Manual

Model **1351**

System Controller

» Software Device Profile 2.11.14.0 «

Curtis Instruments, Inc.
200 Kisco Avenue
Mt. Kisco, NY 10549
www.curtisinstruments.com

Read Instructions Carefully!

Specifications are subject to change without notice.
© 2024 Curtis Instruments, Inc. ® Curtis is a registered trademark of Curtis Instruments, Inc.
© The design and appearance of the products depicted herein are the copyright of Curtis Instruments, Inc.

53227 Rev F Feb. 2024

TABLE OF CONTENTS

CHAPTERS

1: OVERVIEW 1

HOW TO USE THIS MANUAL 1

GETTING THE MOST OUT OF YOUR CURTIS CONTROLLER 2

2: INSTALLATION SPECIFICATIONS AND WIRING 3

PHYSICALLY MOUNTING THE 1351 SYSTEM CONTROLLER 3

PRECAUTIONS 5

THE 35-PIN AMPSEAL CONNECTIONS 6

BATTERY CONNECTIONS (B+, B–) 7

THE LOW-POWER WIRING GUIDELINES 7

Parameter Settings 7

Encoder feedback (Pins 1, 2, 8, 10, and 14, 15) 7

CANbus (Pins 3 & 4 and 5 & 6) 7

All other low power wiring 7

Protected Voltages 12

THE SYSTEM CONTROLLER'S WIRING DIAGRAM (EXAMPLE) 13

PWM (coil-voltage-current/PV) Drivers 14

Half-Bridge Drivers 17

Digital (driver) Outputs 18

Safety Output 18

Analog Output 18

Switch (digital) Inputs 19

Virtual Switches 19

Analog (voltage) Inputs 21

Pot Inputs 22

RTD Inputs 22

High Speed Digital Inputs 23

Encoder Inputs 24

Quadrature Encoders 24

Sine/Cosine Position sensors 25

TABLE OF CONTENTS cont'd

Sawtooth Position Transducer 26
CAN Ports 27
Power Supply Outputs 28
Keyswitch and Coil Return/Safety Output 29
3: PROGRAMMABLE PARAMETERS 30
PROGRAMMABLE PARAMETERS 30
PROGRAMMING MENUS 30
MENU CHART FORMAT 30
Terminology 31
SDO Write Message 33
THE PROGRAMMABLE PARAMETERS 34
SYSTEM CONTROLLER 34
Battery Discharge Indicator 35
INTPUTS 38
Switches 39
Virtual Switches 43
Analog Inputs 45
RTD Input 48
High Speed Digital Inputs 50
Encoder Input 51
OUTPUTS 54
PWM Drivers 55
Half-Bridge Drivers 64
Digital Drivers 68
Safety Output 69
Analog Output 70
CAN 71

TABLE OF CONTENTS cont'd

4: MONITOR VARIABLES 75

SYSTEM CONTROLLER 76

INPUTS 76

OUTPUTS 80

ACCELEROMETER 82

5: VEHICLE CONTROL LANGUAGE (VCL) 84

VCL OVERVIEW 84

SUMMARY OF VCL BASICS 84

Variables and Constants 85

VCL VARIABLE TYPES AND MEMORY 85

SDO Write Message 86

VCL Function Examples 86

VCL, Watchdog Timer and Faults 87

CAN FUNCTIONS (VCL SETUP) 93

NMT Control 93

Node Guarding 94

Emergency Message Monitoring 97

SDO Management 100

CREATING AND USING CAN MAILBOXES (VCL SETUP) 106

Configuration of a Transmit Mailbox 107

Configuration of a Receive Mailbox 109

Controlling a Receive Mailbox 113

Configuration of the non-CANopen Mailbox functions 114

SRDOS 116

Example Transmit SRDO Setup 118

Example Receive SRDO Setup 120

6: INITIAL SETUP & COMMISSIONING 121

INITIAL SETUP 121

BEFORE YOU START 121

TO BEGIN 121

PARAMETER SETTINGS – METHOD OVERVIEW 122

TABLE OF CONTENTS cont'd

7: DIAGNOSTIC AND TROUBLESHOOTING 125
THE DIAGNOSTICS PROCESS 125
8: MAINTENANCE 132
CLEANING 132
FAULT HISTORY 132
APPENDIX A 133
CANBUS PDO MAP SETUP 133
VCL FUNCTIONS 139
APPENDIX B: VEHICLE SYSTEM DESIGN CONSIDERATIONS-&-RECYCLING 150
ELECTROMAGNETIC COMPATIBILITY (EMC) 150
DECOMMISSIONING AND RECYCLING THE CONTROLLER 151
APPENDIX C: PROGRAMMING DEVICES FOR THE 1351 152
APPENDIX D: 1351 MODELS AND SPECIFICATIONS 154

FIGURES

FIGURE 1: THE 1351 SYSTEM CONTROLLER 3
FIGURE 2: MOUNTING AND DIMENSIONS (INCHES & MM) 4
FIGURE 3: THIRTY-FIVE PIN AMPSEAL CONNECTOR 6
FIGURE 4: EXAMPLE WIRING DIAGRAM 13
FIGURE 5: WIRING FOR 3-WIRE POTENTIOMETER 123
FIGURE 6: WIRING FOR 2-WIRE POTENTIOMETER 123

TABLE OF CONTENTS cont'd

TABLES

TABLE 1: THE AMPSEAL CONNECTOR COMPONENTS & PART NUMBERS 6
TABLE 2: LOW CURRENT CONNECTIONS 8
TABLE 3: LOW CURRENT CONNECTION'S PROTECTED VOLTAGE 12
TABLE 4: DRIVER OUTPUTS ELECTRICAL SPECIFICATIONS 17
TABLE 5: HALF-BRIDGE DRIVERS ELECTRICAL SPECIFICATIONS 17
TABLE 6: DIGITAL OUTPUTS (DRIVERS) ELECTRICAL SPECIFICATIONS 18
TABLE 7: ANALOG OUTPUT SPECIFICATIONS 18
TABLE 8: SWITCH (DIGITAL) INPUTS ELECTRICAL SPECIFICATIONS 20
TABLE 9: ANALOG (VOLTAGE) INPUTS ELECTRICAL SPECIFICATIONS 21
TABLE 10: POTENTIOMETER INPUT ELECTRICAL SPECIFICATIONS 22
TABLE 11: RTD INPUTS ELECTRICAL SPECIFICATIONS 23
TABLE 12: HIGH SPEED DIGITAL INPUTS 23
TABLE 13: QUADRATURE ENCODER INPUTS 24
TABLE 14: SINE/COSINE SENSOR INPUTS 25
TABLE 15: SAWTOOTH SENSOR INPUTS 26
TABLE 16: CAN PORTS 27
TABLE 17: POWER-SUPPLY OUTPUTS ELECTRICAL SPECIFICATIONS 28
TABLE 18: KSI & COIL RETURN/SAFETY OUTPUT 29
TABLE 19: PROGRAMMABLE PARAMETERS MENUS 31
TABLE 20: MONITOR VARIABLES MENUS 75
TABLE 21: VCL VARIABLES BY MEMORY TYPE 86
TABLE 22: LED FLASH PATTERNS 125
TABLE 23: FAULT RECORD (SUB-INDEX & BYTES) 126
TABLE 24: FAULT TABLE: THE SET/CLEAR CONDITIONS & FAULT ACTIONS 127
TABLE D: MODEL CHART AND SPECIFICATIONS 154

1 — OVERVIEW

The Curtis Model 1351 System Controller provides 26 multi-functions I/O for application in stand-alone or CAN connected systems. With ample user code space and the enhanced real-time Curtis Vehicle Control Language (VCL), OEMs can use the 1351 controller to develop a wide range of vehicle and system control applications as the manger or an ancillary in a multi-module installation.

The 1351 System Controller was designed for a wide variety of applications, such as material handling vehicle mangers, base controllers for aerial trucks, operator interface in man-up platforms, land-based installation controllers replacing PLCs, ICE and Hybrid system controllers, hydraulic manifold control and many others. Some of these applicable vehicles are picture on this page.

HOW TO USE THIS MANUAL

This manual describes how to:

- Properly mount and wire the module
- Understand the configurable inputs and outputs
- Apply specific features to match an application
- Access and change parameters
- View and use monitor variables
- Customize applications with the Curtis Vehicle Control Language (VCL)
- Preform an initial setup following the commissioning guidelines
- Diagnose and troubleshoot faults
- Select and use the available programming and diagnostic tools

GETTING THE MOST OUT OF YOUR CURTIS CONTROLLER

Thoroughly read and refer to this manual to apply and configure the 1351. Understanding the installation & wiring guidelines, the parameter settings, the VCL functions, the initial setup & commissioning, and use the diagnostic and troubleshooting guide are critical to proper operation of the 1351 System Controller. For additional technical support, contact the Curtis distributor or the regional Curtis sales-support office.

The Curtis Difference

You feel it when you drive it

2 — INSTALLATION SPECIFICATIONS AND WIRING

PHYSICALLY MOUNTING THE 1351 SYSTEM CONTROLLER

Figure 1 is the 1351 System Controller. Its outline and mounting-hole dimensions are in Figure 2.

Mount the controller to a flat surface devoid of protrusions, ridges, or a curvature that can cause damage or distortion to its heatsink (base plate). To simplify the use of the 3-axis accelerometer, mount the 1351 in a level/orthogonal orientation. Secure the controller using evenly torqued bolts to the vehicle's mounting surface. When mounted to a metal surface, additional heat sinking or fan cooling is not necessary to meet the 1351's peak and continuous current ratings.

When installed with the matching vehicle-harness connector, the 1351 system controller meets the IP65 requirements for environmental protection against dust and water. Nevertheless, in order to prevent external corrosion and leakage paths from developing, select a mounting location that will keep the controller as clean and dry as possible.

Figure 1
The 1351 System Controller

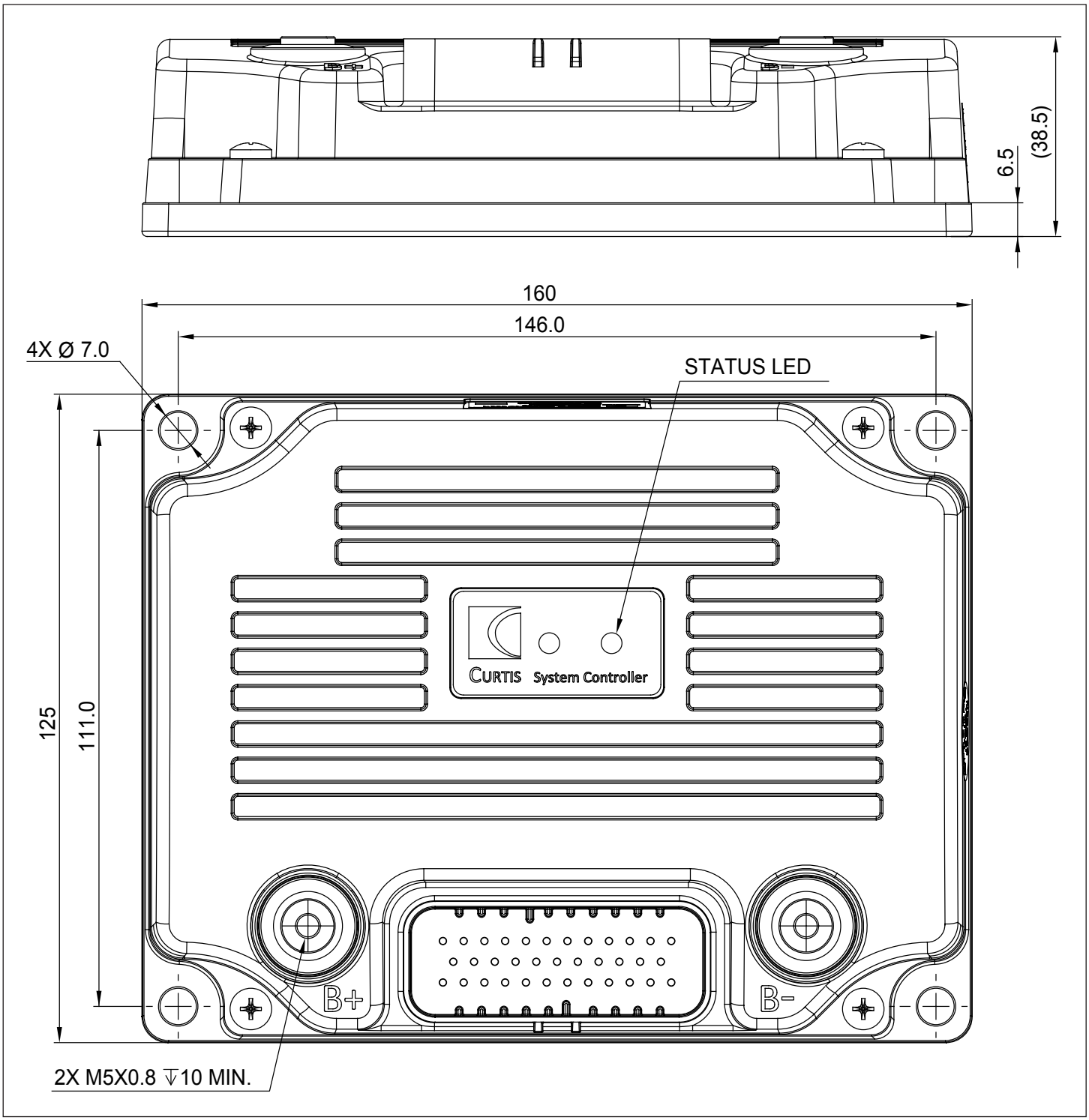


Figure 2

Mounting and dimensions (inches & mm)

PRECAUTIONS

Take the steps during the design and development of the application to ensure that the EMC performance complies with applicable regulations; EMC mitigation techniques are in Appendix B.

