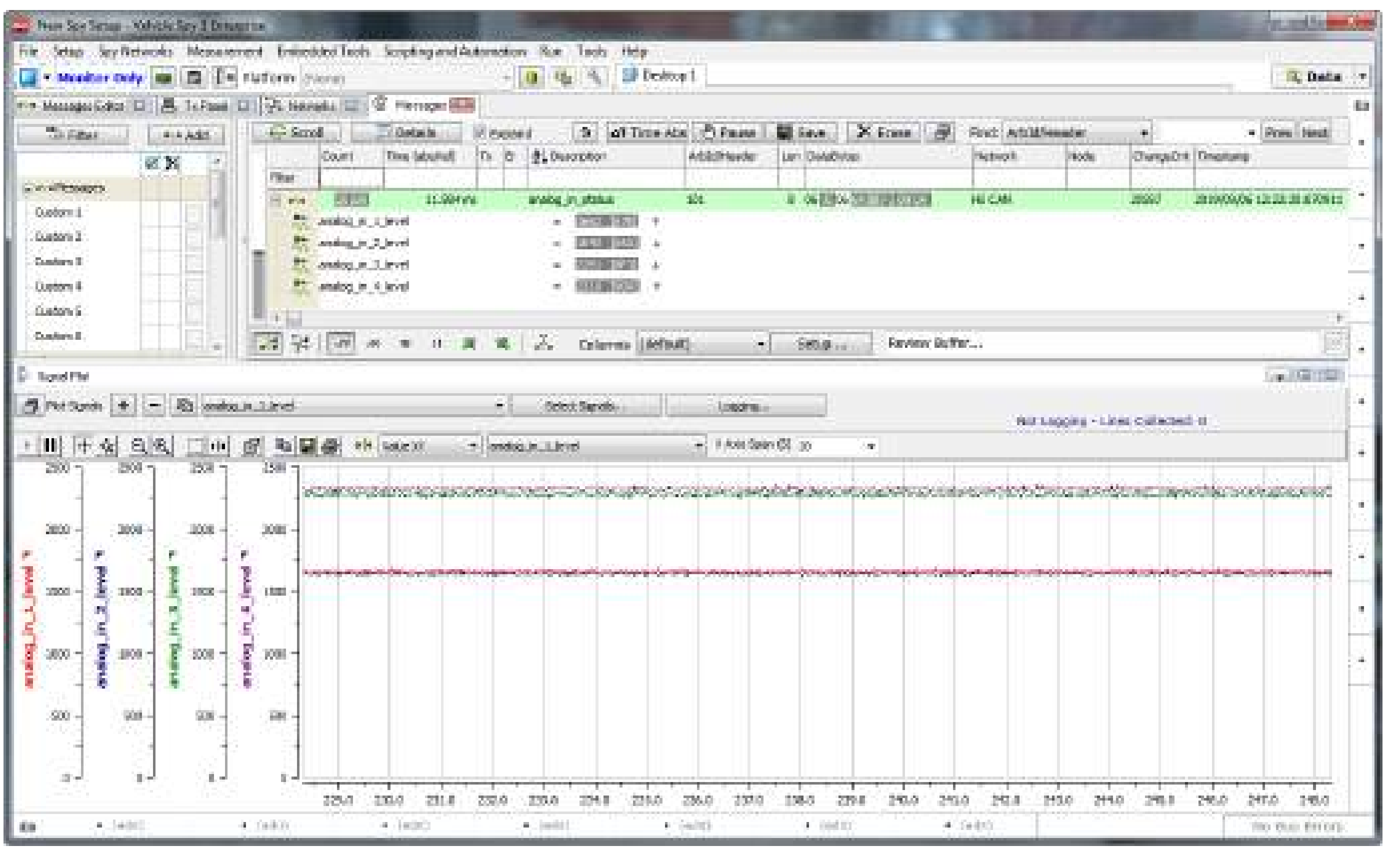
	Count	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes	Network
Filter									
	77128	11.992 ms			analog_in_status	101	8	06 7A 06 6B 08 D3 09 10	HS CAN
	analog_in_1_level				= 1658 [67A]				
	analog_in_2_level				= 1643 [66B]				
	analog_in_3_level				= 2259 [8D3]				
	analog_in_4_level				= 2320 [910]				

The circuit that was used in the example was four LEDs connected to each AIN port. A yellow LED was connected to AIN 1, a red LED connected to AIN 2, a green LED connected to AIN 3, and a blue LED connected to AIN 4. From the signal plot you can see the forward voltage drop on the yellow and red LED are about the same and the forward voltage drop on the green and blue are about the same. To calculate the forward voltage drop of the LED take the measured counts, divide by 4096 (for 12-bit A/D) then multiply by 5 (for 5V maximum voltage).

LED	calculation
yellow	1658 / 4096 * 5 = 2.02 V
red	1642 / 4096 * 5 = 2.00 V
green	2259 / 4096 * 5 = 2.76 V
blue	2320 / 4096 * 5 = 2.83 V

Test circuit used:





5.7 Controlling the Tri-Color LEDs

The neoECU-12 has default behavior for the first two tri-color LEDs. The 1st LED is for HS CAN 1 and the 2nd LED is for HS CAN 2.

Interpretation of RGB LED Colors

These are "RGB" LEDs because they contain separate red, green and blue elements. For networks, each indicates a different aspect of the device's overall status:

- **Green:** Device is transmitting messages on this channel.
- **Blue:** Device is receiving messages on this channel.
- **Red:** Device is detecting errors on this channel.

It is possible for more than one LED component to be lit, producing the following results:

- **Green+Blue (Cyan):** Device is transmitting and receiving on this channel.
- **Green+Red (Yellow):** Device is transmitting and detecting errors on this channel.
- **Blue+Red (Magenta):** Device is receiving and detecting errors on this channel.
- **Green+Blue+Red (White):** Device is transmitting, receiving and detecting errors on this channel.

LED3 blinks magenta when a function block script is running.

In bootloader mode, all five LEDs blink red in succession.

To override the default behavior, use the command “Set Value” and set “LED x (Auto) = 0” as shown below. To restore to default set “LED x (Auto) = 1”. To set the color of the LED set the red/green/blue property of the LED to 255 as shown below. 255 is the maximum intensity. A value of 50 would be a very low intensity. You can also mix the red/green/blue property to create additional colors.