⚠ CAUTION

Working on electrical systems is potentially dangerous. Protect yourself against uncontrolled operation, high current arcs, and outgassing from lead-acid batteries:

UNCONTROLLED OPERATION—Some conditions can cause the system actuators or motors to run unexpectedly—be alert of the system's moving parts and systems disconnect. Block or stay well clear of any potentially trapping, pinching or impact area during development. Test and debug the application in a safe and controlled area.

HIGH CURRENT ARCS—Batteries can supply very high power where arcing can occur if they are short-circuited. Always open the battery circuit before working on the motor control circuits.

Wear safety glasses, and use properly insulated tools to prevent shorts.

LEAD ACID BATTERIES—Charging or discharging generates hydrogen gas, which can build up in and around the batteries. Follow the battery manufacturer's safety recommendations.

Wear safety glasses when servicing, charging and working around the battery.

LITHIUM ION BATTERIES—Follow the battery manufacturer's "safety precautions for the Lithium Ion battery pack."

Wear safety glasses when servicing, charging and working around the system's battery.

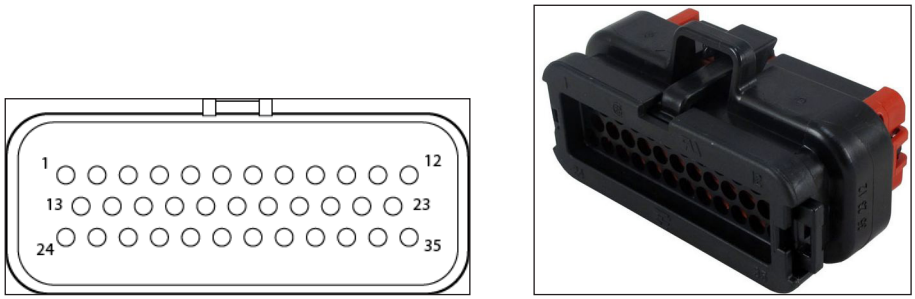
THE 35-PIN AMPSEAL CONNECTIONS

All logic and low power connections are through a single 35-pin AMPSEAL connector integrated into the cover utilizing gold-plated pins. The matching receptacle's wire silos come *sealed* by a membrane. Pierced the membrane by inserting the individually terminated wires. To maintain the IP65 rating, use the proper wire gauge and insulation thickness. Seal any non-used wire positions that have their silo-diaphragm pierced with the specific seal plug.

- Figure 3 shows The AMPSEAL receptacle (plug) housing and silo numbering.
- Table 1 lists the matching vehicle harness components.
- Table 2 lists the individual 1351 System Controller inputs and outputs characterizations, by pin number, including the associated VCL functions and diagnostic Monitor variables.

Figure 3

Thirty-Five pin AMPSEAL connector



AMPSEAL 35 PIN Connector
Harness-side view

Table 1 The AMPSEAL Connector Components & Part Numbers

Matching AMPSEAL 35-pin Component *	Part Number
AMPSEAL receptacle housing (the black vehicle-harness *plug*)	776164-1
Plug's gold-plated socket terminals (strip form p/n)	770520
Plug's gold-plated socket terminals (loose piece p/n)	770854-3
Silo seal plug (for *non-used* pin positions with a *pierced* membrane)	770678-1
Harness wire size (gauge)	0.5 – 1.25 mm^2 (20 – 16 AWG)
Wire diameter (overall) [i.e., uses wire with thin-wall insulation]	1.7 – 2.7 mm

**AMPSEAL components and tooling are available worldwide from multiple TE Connectivity electrical component distributors.*

*Reference the TE Connectivity Document: Application Specification **114-16016**. http://www.te.com/commerce/DocumentDelivery/DDEController*

TE Connectivity website: http://www.te.com/usa-en/products/connectors/automotive-connectors/intersection/ampseal-connectors.html

BATTERY CONNECTIONS (B+, B–)

The 1351 utilizes the 35-pin connector for the logic I/O. The B+ and B– studs provide the higher current needs of the drivers than a typical KSI supplied controller. Notice that the controller uses two pins (11 & 12) for the Coil Return/Safety Output functions. Use both pins if the peak current of the operating drivers could exceed 13 amps. Additionally, use the Coil Return/Safety Output to provide reverse battery polarity protection and safe shutdown of a failed driver.

B+/B– bolt/screw specification:

- Thread: M5 x 0.8
- Material/Class: Class 5.8 or better
- Maximum thread engagement length: 10mm
- Minimum thread engagement: 6 mm
- Torque: 3.2 ± 0.4N.m (28.3 ± 3.5 in-lbs.)

THE LOW-POWER WIRING GUIDELINES

Parameter Settings

The generic 35-pin I/O assignments described in Table 2 correspond to the example wiring diagram configuration shown in Figure 4. Many inputs and outputs (I/O) are configurable by their parameter settings to operate as a digital switch input, an analog input, or as a low-side driver for energizing contactor and relay coils. Table 2 lists the alternative uses, while Tables 4 – 18 provide further details by type.

Encoder feedback (Pins 1, 2, 8, 10, and 14, 15)

All four wires (+5 V, Feedback(s) A & B, and I/O ground) of these position signals should be bundled together as they run between the rotational/position sensor and system controller logic connector. Often, they routing is with the rest of the low current wiring harness, but avoid routing near motor cables. In applications where routing with high-power cables is necessary, shielded cable should be used with the ground shield connected to the I/O ground (pin 8) only at the controller. In extreme applications, utilize common mode filters (e.g. ferrite beads).

CANbus (Pins 3 & 4 and 5 & 6)

A good practice is to route the CAN wires as a twisted pair. Keep the CANbus wiring away from the high current cables, crossing such cables at right angles when necessary.

All other low power wiring

Route the remaining low power wiring according to standard practices. When designing the vehicle's wiring and routing, keep the input lines such as throttle, temperature, and the above mentioned motor feedback signals separate from controller's output lines such as the coil driver outputs. Avoid routing the low-power wiring parallel to the high power (and current) battery and motor cables.

Table 2 Low Current Connections

Pin	Driver Output	Digital Input	Analog Input (Range)	Special or Dedicated Usage	Related VCL*	
					Function	References
1	Digital Out 1	Switch Input 13 Virtual Switch 1	Analog 1 (0-5V)	Encoder 2B	Automate_Driver( ) Put_Driver( )	Digital_Out_1_State Switch_13 Analog_1_Volts Virtual_Switch_1 Encoder_2_RPM Encoder_2_Position
					CIT/1313 HHP Programmer: *Configuration\Outputs\Digital Drivers* *Configuration\Inputs\Switches\SW 13* *Configuration\Inputs\Virtual Switches\VSW 1* *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\Encoder Input*	
2		Switch Input 14 Virtual Switch 2	Analog 2 (0-5V)	Encoder 2A	Scale_Value( )	Analog_2_Volts Switch_14 Virtual_Switch_2 Encoder_2_RPM Encoder_2_Position
					CIT/1313 HHP Programmer: *Configuration\Inputs\Switches\SW 14* *Configuration\Inputs\Virtual Switches\VSW 2* *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\Encoder Input*	
3				CAN 1 Low	Setup_CAN_Transmit_Mailbox( ) Setup_CAN_Receive_Mailbox( ), Etc.	
					CIT/1313 HHP Programmer: *Configuration\CAN\Port 1*	
4				CAN 1 High	See CAN 1 Low	
					CIT/1313 HHP Programmer: *Configuration\CAN\Port 1*	
5				CAN 2 Low		
					CIT/1313 HHP Programmer: *Configuration\CAN\Port 2*	
6				CAN 2 High		
					CIT/1313 HHP Programmer: *Configuration\CAN\Port 2*	
7				KSI		Keyswitch_Voltage
8				I/O Ground		
9				+ 12V (Output)	Control_External_Power()	Ext_12V Ext_12V_Current
					CIT/1313 HHP Programmer: *Configuration\System Controller\External Supplies*	

Table 2 Low Current Connections, cont'd

Pin	Driver Output	Digital Input	Analog Input (Range)	Special or Dedicated Usage	Related VCL*	
					Function	References
10				+ 5V (Output)	Control_External_ Power()	Ext_5V Ext_5V_Current
					CIT/1313 HHP Programmer: *Configuration\System Controller\External Supplies*	
11	Safety Output			Coil Return	Put_Driver( )	Safety_Output_State
12	Safety Output			Coil Return	Put_Driver( )	Safety_Output_State
13	PWM Driver 1	Switch Input 1		Proportional	Automate_Driver( ) Put_Driver( ) Setup_Ramp( ), Etc.	Driver_1_PWM Driver_1_Current Switch_1
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver1* *Configuration\Inputs\Switches\SW 1*	
14		High Speed Input 1 Virtual Switch 3	Analog 3 (0-5V)	Encoder 1B	reset_pulse_counter( )	Virtual_Switch_3 Analog_3_Volts Encoder_1_RPM Encoder_1_Position
					CIT/1313 HHP Programmer: *Configuration\Inputs\Virtual Switches\VSW 3* *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\High Speed Inputs* *Configuration\Inputs\Encoder Inputs*	
15		High Speed Input 2 Virtual Switch 4	Analog 4 (0-5V)	Encoder 1A	reset_pulse_counter( )	Virtual_Switch_4 Analog_4_Volts Encoder_1_RPM Encoder_1_Position
					CIT/1313 HHP Programmer: *Configuration\Inputs\Virtual Switches\VSW 4* *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\High Speed Inputs* *Configuration\Inputs\Encoder Inputs*	
16		Virtual Switch 5	Analog 5 (0-20V)	RTD 1		Virtual_Switch_5 Analog_5_Volts RTD_1_R RTD_1_OUT
					CIT/1313 HHP Programmer: *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\RTD Input* *Configuration\Inputs\Virtual Switches\VSW 5*	
17		Virtual Switch 6	Analog 6 (0-20V)	RTD 2		Virtual_Switch_6 Analog_6_Volts RTD_2_R RTD_2_OUT
					CIT/1313 HHP Programmer: *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\RTD Input* *Configuration\Inputs\Virtual Switches\VSW 6*	

Table 2 Low Current Connections, cont'd

Pin	Driver Output	Digital Input	Analog Input (Range)	Special or Dedicated Usage	Related VCL*	
					Function	References
18		Virtual Switch 7	Analog 7 (0-20V)	RTD 3		Virtual_Switch_7 Analog_7_Volts RTD_3_R RTD_3_OUT
					CIT/1313 HHP Programmer: *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\RTD Input* *Configuration\Inputs\Virtual Switches\VSW 7*	
19		Virtual Switch 8	Analog 8 (0-20V)	RTD 4		Virtual_Switch_8 Analog_8_Volts RTD_4_R RTD_4_OUT
					CIT/1313 HHP Programmer: *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\RTD Input* *Configuration\Inputs\Virtual Switches\VSW 8*	
20		Virtual Switch 9	Analog 9 (0-20V)	Pot Wiper		Virtual_Switch_9 Analog_9_Volts Wiper_Position Pot_Resistance
					CIT/1313 HHP Programmer: *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\Virtual Switches\VSW 9*	
21		Virtual Switch 10	Analog 10 (0-20V)	Pot High (0-10V)		Virtual_Switch_10 Analog_10_Volts
					CIT/1313 HHP Programmer: *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\Virtual Switches\VSW 5*	
22		Virtual Switch 11	Analog 11 (0-20V)	Analog Out		Virtual_Switch_11 Analog_Out_ Command Analog_11_Volts
					CIT/1313 HHP Programmer: *Configuration\Inputs\Analog Inputs* *Configuration\Inputs\Virtual Switches\VSW 11* *Configuration\Outputs\Analog Output*	
23	HB Driver 1				Automate_Driver( ) Put_Driver( )	Driver_11_PWM Driver_11_Current
					CIT/1313 HHP Programmer: *Configuration\Outputs\Half-Bridge Drivers*	
24	PWM Driver 2	Switch Input 2		Proportional	Automate_Driver( ) Put_Driver( )	Driver_2_PWM Driver_2_Current Switch_2
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver2* *Configuration\Inputs\Switches\SW 2*	