Script | Start | Notes — LED Control

After | Before — No Errors

Step	Description	Value	Comment
1	// TODO: Add step commands here		
2	Set Value	{LED 1 (Auto) :neo0-ld0-3-index(0)} = 0	// user control of LED 1
3	Set Value	{LED 2 (Auto) :neo0-ld1-3-index(0)} = 0	// user control of LED 2
4	Set Value	{LED 3 (Auto) :neo0-ld2-3-index(0)} = 0	// user control of LED 3
5	Set Value	{LED 4 (Auto) :neo0-ld3-3-index(0)} = 0	// user control of LED 4
6	Set Value	{LED 5 (Auto) :neo0-ld4-3-index(0)} = 0	// user control of LED 5
7			
8	Set Value	{LED 1 (Green) :neo0-ld0-5-index(0)} = 255	// set LED 1 to green
9	Set Value	{LED 2 (Green) :neo0-ld1-5-index(0)} = 255	// set LED 2 to green
10	Set Value	{LED 3 (Green) :neo0-ld2-5-index(0)} = 255	// set LED 3 to green
11	Set Value	{LED 4 (Green) :neo0-ld3-5-index(0)} = 255	// set LED 4 to green
12	Set Value	{LED 5 (Green) :neo0-ld4-5-index(0)} = 255	// set LED 5 to green
13			

The proper way to set the LED color is to assign values to the red/green/blue property. This example shows how to set LED 1 to green color.

8	Set Value	{LED 1 (Red) :neo0-ld0-4-index(0)} = 0	// set LED 1 to green
9	Set Value	{LED 1 (Green) :neo0-ld0-5-index(0)} = 255	// set LED 1 to green
10	Set Value	{LED 1 (Blue) :neo0-ld0-6-index(0)} = 0	// set LED 1 to green

This example shows how to set LED 1 to yellow color.

8	Set Value	{LED 1 (Red) :neo0-ld0-4-index(0)} = 255	// set LED 1 to yellow
9	Set Value	{LED 1 (Green) :neo0-ld0-5-index(0)} = 255	// set LED 1 to yellow
10	Set Value	{LED 1 (Blue) :neo0-ld0-6-index(0)} = 0	// set LED 1 to yellow

This example shows how to set LED 1 to purple color.

8	Set Value	{LED 1 (Red) :neo0-ld0-4-index(0)} = 255	// set LED 1 to purple
9	Set Value	{LED 1 (Green) :neo0-ld0-5-index(0)} = 0	// set LED 1 to purple
10	Set Value	{LED 1 (Blue) :neo0-ld0-6-index(0)} = 255	// set LED 1 to purple

This example shows how to set LED 1 to cyan color.

8	Set Value	{LED 1 (Red) :neo0-ld0-4-index(0)} = 0	// set LED 1 to cyan
9	Set Value	{LED 1 (Green) :neo0-ld0-5-index(0)} = 255	// set LED 1 to cyan
10	Set Value	{LED 1 (Blue) :neo0-ld0-6-index(0)} = 255	// set LED 1 to cyan

This example shows how to set LED 1 to white color. (note even though the LED is white for the most part you can see the red filament of the LED turn on)

8	Set Value	{LED 1 (Red) :neo0-ld0-4-index(0)} = 255	// set LED 1 to white
9	Set Value	{LED 1 (Green) :neo0-ld0-5-index(0)} = 255	// set LED 1 to white
10	Set Value	{LED 1 (Blue) :neo0-ld0-6-index(0)} = 255	// set LED 1 to white

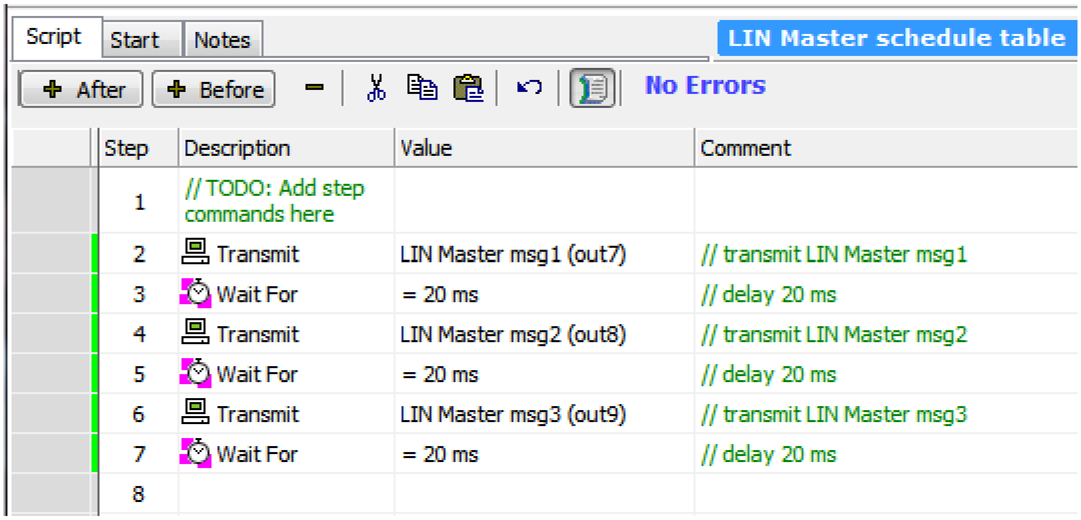
This example shows how to set LED 1 to orange color. Note the intensity changes from 255 to 100 on the green property.

8	Set Value	{LED 1 (Red) :neo0-ld0-4-index(0)} = 255	// set LED 1 to orange
9	Set Value	{LED 1 (Green) :neo0-ld0-5-index(0)} = 100	// set LED 1 to orange
10	Set Value	{LED 1 (Blue) :neo0-ld0-6-index(0)} = 0	// set LED 1 to orange

Recommendation: Since LED 1 thru LED 3 are predefined start with using LED 4 and LED 5 for user control.

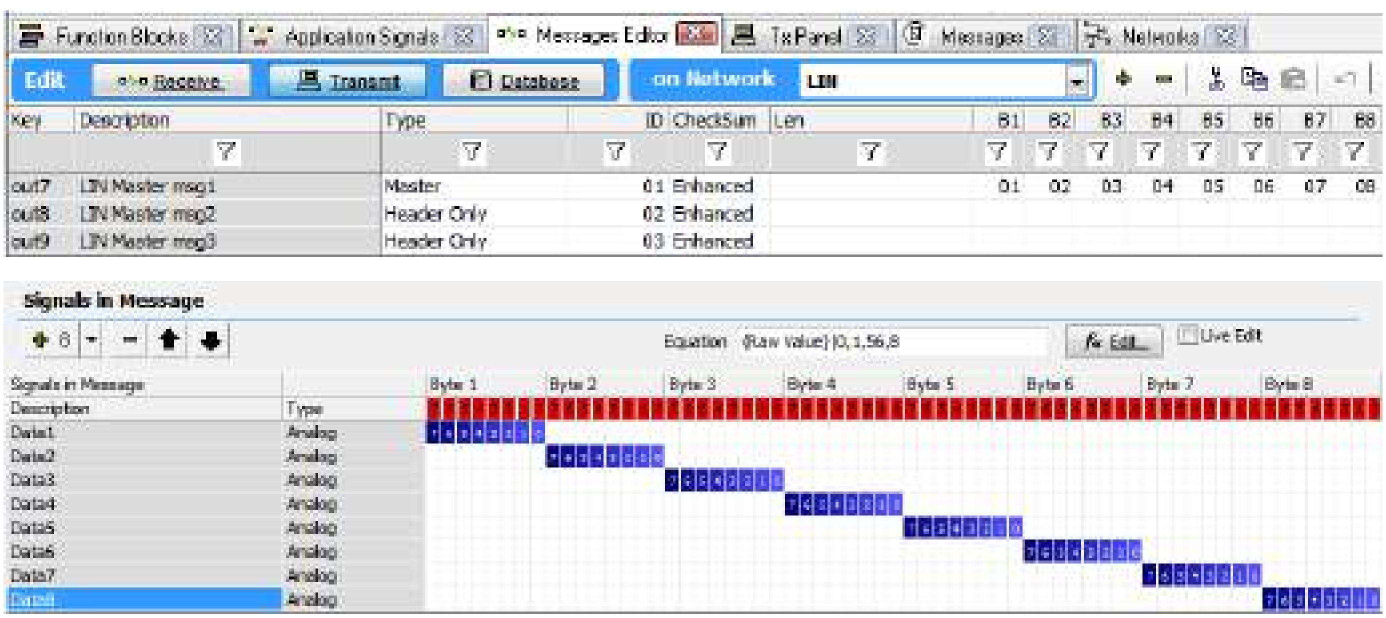
5.8 LIN example

Below is a simple LIN Master schedule table. Each message is transmitted with 20 ms delay between messages.



Step	Description	Value	Comment
1	// TODO: Add step commands here		
2	Transmit	LIN Master msg1 (out7)	// transmit LIN Master msg1
3	Wait For	= 20 ms	// delay 20 ms
4	Transmit	LIN Master msg2 (out8)	// transmit LIN Master msg2
5	Wait For	= 20 ms	// delay 20 ms
6	Transmit	LIN Master msg3 (out9)	// transmit LIN Master msg3
7	Wait For	= 20 ms	// delay 20 ms
8			

In this example, three LIN Transmit messages were created. "LIN Master msg1" is a message that contains data that will be sent to the LIN slaves. The signals were pre-defined as "01" thru "08" but can be modified in the script. The signals in "LIN Master msg1" were named "Data1" thru "Data8" as shown below. Note the Type is "Master". For LIN messages that will contain data from the slaves the Type needs to be set to "Header Only".



Key	Description	Type	ID	CheckSum	Len	B1	B2	B3	B4	B5	B6	B7	B8
out7	LIN Master msg1	Master	01	Enhanced		01	02	03	04	05	06	07	08
out8	LIN Master msg2	Header Only	02	Enhanced									
out9	LIN Master msg3	Header Only	03	Enhanced									

Description	Type
Data1	Analog
Data2	Analog
Data3	Analog
Data4	Analog
Data5	Analog
Data6	Analog
Data7	Analog
Data8	Analog

If we program this example into the neoECU-12 and then use another Intrepid device to monitor the LIN Bus these are the messages on the LIN Bus. Note there is an error for LIN message "LIN 02" and "LIN 03". This is because the monitoring or slave device has not sent back any data to the LIN Master (neoECU-12).

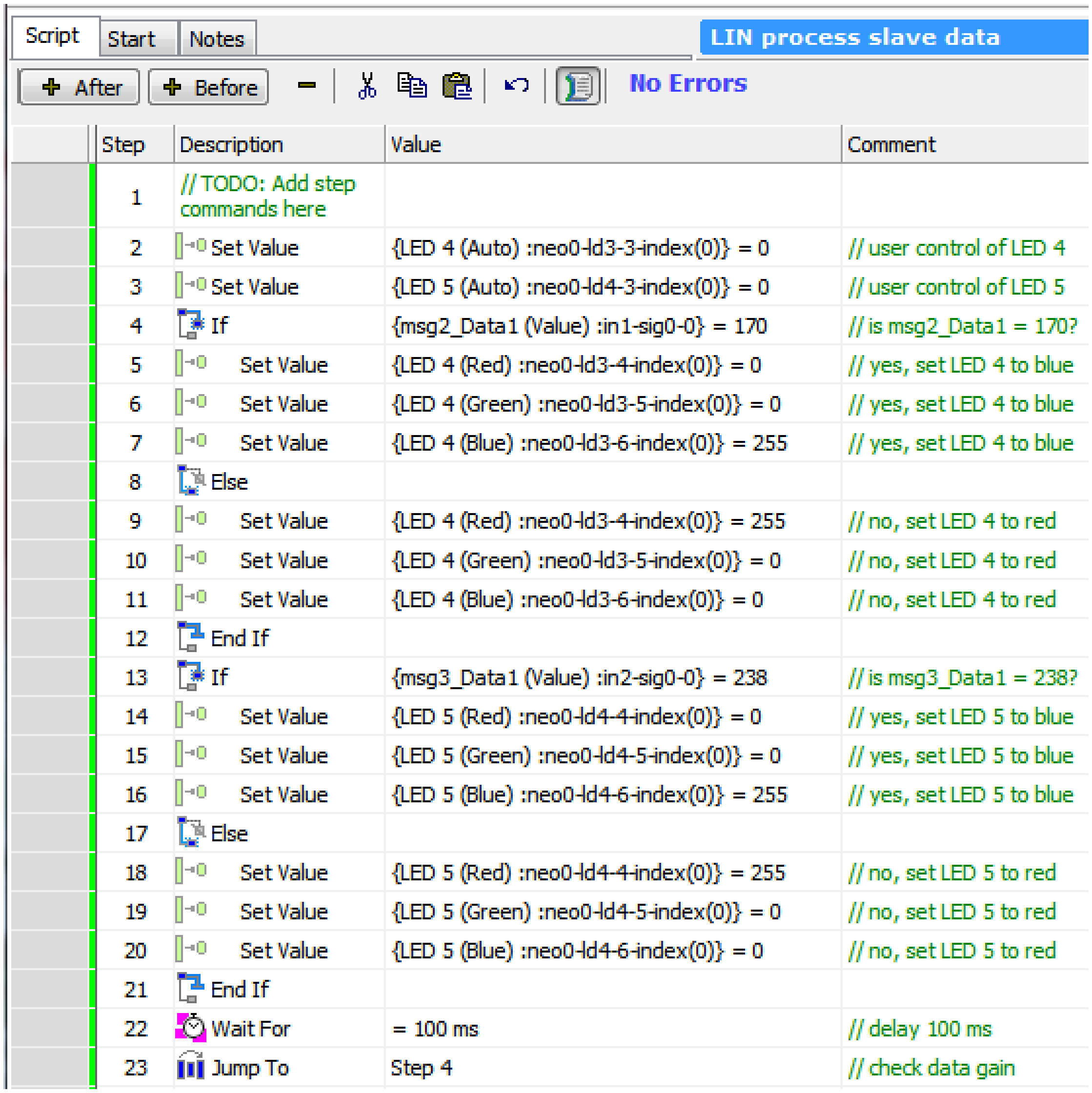
	Count	Time (abs/rel)	TxN	TxS	Er	Description	ArbId/Header	DataBytes	ChkSum	TFrame	SlvStatus	Network
Filter												
	[illegible]	60.970 ms				LIN Master msg1	01 (0xC1)	01 02 03 04 05 06 07 08	1A (Enh)	0 uS	Ok	LIN
	Data1					= 1 [1]						
	Data2					= 2 [2]						
	Data3					= 3 [3]						
	Data4					= 4 [4]						
	Data5					= 5 [5]						
	Data6					= 6 [6]						
	Data7					= 7 [7]						
	Data8					= 8 [8]						
	[illegible]	60.971 ms				LIN 03; No Slave Data	03 (0x03)		00 (Error)	0 uS		LIN
	[illegible]	61.066 ms				LIN 02; No Slave Data	02 (0x42)		00 (Error)	0 uS		LIN