Table 2 Low Current Connections, cont'd

Pin	Driver Output	Digital Input	Analog Input (Range)	Special or Dedicated Usage	Related VCL*	
					Function	References
25	Digital Out 2	Switch Input 11			Automate_Driver( ) Put_Driver( )	Digital_Output_2_State Switch_11
					CIT/1313 HHP Programmer: *Configuration\Outputs\Digital Drivers* *Configuration\Inputs\Switches*	
26	PWM Driver 3	Switch Input 3			Automate_Driver( ) Put_Driver( )	Driver_3_PWM Driver_3_Current Switch_3
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver3* *Configuration\Inputs\Switches\SW 3*	
27	PWM Driver 4	Switch Input 4			Automate_Driver( ) Put_Driver( )	Driver_4_PWM Driver_4_Current Switch_4
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver4* *Configuration\Inputs\Switches\SW 4*	
28	PWM Driver 5	Switch Input 5			Automate_Driver( ) Put_Driver( )	Driver_5_PWM Driver_5_Current Switch_5
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver5* *Configuration\Inputs\Switches\SW 5*	
29	PWM Driver 6	Switch Input 6			Automate_Driver( ) Put_Driver( )	Driver_6_PWM Driver_6_Current Switch_6
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver6* *Configuration\Inputs\Switches\SW 6*	
30	PWM Driver 7	Switch Input 7			Automate_Driver( ) Put_Driver( )	Driver_7_PWM Driver_7_Current Switch_7
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver7* *Configuration\Inputs\Switches\SW 7*	
31	PWM Driver 8	Switch Input 8			Automate_Driver( ) Put_Driver( )	Driver_8_PWM Driver_8_Current Switch_8
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver8* *Configuration\Inputs\Switches\SW 8*	
32	PWM Driver 9	Switch Input 9			Automate_Driver( ) Put_Driver( )	Driver_9_PWM Driver_9_Current Switch_9
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver9* *Configuration\Inputs\Switches\SW 9*	

Table 2 Low Current Connections, cont'd

Pin	Driver Output	Digital Input	Analog Input (Range)	Special or Dedicated Usage	Related VCL*	
					Function	References
33	PWM Driver 10	Switch Input 10			Automate_Driver( ) Put_Driver( ) Automate_ Frequency_Output( ) Disable_Frequency_ Output( )	Driver_10_PWM Driver_10_Current Frequency_Output Virtual_Switch_10 Switch_10
					CIT/1313 HHP Programmer: *Configuration\Outputs\PWM Drivers\Driver10* *Configuration\Inputs\Switches\SW 10*	
34	Digital Out 3	Switch input 12			Automate_Driver( ) Put_Driver( )	Digital_Output_3_ State Switch_12
					CIT/1313 HHP Programmer: *Configuration\Outputs\Digital Drivers* *Configuration\Inputs\Switches*	
35	HB Driver 2				Automate_Driver( ) Put_Driver( )	Driver_12_PWM Driver_12_Current
					CIT/1313 HHP Programmer: *Configuration\Outputs\Half-Bridge Drivers*	

** The related VCL columns are applicable when employing VCL. Use the VCL "functions" to access the various I/O features. VCL "references" are predefined names for specific pins and use in the CIT/1313-HHP Monitor app. References are read-only Monitor variables. Refer to the OS SysInfo file for specific VCL functions, constants, controller system variables, usage, and CAN Object IDs.*

Protected Voltages

The low-power pins' protected voltage ratings listed in Table 3 are absolute—they are not for normal operation. To prevent damage to the controller, do not connect (short circuit) the I/O Ground (pin 8) to battery positive.

Table 3 Low Current Connections Protected Voltages

Controller Pin Number	Minimum (reverse) Voltage	Maximum Voltage	ESD	
			Air Discharge	CONTACT
All other pins	– 1	MaxV + 10V	±15kV	±8kV
7 (KSI)	– MaxV			
11, 12 (Coil Return/Safety Output)	– MaxV			
8 (I/O Ground)	Not protected			

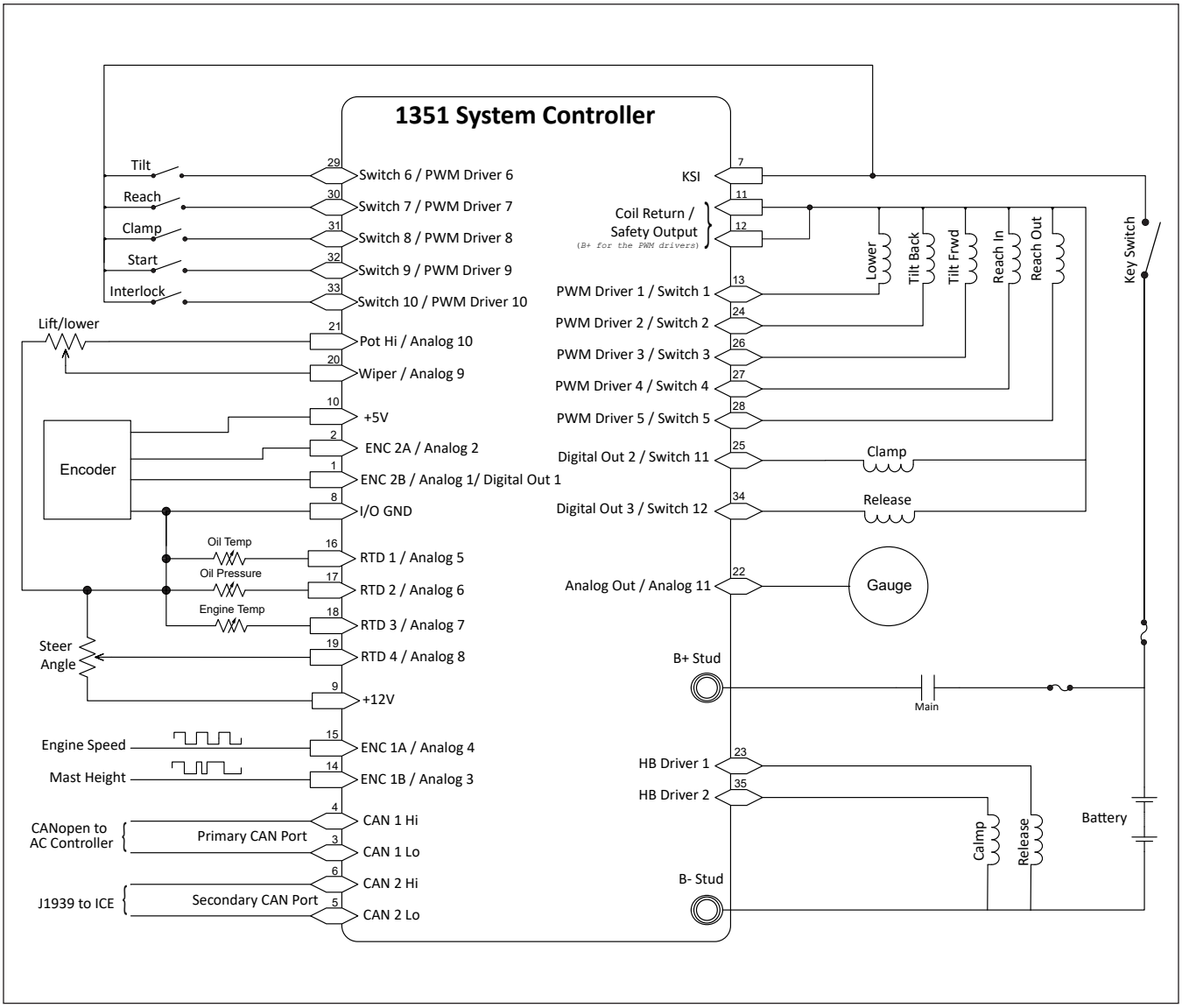
THE SYSTEM CONTROLLER'S WIRING DIAGRAM (EXAMPLE)

The 1351's Inputs and Outputs (I/O) can easily conform to a wide range of applications. Use this chapter and Chapter 3's parameters information for insight into using and setting up each type of I/O. Figure 4 is the example-wiring diagram using the 1351 as an ICE driven hydraulic systems controller. In this example, the Lift/Lower and Steer Angle potentiometers connections are 3-wire potentiometer inputs, while the control of the main contactor is by another device. Figure 4's connections are representative of just some of the possible 1351 System Controller wiring configurations.

Note: In cases where an application's wiring and I/O usage deviates from the wiring shown in Figure 4, it is up to the OEM to evaluate the overall system safety. Always ensure a configuration is thoroughly tested and verified for the application.

Figure 4

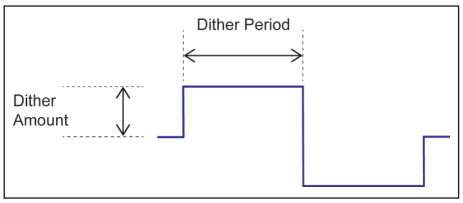
Example Wiring Diagram



PWM (coil-voltage-current/PV) Drivers

Drivers 1 through 10 utilize a low side quarter-bridge topology to drive inductive loads connected to the Coil Return (B+). These are high-frequency pulse width modulation (PWM) drivers. Via parameter settings, each has one of four operating modes: Off, Direct PWM, Voltage Compensated, and Current Control. The drivers can control (sink) up to 3 amperes, although the 15 total drivers have a combined current limit of 23 amperes (when using both Coil Return pins 11 & 12*). All inductive loads shall be connected to the coil return (pins 11 & 12), which provides flyback diode protection. Resistive and RC (inrush) loads shall not exceed three amps (peak). Each driver offers current measurement, output state monitoring, and open/short fault detection. During setup and development, use the Test feature (via VCL) to verify the fault detection of the driver circuits and external load connections. Each driver is protected against shorts to B+ or B–.

Off (Open)	The output driver FET is off (non-energized). Use this mode to disable the driver output to allow the pin usage as an input.
Direct PWM	The Direct PWM mode allows the driver to produce PWM as commanded. FET and wiring diagnostics are performed in this mode. The PWM produced is equal the command percentage by directly writing to the variable *Driver_X_Command*, where the value of 0 – 1000 commands from 0.0 – 100.0% PWM.
Voltage Comp	The Voltage Compensated mode continuously regulates the driver PWM based on the command, the *Nominal Voltage* setting and the present battery voltage in an attempt to provide a constant average voltage at the output. FET and wiring diagnostics are performed in this mode. Initial and continuous PWM % timing control is applied. The command is a % of nominal voltage desired at the output.
Current Control	The Current Regulated Mode interprets the command as a load current request. This mode uses the feedback from current shunts to regulate (using a PI controller) the PWM and thus provide the requested load current. Use this mode to control Proportional Valves. It requires several additional features for proper position control: Dither, Min/Max current settings, and PI gain settings. On a normally closed valve, the Maximum current will provide a full open valve and minimum current commands a fully closed valve. Setting the minimum current above zero amps allows the valve to be quickly energized, and ready to move quickly when commanded. Use the Maximum Current setting to calculate the dither amount. A 0.0% command will command zero current and the FET will be completely off (irrespective of the Minimum setting). 0.1% will command a jump to the Minimum Current and from there the current will linearly increase to Maximum Current at 100% command. Dither provides an amount of cyclically changing current to vibrate the solenoid and keep it from "sticking" in one position. Without dither, it is harder to make small adjustment in the proportional valve position. Dither has both a frequency and amount. Note that the PI controller gains are critical in controlling dither and should be adjusted to ensure the dither is properly generated.



Dither Waveform (*current control/proportional valve mode*)

Note: The drivers' measure current just before the PWM is shut off; therefore, there is a lower limit where the current cannot be read. For the low side drivers, this is 8%. For HB drivers (see below) in low-side mode, this is 12%. The system designer should make sure that the load resistance, voltage supply and minimum current settings will keep the PWM above these values for good regulation.

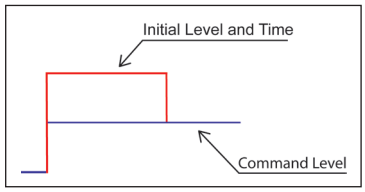
The drivers have the pull-in and hold control feature and command ramping in all three active modes.

**The sum of all loads shall not exceed combined Coil Return (pins 11 & 12) current rating of 23 amps.*

Pull-in and Hold

The Pull-in and Hold function works on the raw VCL command (*Driver_X_Command*) and is applied before the Ramping Function (see below). This function provides a time at an Initial Level and then the rest of the time at the applied command. The function is reset each time the command goes to 0%.

The Initial Level may be above or below the applied command. The figure below represents an Initial level that was above the applied command.

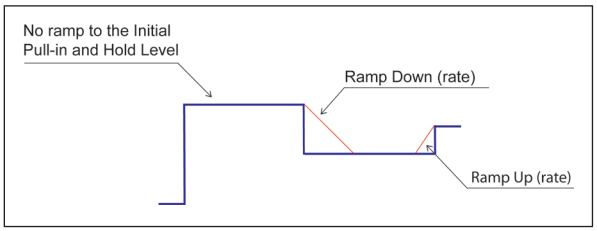


Pull-in and Hold function

Typically, the Initial Command is above the applied (hold) command as it provides a strong initial pull-in of the contactor tips, after which the lower hold-value reduces coil losses and heat when the contactor is closed. It can also be set low to allow a jump function to some "minimum" value that is commend without any ramp.

Ramping

Each driver's ramping functions occur following the commanded pull-in and hold. Following the initial ramp functions, the ramping applies to all changes in the final driver command. There are separate parameter values for ramping up and ramping down the command. Note that the initial pull-in command value is jumped-to and not ramped. (Setting the initial time and initial value of 0 defeats the pull-in function and the ramp up be applied to the first command from 0%).



Ramp-Down and Up function

The ramping times are the time it takes to go between the drivers' set minimum command to the driver's maximum command. In direct PWM, this is from 0 to 100% PWM, but in the Current controlled mode, this is from the Minimum to the Maximum Current parameters. If these parameters are changed, the actual ramp time will stay consistent as programmed, but the final "rate" will change.