Next have the monitoring or slave device add data to "LIN Master msg2" and "LIN Master msg3". Note the data can be any length from 1 to 8 bytes. It is now up to the LIN Master to process the data bytes sent back from the LIN Slave.

	Count	Time (abs/rel)	TxN	TxS	Er	Description	ArbId/Header	DataBytes	ChkSum	TFrame	SlvStatus	Network
Filter												
	[illegible]	60.970 ms				LIN Master msg1	01 (0xC1)	01 02 03 04 05 06 07 08	1A (Enh)	0 uS	Ok	LIN
	Data1					= 1 [1]						
	Data2					= 2 [2]						
	Data3					= 3 [3]						
	Data4					= 4 [4]						
	Data5					= 5 [5]						
	Data6					= 6 [6]						
	Data7					= 7 [7]						
	Data8					= 8 [8]						
	[illegible]	60.969 ms				LIN Master msg2	02 (0x42)	AA BB CC DD	AC (Enh)	0 uS	ChkSum	LIN
	msg2_Data1					= 170 [AA]						
	msg2_Data2					= 187 [BB]						
	msg2_Data3					= 204 [CC]						
	msg2_Data4					= 221 [DD]						
	[illegible]	61.066 ms				LIN Master msg3	03 (0x03)	EE FF	0E (Enh)	0 uS	ID Parity	LIN
	msg3_Data1					= 238 [EE]						
	msg3_Data2					= 255 [FF]						

To see the LIN columns in the message view select LIN from the Columns button at the bottom of the messages view.

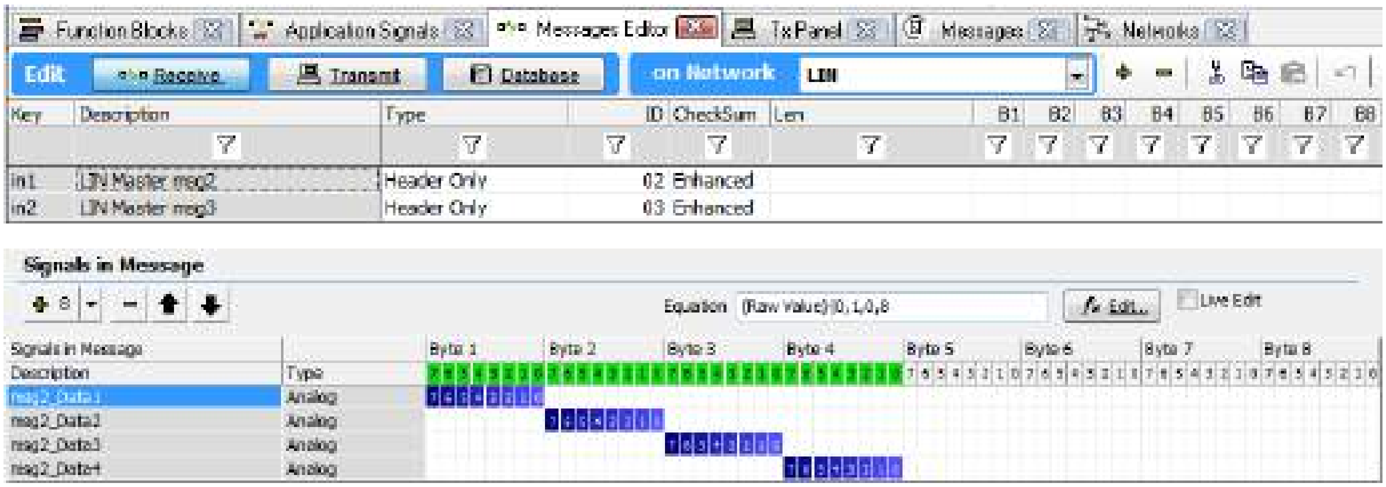
To test the LIN Master is receiving the LIN Slave data a simple script was written to control LED 4 and LED 5 based on receipt of the LIN Slave data. If the “msg2_Data1” = 170 then LED 4 is blue otherwise it is red. If the “mg3_Data1” = 238 the LED 5 is blue otherwise it is red.