The drivers can be commanded by directly writing to the variable *Driver_X_Command*, where the value of 0 – 1000 commands from 0.0 – 100.0% PWM. The other applicable VCL driver functions are *Automate_Driver()* and *Put_Driver()* as listed in Table 2.

The output drivers can also be used as digital inputs (switches 1–10) and are included in that group as well. The wiring example, Figure 4, illustrates drivers 6-10 as switched inputs.

Table 4 Driver Outputs Electrical Specifications

Signal Name	Pin	Output Type / Activity Level Output Low Voltage	PWM Frequency / Accuracy	Output Current* / Short-Circuit	Current Measurement / Sense Limit	Input Impedance
Driver 1	13	Low-Side Driver / Low = On / < 0.25 V at full current and 100 % PWM	> 15kHz / ± 0.5% of full range	3 Amps / < 30 Amps for < 100µS	10mA – 3.5 Amps / 8 – 100% PWM	24kΩ 12-48V models / 47kΩ 36-96V models
Driver 2	24					
Driver 3	26					
Driver 4	27					
Driver 5	28					
Driver 6	29					
Driver 7	30					
Driver 8	31					
Driver 9	32					
Driver 10	33					

Half-Bridge Drivers

The Half-Bridge drivers are high frequency PWM drivers with current feedback. They can be set to operate as Low or High-Side drivers. In the Low-Side topology, connect the load to Coil Return/Safety Output and the driver pin. In the High-Side topology, connect the load from B–to the driver pin.

Set the topology type using its *Type* parameter. The Half-Bridge (HB) Drivers also have the same control modes as the PWM Drivers. Use them to drive high-side connected (Coil Return) or low-side connected (B–) loads in constant current, constant voltage or PWM % modes. These drivers are short circuit protected to B+ or B– .

Table 5 Half-Bridge Drivers Electrical Specifications

Signal Name	Pin	Output Type / Activity Level Output Low Voltage	PWM Frequency / Accuracy	Output Current* / Short-Circuit	Current Measurement / Sense Limit	Input Impedance
HB Driver 1	23	Half-Bridge Driver / Low-Side = 0 High-Side = 1 / < 0.25 V at full current and 100 % PWM	> 15kHz / ± 0.5% of full range	3 Amps / < 30 Amps for < 100µS	10mA – 3.5 Amps / 12 – 88% PWM	> 650kΩ
HB Driver 2	35					

Digital (driver) Outputs

The three digital outputs are low-side drivers with a 3-ampere current (sink) limit. As low-side drivers, they can only be turned ON or OFF. The control modes for PWM, constant current or voltage are not available. For inductive loads, these drivers have a fly-back diode to Coil Return. Resistive loads must not exceed the 3-amp current-sink rating. The VCL function *Digital_Out_X_Command* (where X is 1, 2 or 3) or *Put_Driver()* is used to control these drivers (where a non-zero value = ON).

Digital Output 1 can also be used as either the Encoder 2B or analog1 inputs, based upon parameter settings. The Digital Outputs 2 and 3 parameters can be disabled for use as digital switch inputs (Switches 11 and 12 respectfully).

Table 6 Digital Outputs (drivers) Electrical Specifications

Signal Name	Pin	Output Type / Activity Level	Output Current / Short-Circuit	Input Impedance (Off State)
Digital Out 1	1	Low-side Driver / On/Off	3 Amps (100% On) / < 30 Amps for < 100µS	> 350kΩ
Digital Out 2	25			> 650kΩ
Digital Out 3	34			

Safety Output

The Safety Output provides the power to all connected loads. The safety output must be enabled for the connected drivers to operate. The Safety Output is normally used as the Coil Return connection for low-side driver loads, but it can also be used as a high-side driver for a safety/system contactor.

Once enabled, the safety output can supply up to 23 amps of B+ voltage at the pins (in common) 11 and 12 (see the **Safety Output Command** parameter).

The system designer shall ensure the driver loads at pins 11 and 12 are under the combined 23-ampere current rating. The safety output will shut down if excessive over-current or a short is detected.

Analog Output

The system controller has one analog output. This adjustable 0-10V Op Amp output is intended to drive high-impedance loads, such as a battery discharge indicator or hour meter. This output is generated from a filtered PWM signal and has about 1% ripple. The 2% settling time is <25 ms for a 0–5 V step and <30 ms for a 0–10 V step. This output line is protected against shorts to B+ or B– . The analog output is parameter settable, sharing the output with the Analog 11 input (pin 22).

Set the output voltage by setting the VCL variable *Analog_Out_Command* from 0–1000 (0.00 to 10.00V).

Table 7 Analog Output Specifications

Signal Name	Pin	Output Voltage	Output Current
Analog Out	22	0 to 10 V	10 mA

Switch (digital) Inputs

The controller offers flexibility in configuring the analog/digital inputs as Off/On switch signals. All switch inputs have a pull-down resistor. Some have an additional selectable pull-up resistor, and a few can be enabled to become floating input (no pull-up with high impedance pull-down resistor).

When the Pull-down is enabled, wire the switch input to connect to B+ when "switched" On. With the pull-up enabled, the switch can be connected to B–(ground) when "switched" On. For Switch 13 (pin 1, Analog 1) and Switch 14 (pin 2, Analog 2), there are only the Pull-Up or Pull-Down options. When left floating, the input can be driven by a TTL or voltage device (i.e., not the typical toggle switch). The control of the pull-up/pull-down resistors state (Pull-Up, Pull-Down or Float) is by the Open State parameter for each switch when the input is being used as a switch.

In all cases, the input level that is considered ON is set by the Active Level parameter.

If *Active Level = Low*, and input voltage is less than 1V, the State will be *On*

Alternatively,

If *Active Level = High*, and input voltage is greater than 4V, the State will be *On*

Virtual Switches

Analog inputs 1 through 11 are also monitored as virtual digital inputs by comparing high and low threshold parameters and then applying the Active Level parameter. These are called *Virtual Switch Inputs*. They are configured using parameters VSW1-11 (**Virtual Switch 1** thru **11**). The result is placed in the VCL variables *Virtual_Switch_X* (where X is 1 through 11). See Table 8 for the comparison between Switches and Virtual Switches.

Table 8 Switch (digital) Inputs Electrical Specifications

Signal Name	Pin	Type Logic Threshold	Input Impedance
TYPICAL DIGITAL (SWITCHED) INPUTS			
Switch 1	13	Selectable Pull-up with fixed Pull-down* Low < 1 Volt High > 4 Volts***	Fixed Pull-down: > 23 kΩ (12-48V models)* > 43 kΩ (36-96V models)* Selectable Pull-up: 15kΩ to 15V
Switch 2	24		
Switch 3	26		
Switch 4	27		
Switch 5	28		
Switch 6	29		
Switch 7	30		
Switch 8	31		
Switch 9	32	Selectable Pull-up or Pull-down*	> 660 kΩ (all models)*float state Selectable Pull-down: 24kΩ for 12-48V controllers 47kΩ for 36-96V controllers Selectable Pull-up: 15kΩ to 15V
Switch 10	33		
Switch 11	25	Low < 1 Volt High > 4 Volts ***	
Switch 12	34		
Switch 13 **	1	Note, Switch 13 (pin 1) and Switch 14 (pin 2) are pulled-up to +5V as they are also used for encoder 2	
Switch 14 **	2		
VIRTUAL DIGITAL INPUTS			
Virtual_Switch_1 (Analog 1)	1	Settings correspond with the associated Analog Inputs The Analog Inputs (5-10) have pull-up resistors that can be enabled (tied to the configured function of RTD and potentiometer inputs).	> 178 kΩ
Virtual_Switch_2 (Analog 2)	2		
Virtual_Switch_3 (Analog 3)	14		
Virtual_Switch_4 (Analog 4)	15		
Virtual_Switch_5 (Analog 5)	16		
Virtual_Switch_6 (Analog 6)	17		
Virtual_Switch_7 (Analog 7)	18		
Virtual_Switch_8 (Analog 8)	19		
Virtual_Switch_9 (Analog 9)	20		
Virtual_Switch_10 (Analog 10)	21		
Virtual_Switch_11 (Analog 11)	22		
*With the associated Driver = Off			

** These pins have higher power pull-up or pull-down resistors that load the switch contact with approximately 1mA of current in an attempt to create small arching to clean the connects and prevent high resistance build-up.*

*** Switch Inputs 13 and 14 must have the same Open State setting as their pull-up resistor enable is tied together.*

**** The input voltage at the switch inputs should not be allowed to linger between the low and high threshold range. The state of the switch will be unknown in this region.*

Analog (voltage) Inputs

The system controller supports a variety of analog inputs. Analog inputs measure voltage applied to the input pin. The voltage range depends on the input used, as shown in Table 9.

Table 9 Analog (voltage) Inputs Electrical Specifications

Signal Name	Pin	Measurement Range / High/Low Threshold / Filter Time Constant	Input Impedance
Analog 1	1	0 – 5 V / 0 – 3.000 sec	> 650 kΩ
Analog 2	2		
Analog 3	14		
Analog 4	15		
Analog 5	16	0 – 20 V / 0 – 3.000 sec	> 200 kΩ
Analog 6	17		
Analog 7	18		
Analog 8	19		
Analog 9	20		
Analog 10	21		
Analog 11	22		

Pot Inputs

The 1351 System Controller provides Analog Inputs 9 and 10 to be configured for connection to a potentiometer. Potentiometers can be connected as 2-wire using only the wiper (Analog 9/pin 20) and ground (I/O Gnd, pin 8), or 3-wire, using the wiper, ground, and Analog 10 (pin 21) for pot high. The potentiometer input can be configured for a throttle, brake, steer, or other uses. The wiring diagram (Figure 4) illustrates this usage as a 3-wire lift/lower configuration.

When the analog inputs are configured for use with a potentiometer, the signals are dynamically tested, increasing the fault detection of the potentiometer and wiring beyond just basic out-of-range detection. The total resistance and the wiper position are constantly measured and calculated.

Table 10 Potentiometer Input Electrical Specifications

Signal Name	Pin	Pot Resistance Range / Tolerance / Available Current	Input Impedance	Output Voltage	Fault Detection
Pot Hi	21	3-wire: 0 – 15 kΩ / 0 – 2k Ω 3 mA supplied, max.	> 178 kΩ	15 V (nominal)	Shorted to: B+ B– I/O Gnd Open
Wiper	20	2/3-wire: 0 – 15 kΩ / 0 – 2k Ω 3 mA supplied, max.			Shorted to: B+ B– I/O Gnd Pot Hi Open
I/O GND	8	—	—	—	Open

RTD Inputs

The resistive temperature device (RTD) inputs, RTD Inputs 1 – 4, are connected to Analog Inputs 5 – 8 respectively. Specific or selectable RTD devices are not offered—rather each RTD input is configured (mapped) according to its resistive characteristics. This means the 1351's RTD Inputs can accommodate resistive devices for temperature, position, pressure, etc., solely based upon the resistive sensor and/or a given range determined by the user. For example, a 3-wire "steer angle" potentiometer is illustrated in the example wiring diagram (Figure 4) for RTD 4 input, with RTDs utilized for the oil temperature, oil pressure, and engine temperature connected to RTD inputs 1, 2, and 3.

The RTD parameter-mapping feature can be used for linearizing temperature diodes or NTC resistors to provide values in Fahrenheit or Celsius. Another example is when a potentiometer is used to measure drive wheel angle by a cam or level arm, which provide a non-linear movement based on angle. This happens on 4-wheel trucks with Ackerman steering geometry and/or dual drive vehicles. The steered angle can also be picked up off the actual hydraulic ram using the RTD mapping feature.

Alternatively, use the RTD to convert a tiller steering arm to have more sensitivity near center and then greater angle gain when close to the limits.

Configuring an RTD sensor is based upon its changing resistance over a set of four X/Y mapping parameters (points). Resistive input values below the map's lower point are limited to the lower output, while values above the maps' upper setting are clamped to the upper output. The resistive-values parameter points must be in an ascending/increasing order. RTD Inputs between the map's four resistive points are interpolated.

To use the analog input as a RTD, it must be enabled by the *RTD Enable* parameter (`Pull Up`). Note that the Analog channel's threshold settings will still be active for the analog input reading variables when the RTD is enabled (see Analog Inputs parameters). This means that the Virtual Switches 5-8 can be utilized to expand the system control logic when using RTD Inputs (such as using them to indicate a fault threshold).

Table 11 RTD Inputs Electrical Specifications

Signal Name	Pin	Resistive Range / Pull-up Voltage	Input Impedance
RTD 1	16	0 – 10,000 Ω / ~ 4.4V	> 178 kΩ
RTD 2	17		
RTD 3	18		
RTD 4	19		

High Speed Digital Inputs

There are two high-speed inputs on the 1351—High Speed Input 1 and High Speed Input 2. These inputs are intended to measure the frequency of incoming 5 volt pulses, the pulse width, or used as counters/accumulators. These two inputs can also be used for encoder inputs (see below), but cannot be used as both high-speed inputs and encoders simultaneously.

The function of each pulse input must be set before it can be used by means of its `Type` parameter. For high-speed signal processing, select when the pulse width and counter functions need to react. The parameter also sets whether the function will look at the rising edge or falling edge. If rising edge is selected, the pulse width and period will be measured from the rising of the signal to falling edge and the counter will increment on each rising edge. The frequency always measures from rising to rising edge.