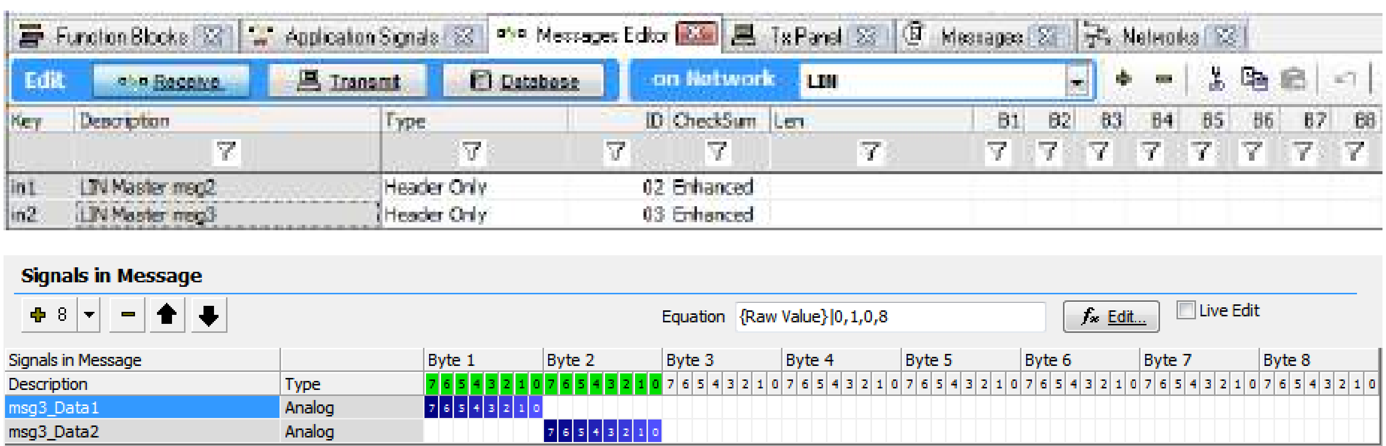
Step	Description	Value	Comment
1	// TODO: Add step commands here		
2	Set Value	{LED 4 (Auto) :neo0-ld3-3-index(0)} = 0	// user control of LED 4
3	Set Value	{LED 5 (Auto) :neo0-ld4-3-index(0)} = 0	// user control of LED 5
4	If	{msg2_Data1 (Value) :in1-sig0-0} = 170	// is msg2_Data1 = 170?
5	Set Value	{LED 4 (Red) :neo0-ld3-4-index(0)} = 0	// yes, set LED 4 to blue
6	Set Value	{LED 4 (Green) :neo0-ld3-5-index(0)} = 0	// yes, set LED 4 to blue
7	Set Value	{LED 4 (Blue) :neo0-ld3-6-index(0)} = 255	// yes, set LED 4 to blue
8	Else		
9	Set Value	{LED 4 (Red) :neo0-ld3-4-index(0)} = 255	// no, set LED 4 to red
10	Set Value	{LED 4 (Green) :neo0-ld3-5-index(0)} = 0	// no, set LED 4 to red
11	Set Value	{LED 4 (Blue) :neo0-ld3-6-index(0)} = 0	// no, set LED 4 to red
12	End If		
13	If	{msg3_Data1 (Value) :in2-sig0-0} = 238	// is msg3_Data1 = 238?
14	Set Value	{LED 5 (Red) :neo0-ld4-4-index(0)} = 0	// yes, set LED 5 to blue
15	Set Value	{LED 5 (Green) :neo0-ld4-5-index(0)} = 0	// yes, set LED 5 to blue
16	Set Value	{LED 5 (Blue) :neo0-ld4-6-index(0)} = 255	// yes, set LED 5 to blue
17	Else		
18	Set Value	{LED 5 (Red) :neo0-ld4-4-index(0)} = 255	// no, set LED 5 to red
19	Set Value	{LED 5 (Green) :neo0-ld4-5-index(0)} = 0	// no, set LED 5 to red
20	Set Value	{LED 5 (Blue) :neo0-ld4-6-index(0)} = 0	// no, set LED 5 to red
21	End If		
22	Wait For	= 100 ms	// delay 100 ms
23	Jump To	Step 4	// check data gain

Also note for this example the "LIN Master msg2" and "LIN Master msg3" were copied from the Transmit table and added to the Receive table. Then "msg2_Data1" thru "msg2_Data4" signals and "msg3_Data1" thru "msg3_Data2" signals were added to their respective messages. The signals needed to be defined so they could be processed in the script.

Receive table – "LIN Master msg2" and signals



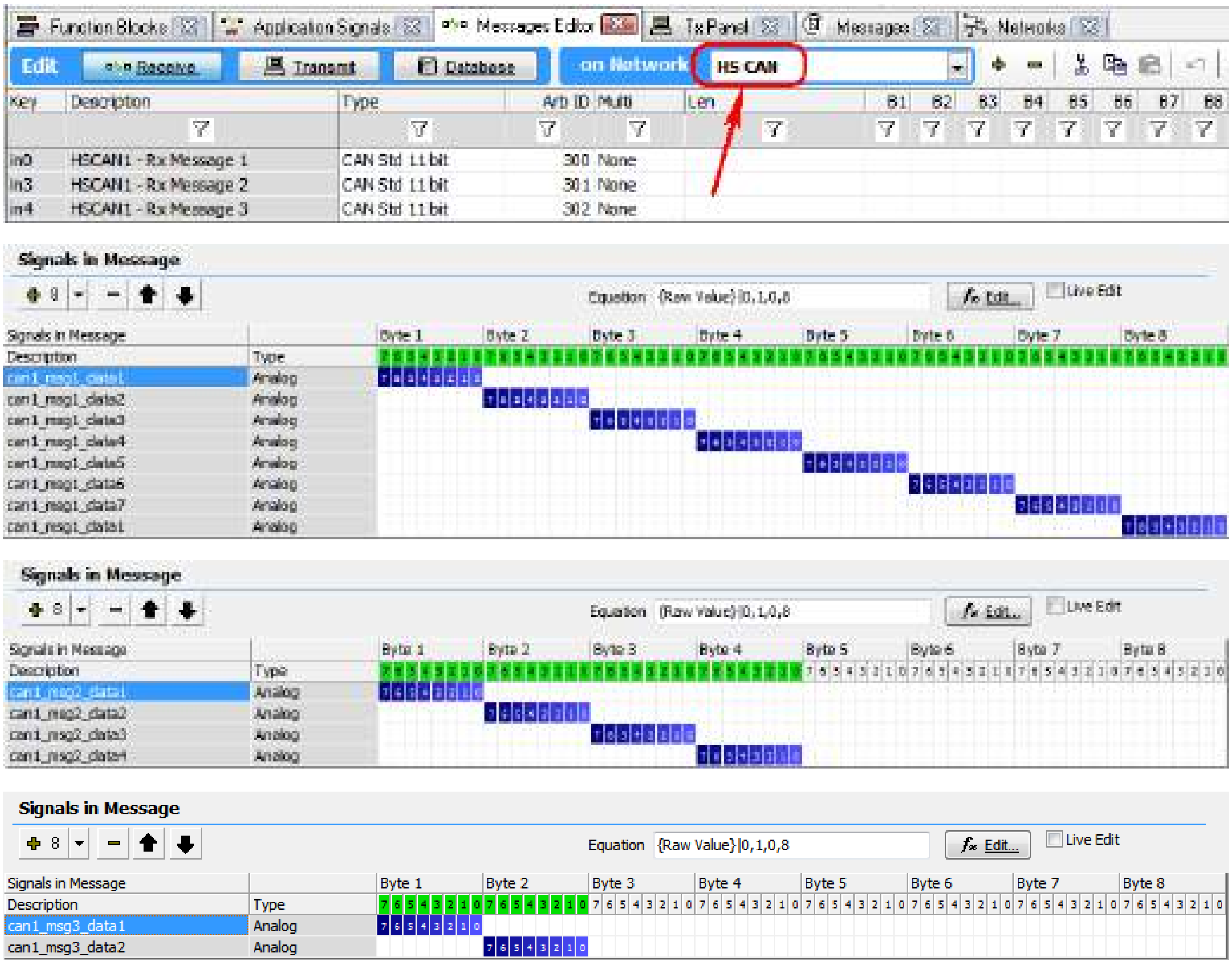
Receive table – "LIN Master msg3" and signals