Table 12 High Speed Digital Inputs

Signal Name	Pin	Options	Input Impedance	Edge Thresholds Rising/Falling Edge
HS Input 1	15	Frequency (Hz) Pulse Width (%) Counter	> 228 kΩ	~2.6V Rising ~1.3V Falling
HS Input 2	14			

Encoder Inputs

The 1351 System Controller accepts two position sensor inputs labeled as Encoder 1 and Encoder 2. Three types of encoders can be connected:

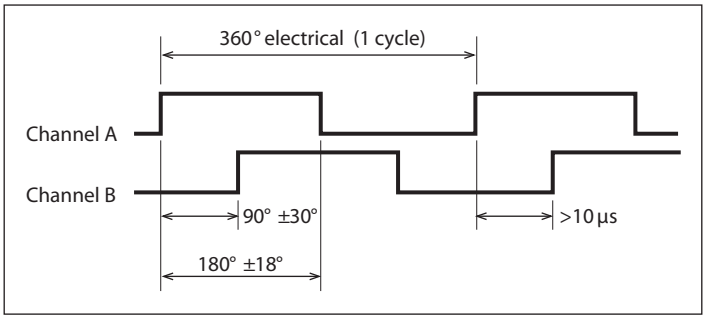
- Quadrature Encoders with Open Collector outputs
- Sine/Cosine Position sensors
- Sawtooth Position transducer

Each type of encoder requires unique wiring and setup parameters, yet the result is the same: position and speed measurements. After selecting the encoder *Type*, its corresponding parameter set will become available, otherwise it is hidden. Notice that Encoder 1 shares its inputs with the High Speed Digital Inputs (see above), thus it cannot be used as both the encoder and the high-speed inputs simultaneously. For each type of encoder, the input signals are mathematically converted to position and speed. In all cases, the connections are referenced to the controller's I/O ground (pin 8).

Quadrature Encoders

Table 13 Quadrature Encoder Inputs

Signal	Pin	Input Voltage Range	Logic Threshold	Pull-Up Resistance / Input Impedance	Maximum Frequency	A-B Phase Range	Phase Duty Cycle
ENC 1A (Analog 4)	15	0 – 5 V	Rising edge= 2.9 V max Falling edge= 2.0 V min	2 kΩ to 5 V / > 115 kΩ	15 kHz	90° ± 30°	50 % ± 10 %
ENC 1B (Analog 3)	14						
ENC 2A (Analog 2)	2						
ENC 2B (Analog 1)	1						



These quadrature encoder signal tolerances must be maintained throughout the application's operating conditions, including voltage and temperature, as well as speed and torque ranges when applied to a motor. A parameter setting is available for channel A and B direction—A before B or B before A.

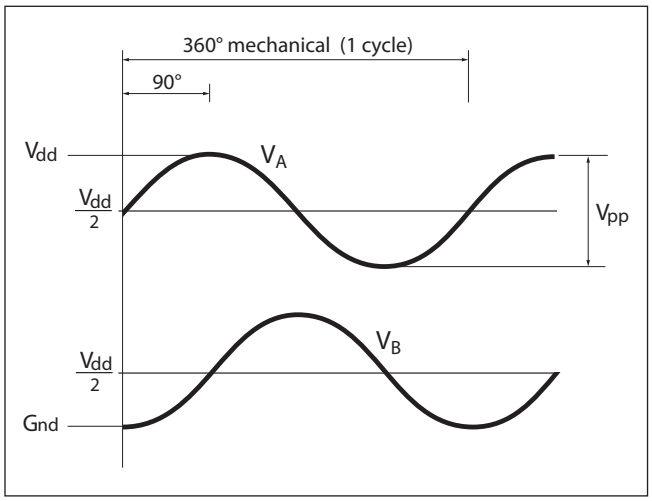
Sine/Cosine Position sensors

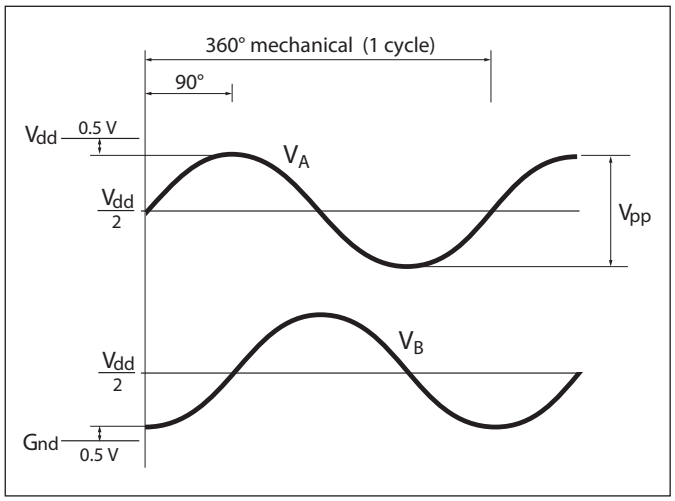
The sine/cosine sensor has two analog signals 90 degrees out of phase. The output voltages are in the form of a sine and cosine over its shaft 360-degree revolution. As illustrated below, the sensor must be connected with one waveform cycle per mechanical revolution. The waveform tolerances must be maintained throughout the application's operating conditions, including voltage and temperature, along with speed and torque ranges when applied to a motor. The waveform peaks must be away from a 5V Vdd and ground by at least 0.5 V (i.e., the right-hand image). If the sensor voltages go outside this range, position sensing will be faulty and not useable by the VCL functions. In this case, a fault will be detected. Channels for sine (e.g., A) and cosine (B) are parameter selectable for direction—A before B or B before A.

While a sin/cos sensor is typically used in Surface Permanent Magnet (SPM) motor applications, an example 1351 application is linear-travel position, such as fork reach, where one revelation covers the range from retracted to fully extended. Measurement and comparison of the two signals at any point can determine the absolute position of the sensor and thus the forks location. The rate of signal change versus time will equate to velocity.

Table 14 Sine/Cosine Sensor

Signal Name	Pin	Operating Voltage	Max. Frequency	Input Impedance
ENC 1A (Analog 4) / Sine	15	0 – 5 Vdd (typ.) 0.5 – 4.5 Vpp	200 Hz sinewave	> 115 kΩ
ENC 1B (Analog 3) / Cosine	14			
ENC 2A (Analog 2) / Cosine	2			
ENC 2B (Analog 1) / Sine	1			





Sawtooth Position Transducer

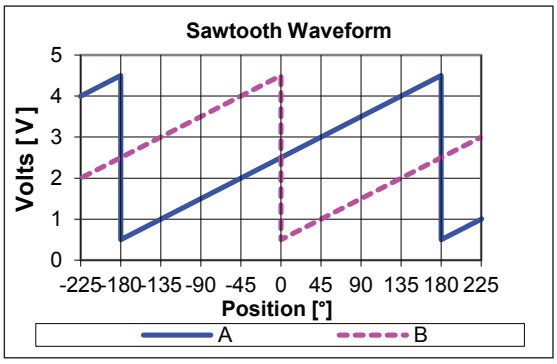
The sawtooth transducer has two analog signals that ramp up as its shaft rotates through 360 degrees. As illustrated below, the signals are offset by 180 degrees. Unlike the sine/cosine sensor, each signal provides an absolute indication of position, thus requiring a different set of setup parameters. The signal voltage range must be set and the voltage that represents 0 degrees is required. If the analog signals go outside the specified ranges for >60mS, a fault is declares. Also, if the position calculated using the channel A is outside a tolerance parameter compared to channel B for >60mS, a fault will be declared. The tolerances must be maintained throughout the application's operating conditions, including voltage and temperature, along with speed and torque ranges when applied to a motor.

As illustrated in the sawtooth waveform diagram, connect the sensor with one waveform cycle per mechanical revolution. Channels for A and B are parameter selectable for direction—A before B or B before A.

An example application is tracking linear-travel and position such as fork-reach where one revelation covers the range from retracted to full extend. Another is for steering angle where one steering-wheel rotation equates to one cycle (360°) of the sawtooth waveform(s).

Table 15 Sawtooth transducer

Signal Name	Pin	Operating Voltage / Signal Range (peak to peak)	Max. Frequency	Input Impedance
ENC 1A (Analog 4) / B	15	0 – 10 (min – max) / 0 – 10.0 V	200 Hz	> 115 kΩ
ENC 1B (Analog 3) / A	14			
ENC 2A (Analog 2) / B	2			
ENC 2B (Analog 1) / A	1			



CAN Ports

Two CAN ports are provided. Either port can be used for protocols that use the 11-bit identifier (such as CANopen) or the 29-bit (such as J1939). Port 1 must be used for FLASH programming and CIT connection. The CAN ports are not isolated, sharing the 1351 internal power supply and I/O ground (B -) reference. Each CAN port requires a unique Node IDs and can run a different baud rate. Additionally, they can be combined (connected together) via a parameter setting. When combined, each port's communications are received and sent out on the other port. When combined, both ports must use the same baud rate. Termination for each port is selected electronically and is controlled by the CAN port setup parameters. Upon power-up/KSI On, the programmed termination state of each port is applied in under 100 ms.

Table 16 CAN Ports

Signal Name	Pin	Data Rate (bps)	Input Impedance	CAN Termination (Internal)*
CAN1 Low	3	0 = 125k 1 = 250K 2 = 500K 3 = 800K 4 = 1M	<1000pF	120 Ω * settable parameter
CAN1 High	4			
CAN2 Low	5			
CAN2 High	6			

Power Supply Outputs

The +5 V and +12 V external supplies provide auxiliary power for low-power circuits such as encoders, potentiometers, RTDs, and similar devices. The I/O Ground (at pin 8) is the return line for these low power devices. Both power supply outputs are protected against shorts to B+ or B– . The default state is Off. They are individually enabled (turned On) using either their enable parameter or VCL.

The +5 V supply is typically used for encoder feedback devices (e.g., quadrature encoder or Sin/Cos sensor). The current draw on this supply is measured and can be used to detect command sensor faults, or wiring faults by use of min/max thresholds: see the monitor variable `Ext 5V Current`.

Use the +12 V supply for devices requiring higher voltage. Its current draw is measured and can be used to detect sensor faults or wiring faults by use of min/max thresholds: see the monitor variable Ext 12V Current.

Table 17 Power-Supply Outputs Electrical Specifications

Signal Name	Pin	Output Voltage	Output Current
+5 V	10	5 V ± 5 % (regulated)	150 mA (typ.)*
			MEASUREMENT RANGE
			5 – 315 mA
+ 12 V	9	12 V ± 15 % (load dependent)	150 mA (typ.)*
			MEASUREMENT RANGE
			5 – 315 mA
Ground (I/O Gnd)	8	—	—

** The combined load current between the +5V and +12V outputs cannot exceed 300 mA*

Keyswitch and Coil Return/Safety Output

Connect the KSI (keyswitch, pin 7) to B+ via a key switch and fuse as shown in Figure 4. The KSI input feeds the controller's internal power supplies before the main contactor closes. The lead-acid Battery Discharge Indicator (BDI) uses the keyswitch voltage.

The Coil Return/Safety Output (pins 11 and 12) are derived from B+. If the 1351 will drive a contactor that supplies power to the B+ stud, its coil must be wired to KSI (pin 7) and have its own flyback diode, because the Safety Output/Coil Return is only active if there is power on the B+ stud. Two Coil Return pins are used to increase the available sourcing current to the 10 low-side PWM drivers. Each of the coil return pins are limited to ~12 amperes*. Note that the peak sum of all currents possible though all loads connected to the Coil Return shall not exceed the two KSI Coil Return's maximum current rating. Table 4 lists the drivers' peak currents. The Safety Output (feature) is not PWM-able. It is either ON or OFF.

As illustrated in the wiring diagram, Figure 4, connect the Coil Return circuits directly to one side of the contactors' coil terminals and the other terminal to the driver pin. The controller includes an internal fly-back diode between each Driver and Coil Return to suppress the coils' inductive voltage spike. Coils with their own inductive-spike suppression are allowed (but care must be made to use the proper connections to Coil Return and the Driver output), yet resistive-based coil suppression is discouraged because of leakage currents within the 1351 between the drivers and coil return.

Proper connections to Coil Return and B+/B– studs ensure reverse polarity protection (vehicle wiring correct, battery terminals reversed). Reference the wiring diagram, Figure 4, for the typical KSI and Coil Return/Safety Output connections.

Table 18 KSI & Coil Return/Safety Output

Signal Name	Pin	Operating Voltage	Current**
KSI (keyswitch)	7	Between the under and over voltage cutback limits	9 A (input/sink)
Coil Return / Safety Output	11		23 amps total or ~ 12 amps per pin
Coil Return / Safety Output	12		

*** Gold plated terminal basis.*

** See the mating contact specification (Table 1) to achieve this amp rating.*

3 — PROGRAMMABLE PARAMETERS

PROGRAMMABLE PARAMETERS

The 1351 System controller's programmable parameters enable the user to customize it to the needs of specific applications even without using VCL. All parameters are programmed over CAN Port1 using the PC-based Curtis Integrated Toolkit™ (CIT) program or the CAN model 1313 handheld programmer[1]. The 1351 does not support serial-communication programming. CIT and the 1313HHP programmers automatically force all changes to be stored into non-volatile memory.

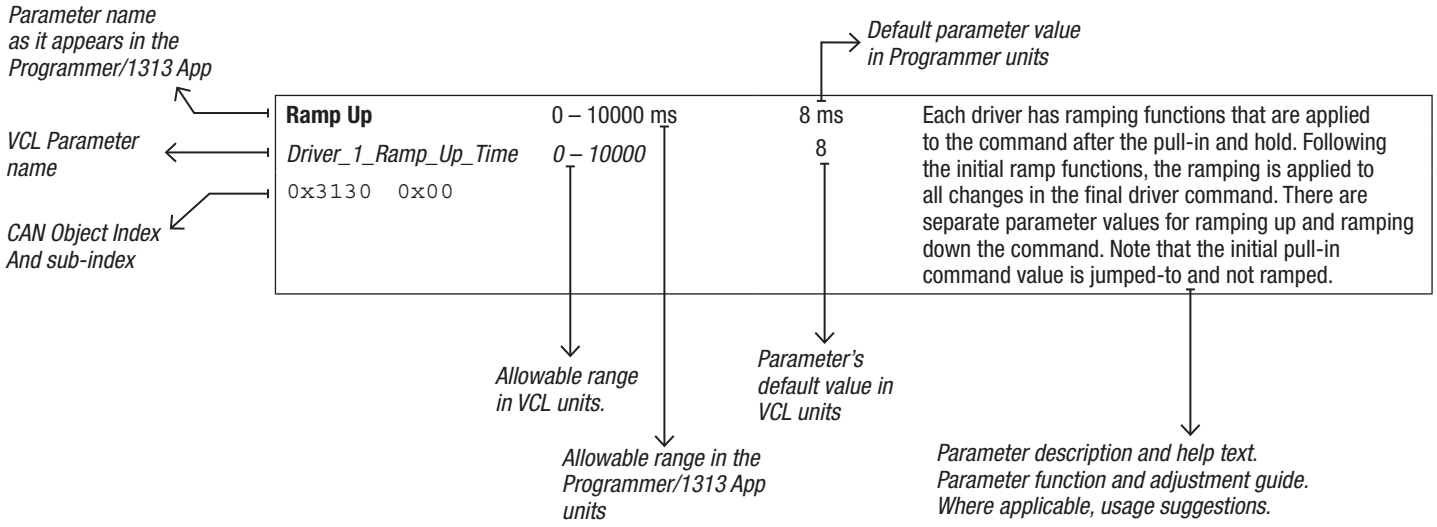
PROGRAMMING MENUS

The programmable parameters are grouped into nested hierarchical menus, as shown in Table 19. Each menu section has a leading lookup table to locate individual parameters.

Each parameter follows the menu chart format shown below. For every parameter, the menu chart contain the name, VCL name, and CAN Index. It gives the programming range and the default setting. Each description explains the parameter. This is where specific features and/or the parameter's affects are described in detail.

MENU CHART FORMAT

Individual parameters are presented as follows in the menu charts:



[1] *The Curtis Integrated Toolkit™ program is compatible with many leading USB ⇔CAN interface dongles from Peak, Kvaser, iFAC, Sondheim, etc.*

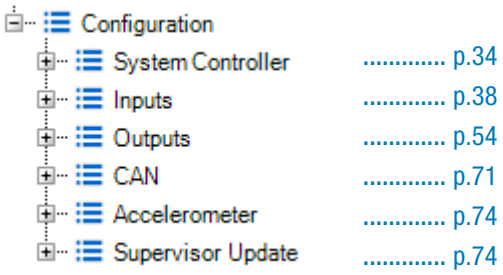
The CAN model 1313 handheld programmer (1313 HHP) is the blue band Curtis 1313-xx31 Handheld Programmer.

For technical support, and to obtain either of these programming tools, contact the Curtis distributor where you obtained the 1351 system controller or the Curtis sales-support office in your region.

See Appendix C for further information about the programming tools.

Note, the Parameter and Monitor variables that are repetitive in nature are described in the following tables by replacing their individual item "*number*" with an "*X*". Complex parameters may have their leading item (e.g., 1) as the example, with the subsequent items described using the '*X*" nomenclature. The programming tools, CIT and the 1313 HHP will always list each item, which has its own unique CAN Object Index, so the actual parameter-setup programming and monitoring will be item-by-item.

Table 19 Programmable Parameters Menus.



As displayed using the Curtis Integrated Toolkit™ or the 1313 HHP

Terminology

When setting parameters and commissioning the vehicle, follow these definitions.

CiA/CANopen	CAN in Automation (CiA) is the international users' and manufacturers' group for the CAN network (Controller Area Network), internationally standardized in the ISO 11898 series. CANopen is a CAN-based communication system. It was designed originally for motion-oriented machine control systems, such as handling systems. Today it is used in various application fields, such as medical equipment, off-road vehicles, maritime electronics, railway applications, or building automation. References include CiA 301, 303-1, etc.
Curtis Integrated Toolkit	The Curtis Integrated Toolkit™ (CIT) is a Curtis Instruments developed software toolkit for configuring and communicating with Curtis Instruments products. Use CIT to program (change and edit parameters, etc.) the 1351 System Controller.
Driver Types	A High-Side Driver "switches" positive voltage to the load, which is connected to ground. The driver is essentially the current source to the load. When the driver is "On", conventional current is sourced from the high-side driver through the load and then to ground, completing the circuit. A Low-Side Driver "switches" the load to ground. The load is always connected to positive voltage. The driver sinks current from the load, in essence. The driver has to be "On" to complete the load circuit.
F-Series	The Curtis F-series controller that are CANbus programmable (not serial programmable). For example, the AC-F2-A and AC F2-C controllers.
I/O	Input/Output. I/O generally refers to the controller 35-pin connector's input signals or switches, output signals, power, or low-side drivers. I/O GND (I/O Ground) is the controller's ground reference (pin).

Object Index	The object dictionary is essentially a table that stores configuration and process data. The CANopen standard defines a 16-bit bit index and an 8-bit sub-index. The object dictionary is the method by which a CANopen device communicates. Every 1351 parameter and monitor variable has its own unique CAN Object Index, which is listed in the parameter and monitor tables.
PDO	Process Data Objects (PDO). CANopen's real-time data transfer used for the fast transfer of 8 bytes of data or less without protocol overhead. The meaning of the data content is defined beforehand by a corresponding PDO mapping structure within the Device Object Dictionary. See the PDO Setup in Appendix A.
RPDO	Receive Process Data Object (RPDO). Data received by the Consumer from Producer communication, (e.g., the manger controller receives data from the ancillary controller).
RX	Receive. In CANopen, RX (Rx) is from the perspective of the salve controller.
TPDO	Transmit Process Data Object (TPDO). Data transmission by the PDO Producer to PDO Consumer, (e.g., the ancillary controller transmits data to the manger controller).
SDO	Service Data Object (SDO). A SDO is a low priority message used to transfer multiple data sets from a client to a server and vice versa. Several types of data transfer are available, with the Client (manger controller) taking the initiative for a transfer. The SDO process is used primarily to read or write to an object index of a Server (ancillary controller). A SDO is used for configuring the 1351 system controller via the CAN network. The contents of the data set are defined within the Object Dictionary.
SSDO	Server-SDO. Owner of the Object Dictionary (e.g., the ancillary controller). Server to Client communication.
TX	Transmit. In CANopen, TX (Tx) is from the perspective of the salve controller.
CSDO	Client-SDO. Client to Server communication. SDO Download: Through this service the client (i.e., the manger) of a SDO downloads data to the server (i.e., the salve, which is the owner of the Object Dictionary). SDO Upload: Through this service, the client of an SDO uploads (reads) data from the server.
VCL	VCL is the unique Curtis Vehicle Control language. VCL provides the application level programmability to customize the usage, or allow Curtis controllers to perform as 'vehicle managers' eliminating the need for costly, additional system controllers. See our website: Curtis Vehicle Control Language (VCL). Note, for migrating applicable VCL to the 1351, contact the Curtis distributor where you obtained your controller or the Curtis sales-support office in your region.

SDO Write Message

To retain parameter values in non-volatile memory (NVM) that are sent by standard CANopen SDO write messages (such as from a different controller or module), a non-zero value must be written to *SDO_NVM_Write_Enable* (Object Index 0x2008, sub-index 0x00) before changing parameter values. This will have the parameter changes written to non-volatile memory. The data length in SDO write commands must be specified (i.e., 23h, 27h, 2Bh, 2Fh).

For example, an the SDO message for saving parameter changes to NVM, using the default Node ID of 0x11:

611h 8 23 08 20 00 **01** 00 00 00 // "SDO NVM Write Enable"

Do not "leave" the controller in the "write" state, however. After completing the NVM write actions, turn it off (send the value of '0' ...).

Note that having the *SDO_NVM_Write_Enable* set to zero only saves the parameter changes to ephemeral (RAM) memory.

For compliant devices, optionally use the "Save" SDO command (0x1010) by writing the ASCII text string "save" to sub-index 01h which initiates a one-time complete storage of all updated parameters to EEPROM. The ASCII 's' = 73h, 'a' = 61h, 'v' = 76h, and 'e' = 65h. Verify before usage. For example:

611h | 23 10 10 01 73 61 76 65 // "save"

The CAN Objects 0x1010 and 0x2008 are not listed in the 1351 System Info.

To save the new parameters from a VCL program, use the *NVM_Write_Parameter()* function. (listed in Sys Info)

Note that parameter changes performed when using CIT and the 1313HHP programmers automatically force all changes to be stored into non-volatile memory regardless of the SDO NVM WRITE ENABLE setting.

THE PROGRAMMABLE PARAMETERS

The System Controller menu is where the 1351's voltage (Power), battery discharge indicator (BDI), 5 and 12-volt power supplies (External Supplies), and the 1351's Temperature parameters are located. Note that within the parameter menus are read-only monitor variables which are helpful when setting parameters and their affects. These read only variables are *italicized* in this chapter's menu index-tables (e.g., in the System Controller table, below).

SYSTEM CONTROLLER

SYSTEM CONTROLLER
POWER..... p. 35
—*Keyswitch Voltage*
—Nominal Voltage
—Max Voltage
—Min Voltage
—Brown Out Voltage
BDI..... p. 36
—*BDI Reading*
—Reset Volts Per Cell
—Full Volts Per Cell
—Empty Volts Per Cell
—Discharge Time
— BDI Reset Percentage
EXTERNAL SUPPLIES..... p. 37
—*Ext 5V*
—*Ext 12V*
—*Ext 5V Current*
—*Ext 12V Current*
—Ext 5V Enable
—Ext 12V Enable
TEMPERATURE..... p. 37
—*Module Temperature*
—Max Temp
—Min Temp

SYSTEM CONTROLLER — POWER

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Keyswitch Voltage *Keyswitch_Voltage* `0x331C 0x00`	0.0 – 140.0 V *0 – 14000*	Read Only	The measured voltage at the KSI input (pin 7).
Nominal Voltage *Nominal_Voltage* `0x3000 0x00`	12.0 – 140.0 V *1200 – 14000*	24.0 V *2400*	This parameter is limited to the System Controller's model voltage. It must be set to the vehicle's nominal battery pack voltage. Use this parameter as a reference for determining the Overvoltage and Undervoltage protection thresholds (see the Max and Min Voltage parameters, below). This sets the voltage that battery compensation functions will be based upon. See the defined nominal voltage in Table D in Appendix D.
Max Voltage *Max_Voltage* `0x3243 0x00`	24.0 – 100.0 V *2400 – 10000*	60.0 V *6000*	If the KSI exceeds this setting for >50ms, an Overvoltage fault is issued, an Emergency message sent and the drivers will be shutdown.
Min Voltage *Min_Voltage* `0x3244 0x00`	0 – 24.0* V *0 – 2400*	0.0 V *0*	If the KSI falls below this setting for >50ms, an Undervoltage fault is issued, an Emergency message sent and the drivers will be shutdown. The Undervoltage set point must be set at or above the Brownout Voltage setting below.
Brown Out Voltage *Brown_Out_Voltage* `0x3001 0x00`	6.5* – 100.0 V *650 – 10000*	6.5 V *0*	Below this voltage, the system will start the brown out timer and start saving NVM parameters. The Drivers will shut down. Note: Reducing this value to below the model's minimum (default) will result in not saving the nvuser variables.

** Depends on the Model, defaults shown are for the 12-36V model. The 36-96V model has a default of 15.0 Volts.*

It is recommended to set the 1351's brown out voltage parameter to no less than the model's minimum operating voltage (after startup).

Battery Discharge Indicator

The lead-acid battery discharge indicator (BDI) algorithm continuously calculates the battery state-of-charge (SOC) from the keyswitch voltage (KSI, pin 7) at power-up. The result of the BDI algorithm is the monitor variable BDI Reading, which is the state-of-charge percentage. When KSI is turned off, the present BDI percentage is stored in nonvolatile memory.

For flooded lead-acid batteries and sealed maintenance-free lead-acid batteries, the standard values for volts per cell are as follows,

Parameter	Lead-Acid Battery Type	
	Flooded	Sealed
Reset Volts Per Cell	2.09	2.09
Full Volts Per Cell	2.04	2.04
Empty Volts Per Cell	1.73	1.90

Use these standard values for the battery's starting point in setting the reset, full, and empty volts-per-cell parameters.

Note: For non-lead-acid batteries, including Lithium-Ion battery packs, use the battery pack or cell manufacturer's approved Battery Management System (BMS) for determining the SOC.