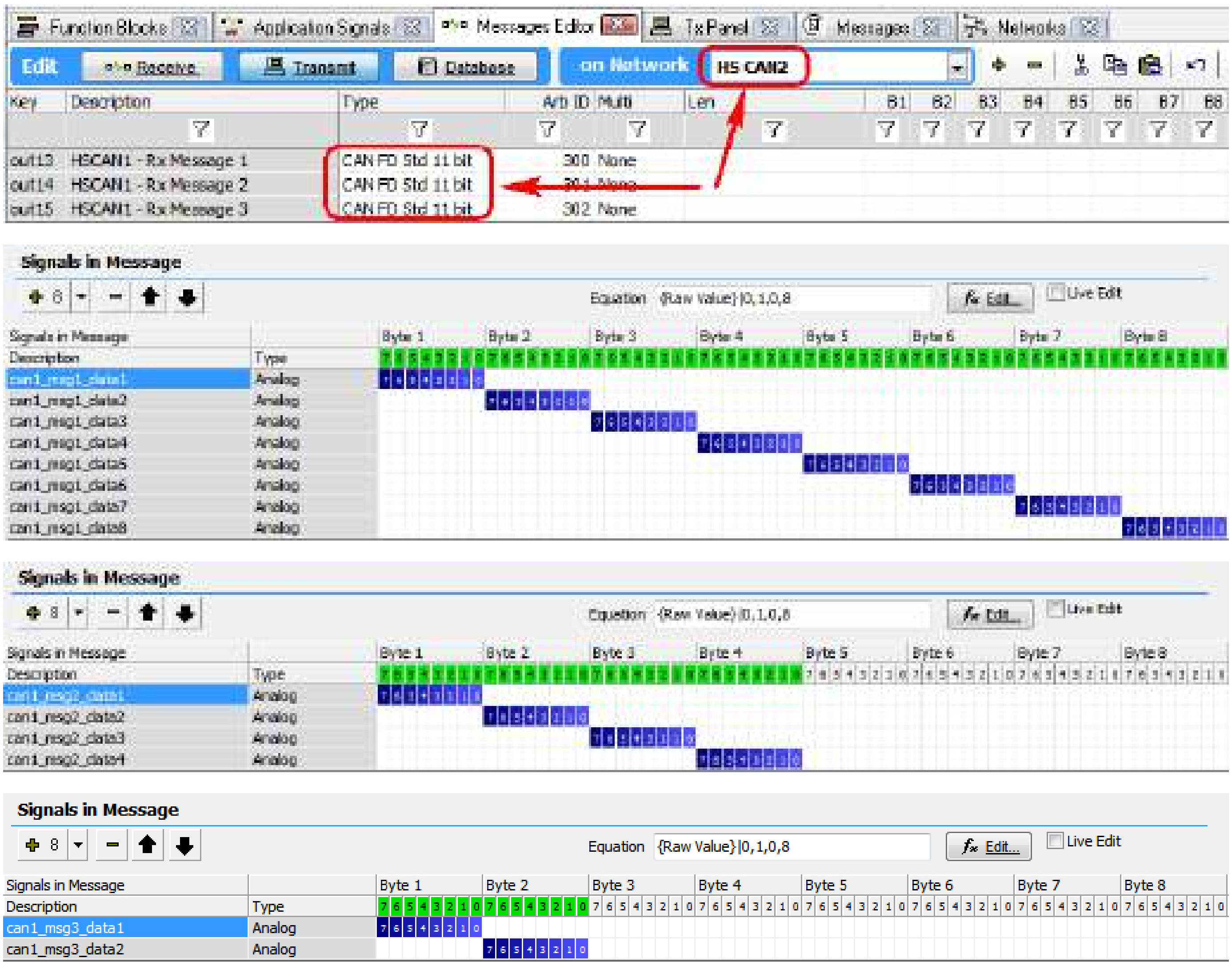
5.9 CAN to CAN-FD Gateway example

In this example, HS CAN 1 will be configured for standard CAN 2.0B. HS CAN 2 will be configured for HS CAN 2. The baud rate will be set to 500 Kbps and 2000 Kbps, respectively.

Create 3 HS CAN 1 Receive messages and their respective signals as shown below.



Create 3 HS CAN 2 Transmit messages and their respective signals as shown below. Hint: go to the HS CAN 1 Receive table and copy the 3 messages. Then go to the HS CAN 2 Transmit table and paste the messages. Paste into the Description column. This will copy over the signals as well! Change the Type to "CAN FD Std 11 bit" for all 3 messages.



Next, create a script to copy the signals from the Receive messages into the Transmit messages.