SYSTEM CONTROLLER — BDI

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
BDI Reading *BDI_Percentage* `0x331D 0x00`	0 – 100 % 0 – 100	Read Only	Present BDI % reading. The Battery Discharge Indicator (BDI) is the battery State-of-Charge (SOC) as a percentage. The percentage is based upon lead-acid battery chemistry and the associated parameters in this BDI parameters menu.
Reset Volts Per Cell *BDI_Reset_Volts_Per_Cell* `0x3246 0x00`	0.900 – 3.000 V *900 – 3000*	2.090 V *2090*	The reset voltage level is checked only once when the keyswitch is first turned on. The BDI Reset Percent parameter also influences the algorithm that determines whether the BDI percentage is reset to 100 % at key-on. This Reset Volts Per Cell parameter should always be set higher than the Full Volts Per Cell parameter (see the chart, above). This parameter is applicable for lead-acid battery chemistry. The Reset Voltage Level = Reset Volts Per Cell × number of cells in the battery pack.*
Full Volts Per Cell *BDI_Full_Volts_Per_Cell* `0x3247 0x00`	0.900 – 3.000 V *900 – 3000*	2.040 V *2040*	This parameter sets the Keyswitch Voltage value that is considered to be 100 % state-of-charge. When a battery (under load) drops below this voltage, it begins to lose charge (lowering the BDI percentage). The Keyswitch Voltage variable is also viewable in the programmer's Monitor » System Controller menu. The Full Voltage Level = Full Volts Per Cell x number of cells in the battery pack.*
Empty Volts Per Cell *BDI_Empty_Volts_Per_Cell* `0x3248 0x00`	0.900 – 3.000 V *900 – 3000*	1.730 V *1730*	The empty voltage level sets the Keyswitch Voltage value that is considered to be 0 % state-of-charge. The Empty Voltage Level = Empty Volts Per Cell × number of cells in the battery pack.*
Discharge Time *BDI_Discharge_Time* `0x3249 0x00`	0 – 600 Minutes *0 – 600*	34 Min *34*	This parameter sets the minimum time for the BDI algorithm to count down the BDI Percentage from 100 % to 0 %. The BDI algorithm integrates the time the filtered keyswitch voltage is below the state of charge voltage level. When that cumulative time exceeds the Discharge Time/100, the BDI Percentage is decremented by one percentage point and a new state of charge voltage level is calculated. Set this parameter based upon the system's typical duty cycle over which the battery is discharged. The State of Charge Level = ((Full Voltage Level – Empty Voltage Level) × BDI Percentage/100) + Empty Voltage Level.
Reset Percent *BDI_Reset_Percent* `0x324A 0x00`	0 – 100 % *0 – 100*	75 % *75*	When a battery has a high BDI percentage, its float voltage at KSI On can sometimes cause false resets. The BDI Reset Percent parameter addresses this problem by allowing the user to define a BDI Percentage value above which the BDI Percentage variable will not reset. When KSI is first powered on the BDI Percentage variable will reset to 100 % only if (Keyswitch Voltage > Reset Voltage Level) and (BDI Percentage < BDI Reset Percent).

** To determine the number of cells in the battery pack, divide the Nominal Voltage setting (above) by the battery chemistry's nominal volts-per-cell. Lead-acid: 2.0 V/cell, nominal.*

SYSTEM CONTROLLER — EXTERNAL SUPPLIES

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Ext 5V *Ext_5V* `0x3325 0x00`	0.0 – 10.0 V *0 – 1000*	Read Only	The measured output voltage at the 5V supply (pin 10)
Ext 12V *Ext_12V* `0x3326 0x00`	0.0 – 20.0 V *0 – 2000*	Read Only	The measured output voltage at the 12V supply (pin 9)
Ext 5V Current *Ext_5V_Current* `0x3327 0x00`	0 – 1000 mA *0 – 1000*	Read Only	The measured output current at the 5V supply (pin 10) **Note, the combined load current between the +5V and +12V outputs cannot exceed 300 mA**
Ext 12V Current *Ext_12V_Current* `0x3328 0x00`	0 – 1000 mA *0 – 1000*	Read Only	The measured output current at the 12V supply (pin 9) **Note, the combined load current between the +5V and +12V outputs cannot exceed 300 mA**
Ext 5V Enable *Low_Power_5V_Mode* `0x300B 0x00`	0 – 1 (Off/On) *0 – 1*	0 *0*	0 = the 5V supply (pin 10) is Off 1 = the 5V supply (pin 10) is On
Ext 12V Enable *Low_Power_12V_Mode* `0x300A 0x00`	0 – 1 (Off/On) *0 – 1*	0 *0*	0 = the 12V supply (pin 9) is Off 1 = the 12V supply (pin 9) is On

SYSTEM CONTROLLER — TEMPERATURE

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Module Temperature *Module_Temperature* `0x3329 0x00`	–50.0 – 100.0°C *–500 – 1000*	Read Only	The internal temperature of the 1531 System Controller
Max Temp *Max_Temp* `0x3250 0x00`	–40.0 – 90.0°C *–400 – 900*	85.0°C *1000*	If the module temperature exceeds this setting, an Overtemperature fault is issued, an Emergency message sent and the drivers will be shutdown.
Min Temp *Min_Temp* `0x3251 0x00`	–40.0 – 90.0°C *–400 – 900*	–20.0°C *–200*	If the module temperature falls below this setting, an Undertemperature fault is issued, an Emergency message sent and the drivers will be shutdown.

INPUTS

INPUTS

SWITCHES..... p. 39
- —SW 1
  - —*Status*
  - —Open State
  - —Active Level
  - —Debounce
- . . . . .
- —SW 14

VIRTUAL SWITCHES..... p. 43
- —VSW 1
  - —*Status*
  - —Active Level
- . . . . .
- —VSW 11

ANALOG INPUTS..... p. 45
- —Analog 1
  - —*Value*
  - —VSW Status
  - —Filter Time Constant
  - —High Threshold
  - —Low Threshold
- . . . . .
- —Analog 11

POT INPUT..... p. 47
- —*Resistance*
- —*Wiper Position*
- —Type
- —Nominal Resistance
- —Tolerance

RTD INPUT..... p. 48
- —RTD 1
  - —*Resistance*
  - —*Value*
  - —RTD Enable
  - —RTD 1 Map
  - —RTD Resistance
    - —Point 1
    - —Point 2
    - —Point 3
    - —Point 4
  - —Output Value
    - —Point 1
    - —Point 2
    - —Point 3
    - —Point 4
- . . . . .
- —RTD 4

HIGH SPEED DIGITAL INPUTS..... p. 50
- —*Count*
- —*Frequency*
- —*Pulse Width*
- —Type

ENCODER INPUT ..... p. 51
- —Encoder 1
  - —*RPM*
  - —Type
  - —Direction
- —Encoder 2

Switches

All switch inputs have a pull-down resistor. Some have and additional selectable pull-up circuit and a few can be enabled to be floating (no pull-up circuit or pull-down resistor). In this section, the switch parameters are described by their common groupings, while Figure 4 illustrates their external wiring diagram and pinouts. Notice that,

- When the external switch will be closed-to-B+, select the Pull-down option.
- When the external switch will be closed-to-B-, select the Pull-up option.
- When the external switch is a voltage signal (Hall switch or a TTL driver from another module), select the Float option.

Switch Inputs 1 – 8 are the same type of input circuits. They share the same parameters and monitor variables, distinguished by their individual switch number as illustrated in the switches table (below).

Switch Inputs 9 – 12 are the same type of input circuits. They share the same parameters and monitor variables, distinguished by their individual switch number as illustrated in the switches table (below).

Based upon the switch parameter options described below, select the switch type that matches the application, then configure the given switch parameters to match the application wiring and VCL logic.

INPUTS — SWITCHES

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
SW1			
Status *Switch_1* `0x338B 0x00`	Off – On *0 – 1*	Read Only	Switch 1 Status: ON or OFF This is the same variable as in the monitor menu: *Monitor » Inputs » Switch Input » **Switch 1*** Note: This pin can be used as the PWM Driver (see Figure 4)
Open State *Switch_1_Open_State* `0x31F0 0x00`	Pull Down – Pull Up *0 = Pull-down* *1 = Pull-up*	Pull Down *0*	Select either the Pull-down or the Pull-up input option. Pull-Down means the controller's internal input has its default resistive circuit connected to I/O Ground (GND). Use this option when a "closed switch" will be connected to KSI (B+). Pull-up means the controller's internal input is pulled to approximately +5V. Use this option when a "closed switch" will be connected to I/O Ground (GND or B–). See the explanations, below, for the selection based upon the type of switch connection utilized and VCL logic. The pin's input impedance is as noted in Chapter 2, Switch (digital) Inputs.
Active Level *Switch_1_Active_Level* `0x3200 0x00`	Low – High *0 = Low* *1 = High*	High *1*	Select the active ON state of the input: • Low = ON (where Low means the input is GND, B–) • High = ON (where High means the input is KSI, B+) See the explanation, below, for selection based upon the type of switch connection utilized and VCL usage.
Debounce *Switch_1_Debounce* `0x3210 0x00`	0 – 1000 ms *0 – 1000*	0 ms *0*	Length of time that the switch state must be steady (active) before it is declared (valid). The ON and OFF state of the switch is de-bounced before being presented to VCL. This parameter sets the length of the de-bounce time. Each de-bounce period is 1 millisecond.

INPUTS — SWITCHES, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
SW2 through SW8: X = the Switch Number, which is part of the VCL parameters' name (see SW1 as the example)			
Status *Switch_X* Variable's CAN Index by Switch No. 2 = 0x338C 0x00 3 = 0x338D 0x00 4 = 0x338E 0x00 5 = 0x338F 0x00 6 = 0x3390 0x00 7 = 0x3391 0x00 8 = 0x3392 0x00	Off – On *0 – 1*	Read Only	Switch X Status: ON or OFF This is the same variable as in the monitor menu: *Monitor » Inputs » Switch Input » Switch X* Note: This pin can be used as the PWM Driver (see Figure 4)
Open State *Switch_X_Open_State* Parameter's CAN Index by Switch No. 2 = 0x31F1 0x00 3 = 0x31F2 0x00 4 = 0x31F3 0x00 5 = 0x31F4 0x00 6 = 0x31F5 0x00 7 = 0x31F7 0x00 8 = 0x31F8 0x00	Pull Down – Pull Up *0 = Pull-down* *1 = Pull-up*	Pull Down *0*	Select either the Pull-down or the Pull-up input option. See the description for SW1, above.
Active Level *Switch_X_Active_Level* Parameter's CAN Index by Switch No. 2 = 0x3201 0x00 3 = 0x3202 0x00 4 = 0x3203 0x00 5 = 0x3204 0x00 6 = 0x3205 0x00 7 = 0x3207 0x00 8 = 0x3208 0x00	Low – High *0 = Low* *1 = High*	High *1*	Select the active ON state of the input. See the description for SW1, above.
Debounce *Switch_X_Debounce* Parameter's CAN Index by Switch No. 2 = 0x3211 0x00 3 = 0x3212 0x00 4 = 0x3213 0x00 5 = 0x3214 0x00 6 = 0x3215 0x00 7 = 0x3217 0x00 8 = 0x3218 0x00	0 – 1000 ms *0 – 1000*	0 *0*	Length of time that the switch state must be steady (active) before it is declared (valid). See the description for SW1, above.

The voltage at the pin does not indicate if the switches 1 – 8 are open or closed (OFF or ON). The OFF or ON state depends upon how the switch is connected and configured. A "closed" switch can be connected to I/O Ground (B-) or it can be connected to KSI (B+).

- If the application is to consider B– or B+ to be the ON state, this is set using the Active State parameter.
- If the application is to consider an open input to be high or low, this is set using the Open State parameter.

Note that if the Open State is set to Pull-up, the external pin must switch to I/O Ground (B–) else the switch position (ON or OFF) cannot be sensed. Likewise, if the Open State is set to Pull-down, the external pin must switch to KSI (B+), else the switch position (ON or OFF) cannot be sensed.