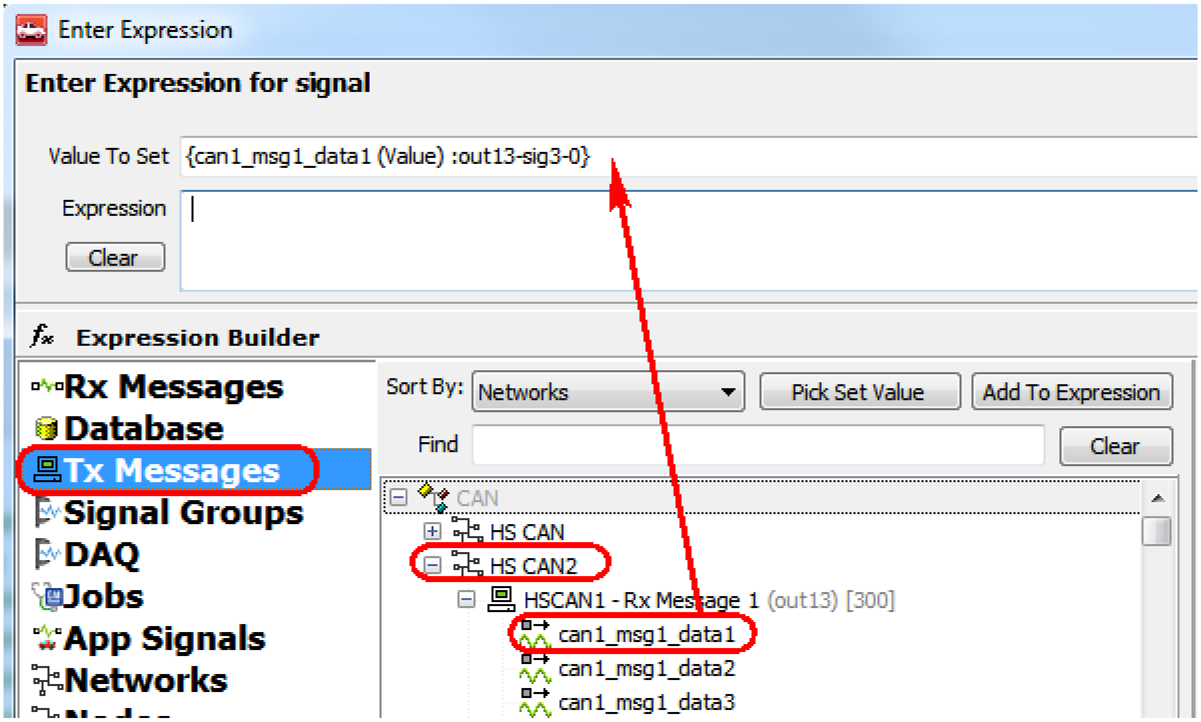
1	// TODO: Add step commands here	
2	If	{HSCAN1 - Rx Message 1 (Present) :in0-0}
3	Set Value	{HSCAN1 - Rx Message 1 (Present) :in0-0} = 0
4	Set Value	{can1_msg1_data1 (Value) :out13-sig3-0} = {can1_msg1_data1 (Value) :in0-sig3-0}
5	Set Value	{can1_msg1_data2 (Value) :out13-sig4-0} = {can1_msg1_data2 (Value) :in0-sig4-0}
6	Set Value	{can1_msg1_data3 (Value) :out13-sig5-0} = {can1_msg1_data3 (Value) :in0-sig5-0}
7	Set Value	{can1_msg1_data4 (Value) :out13-sig6-0} = {can1_msg1_data4 (Value) :in0-sig6-0}
8	Set Value	{can1_msg1_data5 (Value) :out13-sig7-0} = {can1_msg1_data5 (Value) :in0-sig7-0}
9	Set Value	{can1_msg1_data6 (Value) :out13-sig8-0} = {can1_msg1_data6 (Value) :in0-sig8-0}
10	Set Value	{can1_msg1_data7 (Value) :out13-sig9-0} = {can1_msg1_data7 (Value) :in0-sig9-0}
11	Set Value	{can1_msg1_data8 (Value) :out13-sig10-0} = {can1_msg1_data8 (Value) :in0-sig10-0}
12	Transmit	HSCAN1 - Rx Message 1 (out13)
13	End If	
14	If	{HSCAN1 - Rx Message 2 (Present) :in3-0}
15	Set Value	{HSCAN1 - Rx Message 2 (Present) :in3-0} = 0
16	Set Value	{can1_msg2_data1 (Value) :out14-sig0-0} = {can1_msg2_data1 (Value) :in3-sig0-0}
17	Set Value	{can1_msg2_data2 (Value) :out14-sig1-0} = {can1_msg2_data2 (Value) :in3-sig1-0}
18	Set Value	{can1_msg2_data3 (Value) :out14-sig2-0} = {can1_msg2_data3 (Value) :in3-sig2-0}
19	Set Value	{can1_msg2_data4 (Value) :out14-sig3-0} = {can1_msg2_data4 (Value) :in3-sig3-0}
20	Transmit	HSCAN1 - Rx Message 2 (out14)
21	End If	
22	If	{HSCAN1 - Rx Message 3 (Present) :in4-0}
23	Set Value	{HSCAN1 - Rx Message 3 (Present) :in4-0} = 0
24	Set Value	{can1_msg3_data1 (Value) :out15-sig0-0} = {can1_msg3_data1 (Value) :in4-sig0-0}
25	Set Value	{can1_msg3_data2 (Value) :out15-sig1-0} = {can1_msg3_data2 (Value) :in4-sig1-0}
26	Transmit	HSCAN1 - Rx Message 3 (out15)
27	End If	

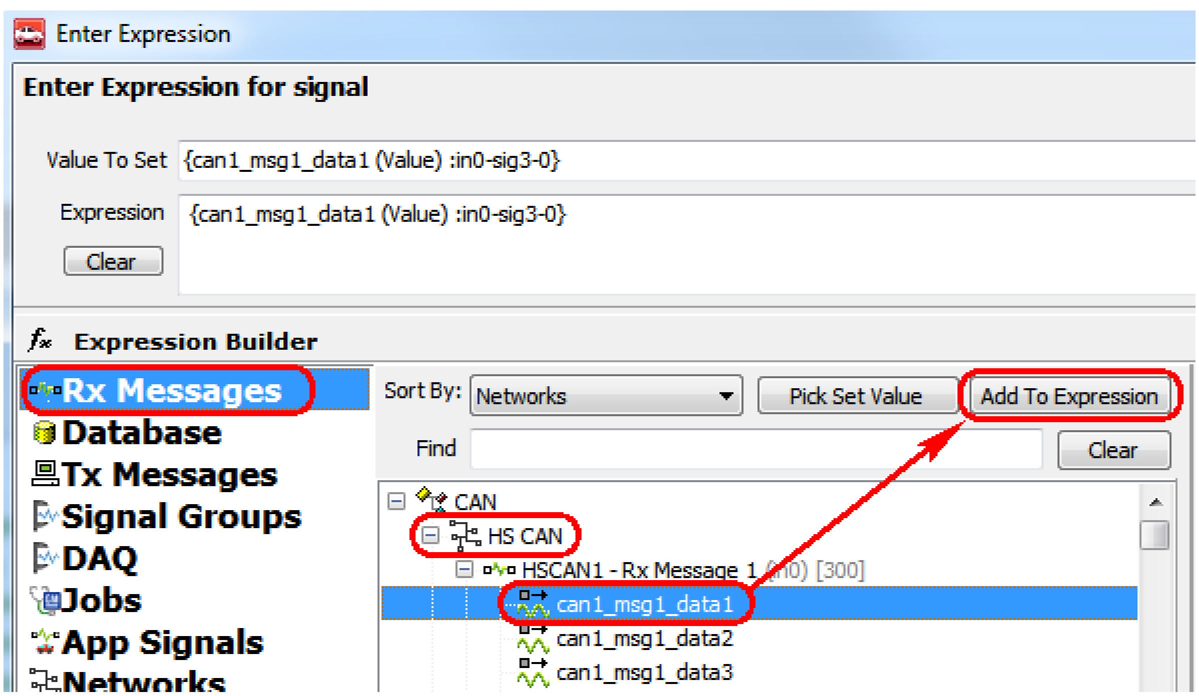
The following screenshot shows the comments for each script line item …

	// is HSCAN1 - Rx Message 1 present?
	// yes, clear present flag
:in0-sig3-0}	// copy byte 1 from HSCAN1 rx message to HSCAN2 tx message
:in0-sig4-0}	// copy byte 2 from HSCAN1 rx message to HSCAN2 tx message
:in0-sig5-0}	// copy byte 3 from HSCAN1 rx message to HSCAN2 tx message
:in0-sig6-0}	// copy byte 4 from HSCAN1 rx message to HSCAN2 tx message
:in0-sig7-0}	// copy byte 5 from HSCAN1 rx message to HSCAN2 tx message
:in0-sig8-0}	// copy byte 6 from HSCAN1 rx message to HSCAN2 tx message
:in0-sig9-0}	// copy byte 7 from HSCAN1 rx message to HSCAN2 tx message
:) :in0-sig10-0}	// copy byte 8 from HSCAN1 rx message to HSCAN2 tx message
	// transmit HSCAN2 message 1
	// is HSCAN1 - Rx Message 2 present?
	// yes, clear present flag
:in3-sig0-0}	// copy byte 1 from HSCAN1 rx message to HSCAN2 tx message
:in3-sig1-0}	// copy byte 2 from HSCAN1 rx message to HSCAN2 tx message
:in3-sig2-0}	// copy byte 3 from HSCAN1 rx message to HSCAN2 tx message
:in3-sig3-0}	// copy byte 4 from HSCAN1 rx message to HACAN2 tx message
	// transmit HSCAN2 message 2
	// is HSCAN1 - Rx Message 3 present?
	// yes, clear present flag
:in4-sig0-0}	// copy byte 1 from HSCAN1 rx message to HSCAN2 tx message
:in4-sig1-0}	// copy byte 2 from HSCAN1 rx message to HSCAN2 tx message
	// transmit HSCAN2 message 3

When the script is complete program it into the neoECU-12.

When creating the script make sure the Tx Message is selected followed by HS CAN2 followed by the 1st data byte in message 1. Next, make sure the Rx Messages is selected followed by HS CAN followed by the 1st data byte in message 1 then click "Add to Expression" button. Repeat for all signals in all 3 messages.





Now use another Intrepid tool that is CAN/CAN-FD capable to generate CAN traffic on HS CAN 1. Transmit 3 CAN messages with ArbID 300, 301, and 302. The 1st message should contain 8 data bytes, the 2nd message should contain 4 data bytes, and the 3rd message should contain 2 data bytes.

Key	Description	Type	Arb ID	Multi	Len	B1	B2	B3	B4	B5	B6	B7	B8
out2	HSCAN1 - Tx Message 1	CAN Std 11 bit	300	None		01	02	03	04	05	06	07	08
out3	HSCAN1 - Tx Message 2	CAN Std 11 bit	301	None		AA	BB	CC	DD				
out4	HSCAN1 - Tx Message 3	CAN Std 11 bit	302	None		EE	FF						

In the Tx Panel set the rate to be 100 ms and periodic. This is a test only. The ECU will have it's own rates for each message or you can duplicate the rates here.

Description	Tx	Auto Tx	Rate (s)	Arb ID	Len	B1	B2	B3	B4	B5	B6	B7	B8	Network	Color
HSCAN1 - Tx Message 1		Periodic	0.100000	300		01	02	03	04	05	06	07	08	HS CAN	
HSCAN1 - Tx Message 2		Periodic	0.100000	301		AA	BB	CC	DD					HS CAN	
HSCAN1 - Tx Message 3		Periodic	0.100000	302		EE	FF							HS CAN	

Next transmit these messages and monitor both HS CAN 1 and HS CAN 2. Each time the message was received on HS CAN 1 it was forwarded to HS CAN 2. In this case, the same ArbID and all of the data bytes were forwarded as is and not changed or scaled. Note the FDF bit (CAN-FD) bit is set and the BRS bit (Bit Rate Switch) is set. If you were to put a scope on the HS CAN 2 line you would see the data bytes are transmitted at the higher bit rate. To see the CAN-FD columns in the message view select CAN FD from the Columns button at the bottom of the messages view.

	Count	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes	Network	Node	DLC	FDF	BRS	ESI
	144	102.986 ms			HS CAN2 $300	300	8	01 02 03 04 05 06 07 08	HS CAN2		8	1	1	0
	144	102.994 ms			HS CAN2 $301	301	4	AA BB CC DD	HS CAN2		4	1	1	0
	144	102.985 ms			HS CAN2 $302	302	2	EE FF	HS CAN2		2	1	1	0
	144	102.930 ms			HSCAN1 - Tx Message 1	300	8	01 02 03 04 05 06 07 08	HS CAN		8			
	144	102.931 ms			HSCAN1 - Tx Message 2	301	4	AA BB CC DD	HS CAN		4			
	144	102.931 ms			HSCAN1 - Tx Message 3	302	2	EE FF	HS CAN		2			

Check out our new Gateway Builder in Vehicle Spy Enterprise version. Drag and Drop GUI makes it super easy to build a custom Gateway! It generates the scripts for you. 😉

6.0 Support Contact Information

6.1 ICS United States Headquarters and West Coast office

Intrepid Control Systems, Inc.
31601 Research Park Drive
Madison Heights, MI 48071

Phone: (800) 859-6265 or (586) 731-7950
Fax: (586) 731-2274
Email: Email ICS

Intrepid Control Systems, Inc.
1925 Winchester Blvd, Ste 200
Campbell, CA 95008 USA

Phone: 1-800-859-6265 or (586) 731-7950
Fax: 586-731-2274
Email: Email Sales

6.3 ICS International Offices

European Union

Haid-und-Neu-Straße 7
76131 Karlsruhe
Germany
Phone: +49 721 1803083-0
Fax: +49 721 1803083 -9
Email: Email Sales

United Kingdom

Serviced Office Suite 1.03
MIRA Technology Park
Technology Centre NW05
Watling Street
Nuneaton
Warwickshire
CV10 0TU
Phone: +44 24 7718 0296
Email: Email Sales

China (Beijing)

https://www.intrepidcs.net.cn

Room 214, No.45 Chengfu Road,
Zhongguancun Zhizao Street
Block C, Haidian District,
Beijing P.R.China 100083
Phone: +86 176 8014 3205
Email: Email Sales

China (Shanghai)

https://www.intrepidcs.net.cn

Room 902, Building 16
No. 1000 Jinhai Road
City of Elite
Pudong, Shanghai
P.R.China 201206
Phone: 021-61637366
Fax: 021-61637366 * 600
Email: Email Sales

China (Shenzhen)

https://www.intrepidcs.net.cn

Room 22-YZ, Block A,
Che Kung Temple Fortune Plaza
No.5 Xianglin Road
Shenzhen, 518040 CHINA
Phone: +86 0755 82723212
Email: Email Sales

India

Office 306, B Building, Third Floor,
GO Square,
Wakad,
Pune, MH, 411057 India
Phone:
Sales: +91 77 55 99 00 70
Tech Support: +91 77 55 99 00 64
All Others: +91 77 55 99 00 74
Email: Email Sales

South Korea

http://www.intrepidcs.co.kr/

#1310, Keurancheu Techno B/D,
388, Dunchon-daero, Jungwon-gu,
Seongnam-si, Gyeonggi-do, 13403,
South Korea
Phone/전화: +82 31 698 3460
Fax/팩스: +82 31 698 3461
Email: Sales

Japan

http://www.intrepidcs.jp

164-0003
Shinryou Building 3F
Higashinakano 1-59-6
Nakano-Ku: Tokyo
電話番号: +81 03-5937-1523
ファックス: +81 03-5937-2524
営業 / 一般的なお問い合わせ: icsjapan@intrepidcs.com
技術サポート: icsjapansupport@intrepidcs.com

Australia

67A Hardiman St
Kensington, VIC 3031
Australia
Phone: 03 9466 4948
(International callers: +61 3 9466 4948)
Email: Julian Merritt