INPUTS — SWITCHES, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Usage examples:
When the Open State is set to Pull-Up, and the Active State is set to Low, switching the input to ground (B–) will be seen in VCL as the switch is ON
When the Open State is set to Pull-Up and the Active State is set to High, switching the input to ground will be seen in VCL as the switch is OFF; note that opening the input or pulling the input to B+ will be seen as ON
When the Open State is set to Pull-Down and the Active State is set to High, switching the input to B+ will be seen in VCL as the switch is ON; note that opening the input or pulling the input to ground will be seen as OFF
When the Open State is set to Pull-Down and the Active State is set to Low, switching the input to B+ will be seen in VCL as the switch is OFF; note that opening the input or pulling the input to ground will be seen as ON
One input, two operations example:
; Reach-in and Reach-out = switch (SW7), configured as Open State = Pull-down and Active Level = High.
If (SW7 = OFF) ; Reach Switch is open. Or the wire is off (broken) at the input (fail safe = Forks In)
Driver_5_Command = 0 ; Stop the fork Reach Out driver (Driver 5)
Driver_4_Command = 1000 ; Force the forks in, Reach In driver (Driver 4) is ON. 1000 = 100% PWM
If (SW7 = ON) ; Reach Switch is closed (switched the KSI)
Driver_4_Command = 0 ; Stop the fork Reach In (Driver 4 = OFF)
Driver_5_Command = 1000 ; Reach Out active (Driver 5 = ON) is ON. 1000 = 100% PWM			

SW9			
Status *Switch_9* `0x3393 0x00`	0 – 1 (Off/On) *0 – 1*	Read Only	Switch X status: On = 1 or Off = 0 (pin 32) Same as in monitor: *Monitor » Inputs » Switch Input »* ***Switch 9*** Note: This pin (32) can be used as the PWM Driver 9 (see Figure 4)
Open State *Switch_9_Open_State* `0x31F9 0x00`	Pull Down/Pull Up/Float *0 = Pull-down* *1 = Pull-up* *2 = Float*	Pull Down *0*	Selectable Pull-down or Pull-up input. This input can also be set to a float state, utilizing neither the pull-up nor the pull-down circuits. The input impedance is as noted in Chapter 2, Switch (digital) Inputs.
Active Level *Switch_9_Active_Level* `0x3209 0x00`	Low/High *0 = Low* *1 = High*	High *1*	Set the active state of the input, whether low = On or High = On 0 = Low, 1 = High
Debounce *Switch_9_Debounce* `0x3219 0x00`	0 – 1000 ms *0 – 1000*	0 ms *0*	Length of time that the switch state must be steady (active) before it is declared (valid). The on and off state of the switch is de-bounced before being presented to VCL. This parameter sets the length of the de-bounce time. Each de-bounce period is 1 millisecond.
SW10 through SW12: X = the Switch Number, which is part of the VCL parameters' name (see SW9 as the example)			

Status
Switch_X
Variable's CAN Index by Switch No.
`10 = 0x3394 0x00`
`11 = 0x3395 0x00`
`12 = 0x3396 0x00`
	0 – 1 (Off/On)
	0 – 1
		Read Only	Switch X status: On = 1 or Off = 0
			Same as in monitor: *Monitor » Inputs » Switch Input »* ***Switch X***
			Note: This switch's pin can be used as the PWM Driver (see Figure 4)

Open State
Switch_X_Open_State
Parameter's CAN Index by Switch No.
`10 = 0x31FA 0x00`
`11 = 0x31F6 0x00`
`12 = 0x31FB 0x00`
	Pull Down/Pull Up/Float
	0 = Pull-down
	1 = Pull-up
	2 = Float
		Pull Down
		0
			Selectable Pull-down or Pull-up input. This input can also be set to a float state, utilizing neither the pull-up nor the pull-down circuits. The input impedance is as noted in Chapter 2, Switch (digital) Inputs.

INPUTS — SWITCHES, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Active Level
Switch_X_Active_Level
 	
Parameter's CAN Index by Switch No.
`10 = 0x320A 0x00`
`11 = 0x3206 0x00`
`12 = 0x320B 0x00`
	Low/High
	0 = Low
	1 = High
		High
		1
			Set the active state of the input, whether low = On or High = On
			0 = Low, 1 = High

Debounce
Switch_X_Debounce
 	
Parameter's CAN Index by Switch No.
`10 = 0x321A 0x00`
`11 = 0x3216 0x00`
`12 = 0x321B 0x00`
	0 – 1000 ms
	0 – 1000
		0 ms
		0
			Length of time that the switch state must be steady (active) before it is declared (valid).
			The on and off state of the switch is de-bounced before being presented to VCL. This parameter sets the length of the de-bounce time. Each de-bounce period is 1 millisecond.

SW 13 and 14: X = the Switch Number, which is part of the VCL parameters' name (similar to the examples, above)			

Status
Switch_X
 	
Variable's CAN Index by Switch No.
`13 = 0x3397 0x00`
`14 = 0x3398 0x00`
	0 – 1 (Off/On)
	0 – 1
		Read Only	Switch X status: On = 1 or Off = 0
			Same as in monitor: *Monitor » Inputs » Switch Input »* ***Switch X***
			Note: This is the Encoder A, 2A and 2B, in Figure 4

Open State
Switch_13_14_Open_State
 	
Parameter's CAN Index by Switch No.
Same
`13 = 0x31FC 0x00`
`14 = 0x31FC 0x00`
	Pull Down/Pull Up
	0 = Pull-down
	1 = Pull-up
		Pull Down
		0
			Select either the Pull-down or the Pull-up input option.
			Note that both Switch 13 and 14 must be set the same.
			See the description for SW1, above.

Active Level
Switch_X_Active_Level
 	
Parameter's CAN Index by Switch No.
`13 = 0x320C 0x00`
`14 = 0x320D 0x00`
	Low/High
	0 = Low
	1 = High
		High
		1
			Select the active ON state of the input:
			• Low = ON (where Low means the input is GND, B–)
			• High = ON (where High means the input is KSI, B+)

Debounce
Switch_X_Debounce
 	
Parameter's CAN Index by Switch No.
`13 = 0x321C 0x00`
`14 = 0x321D 0x00`
	0 – 1000 ms
	0 – 1000
		0 ms
		0
			Length of time that the switch state must be steady (active) before it is declared (valid).
			The on and off state of the switch is de-bounced before being presented to VCL. This parameter sets the length of the de-bounce time. Each de-bounce period is 1 millisecond.

Virtual Switches

The 1351 System Controller can process each of the analog input channels as a digital input switch by comparing the incoming voltage to corresponding high and low threshold parameters—resulting in a "virtual" Off or On switch. The resulting virtual switch (VSW) status is available to VCL.

- Virtual switches 1 – 4 share the encoder inputs.
- Virtual switches 5 – 8 are common to the RTD inputs.
- Virtual switches 9 & 10 are the pot high and wiper inputs.
- Virtual switch 11 shares the analog input at pin 22.

See the corresponding Analog Input Low and High Thresholds parameters, which set the high and low thresholds for the virtual switch inputs.

INPUTS — VIRTUAL SWITCHES

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
VSW 1 through 4: X = Virtual Switch Number which is part of the VCL parameters' name			
Status *Virtual_Switch_X* Variable's CAN Index by Switch No. 1 = 0x3380 0x00 2 = 0x3381 0x00 3 = 0x3382 0x00 4 = 0x3383 0x00	0 – 1 *0 – 1*	Read Only	Virtual Switch 1 status: On = 1 or Off = 0 (pin 1) The status is also available in the corresponding programmer's analog inputs menu and the monitor menu: Configuration » Inputs » Analog Inputs » Analog X:VSW Status *Monitor » Inputs » Switch Input » Virtual Switch X* X = 1, 2, 3, or 4 for Virtual Switches 1 – 4.
Active level *Virtual_Switch_X_Active_Level* Parameter's CAN Index by Switch No. 1 = 0x31D0 0x00 2 = 0x31D1 0x00 3 = 0x31D2 0x00 4 = 0x31D3 0x00	*Low – High* *0 – 1*	*High* *1*	When an analog input is used as virtual switch, set the active level to Low or High (here). Then set the corresponding Analog Input Low and High Thresholds parameters to match. See: *Programmer » Configuration » Inputs » Analog Inputs » Analog X*: *High Threshold* *Low Threshold.*
VSW 5 through 8: X = Virtual Switch Number which is part of the VCL parameters' name			
Status *Virtual_Switch_X* Variable's CAN Index by Switch No. 5 = 0x3384 0x00 6 = 0x3385 0x00 7 = 0x3386 0x00 8 = 0x3387 0x00	0 – 1 *0 – 1*	Read Only	Virtual Switch X status: On = 1 or Off = 0 X = 5, 6, 7, or 8 for Virtual Switches 5 – 8 The status is also available in the corresponding programmer's analog inputs menu and the monitor menu: Configuration » Inputs » Analog Inputs » Analog X:VSW Status *Monitor » Inputs » Switch Input »* ***Virtual Switch X*** Note: The pin can also be used as the AnalogX / RTDX input (see Figure 4). Chose one type of usage, only.
Open State *RTD_X_Enable* Parameter's CAN Index by Switch No. 5 = 0x3050 0x00 6 = 0x3051 0x00 7 = 0x3052 0x00 8 = 0x3053 0x00	Float – Pull Up *0 – 1*	Float *0*	Float = analog input as a virtual switch Pull Up = RTD Enabled X = 5, 6, 7, or 8 for Virtual Switches 5 – 8 Note that changing this parameter will change to the corresponding *RTD X* Enable parameter and vice versa. See: *Configuration\Inputs\RTD Input\RTDX\RTD Enable*

INPUTS — VIRTUAL SWITCHES, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Active level
Virtual_Switch_X_Active_Level
 	
Parameter's CAN Index by Switch No.
`5 = 0x31D4 0x00`
`6 = 0x31D5 0x00`
`7 = 0x31D6 0x00`
`8 = 0x31D7 0x00`
	Low – High
	0 – 1
		Low
		0
			When an analog input is used as *virtual switch*, set the active level to Low or High (here). Then set the corresponding Analog Input Low and High Thresholds parameters to match. See:
			Programmer » Configuration » Inputs » Analog Inputs » Analog X:
			High Threshold
			Low Threshold.
			X = 5, 6, 7, or 8 for Virtual Switches 5 – 8

VSW 9 and 10: X = Virtual Switch Number which is part of the VCL parameters' name			

Status
Virtual_Switch_X
 	
Variable's CAN Index by Switch No.
`9 = 0x3388 0x00`
`10 = 0x3389 0x00`
	0 – 1
	0 – 1
		Read Only	Virtual Switch X status: On = 1 or Off = 0
			The status is also available in the corresponding programmer's analog inputs menu and the monitor menu:
			Configuration » Inputs » Analog Inputs » Analog X:VSW Status
			Monitor » Inputs » Switch Input » Virtual Switch X
			X = 9, 10 for Virtual Switches 9 and 10

Open State
Virtual_Switch_X_Open_State
 	
Parameter's CAN Index by Switch No.
`9 = 0x31C4 0x00`
`10 = 0x31C5 0x00`
	Float – Pull Up
	0 – 1
		Float
		0
			Do not set the Open State to Pull-Up if this input is used for a Potentiometer connection
			X = 9, 10 for Virtual Switches 9 and 10

Active level
Virtual_Switch_9_Active_Level
 	
Parameter's CAN Index by Switch No.
`9 = 0x31D8 0x00`
`10 = 0x31D9 0x00`
	Low – High
	0 – 1
		High
		1
			When an analog input is used as *virtual switch*, set the active level to Low or High (here). Then set the corresponding Analog Input Low and High Thresholds parameters to match. See:
			Programmer » Configuration » Inputs » Analog Inputs » Analog X:
			High Threshold
			Low Threshold.
			X = 9, 10 for Virtual Switches 9 and 10

VSW 11			
Status Virtual_Switch_11 `0x338A 0x00`	0 – 1 *0 – 1*	Read Only	Virtual Switch 11 status: On = 1 or Off = 0 (pin 22) The status is also available in the corresponding programmer's analog inputs menu and the monitor menu: *Configuration » Inputs » Analog Inputs » Analog 11:VSW Status* *Monitor » Inputs » Switch Input » **Virtual Switch 11***
Active level *Virtual_Switch_11_Active_Level* `0x31DA 0x00`	Low – High *0 – 1*	Low *0*	When an analog input is used as a *virtual switch*, set the active level to Low or High (here). Then set the corresponding Analog Input Low and High Thresholds parameters to match. See: *Programmer » Configuration » Inputs » Analog Inputs » Analog 11:* *High Threshold* *Low Threshold.*

Analog Inputs

The 1351 System Controller processes each of the analog input channels and makes the value available to VCL. The parameter sets for each analog input are the same—although Analog 1-4 have a lower input voltage limit. The analog inputs share the same set of parameters and monitor variables, distinguished by their individual switch number as illustrated in the switches table (below). Their usages vary as illustrated in the Figure 4 wiring diagram and as described in the (shared) Virtual Switch parameter settings, above.

INPUTS — ANALOG INPUTS

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Analog 1 through 4: X = Analog Input Number which is part of the VCL parameters' name			

Value
Analog_X_Volts
 	
Variable's CAN Index by Analog Input:
`1 = 0x3300 0x00`
`2 = 0x3301 0x00`
`3 = 0x3302 0x00`
`4 = 0x3303 0x00`
	0.0 – 5.0 V
	0 – 500
		Read Only	The analog voltage at the specific analog input (See Figure 4)
			Analog Inputs 1 – 4 are limited to the 5V input.
			These inputs are also used as the encoder signals.
			X = 1, 2, 3, or 4 for the Analog Inputs 1-4: *Analog_X_Volts*

VSW Status
Virtual_Switch_X
 	
Variable's CAN Index by Analog Input:
`1 = 0x3380 0x00`
`2 = 0x3381 0x00`
`3 = 0x3382 0x00`
`4 = 0x3383 0x00`
	0 – 1
	0 – 1
		Read Only	Virtual Switch 1 (pin 1) status:
			0 = Off
			1 = On
			X = 1, 2, 3, or 4 for the Analog Inputs 1-4: *Virtual_Switch_X*

Filter Time Constant
Analog_Input_X_Filter_TC
 	
Parameter's CAN Index by Analog Input:
`1 = 0x3030 0x00`
`2 = 0x3031 0x00`
`3 = 0x3032 0x00`
`4 = 0x3383 0x00`
	4 – 3000 ms
	4 – 3000
		8 ms
		8
			Time to 63% of the full voltage range.
			X = 1, 2, 3, or 4 for the Analog Inputs 1-4: *Analog_Input_X_Filter_TC*

High Threshold
Analog_Input_X_High_Threshold
 	
Parameter's CAN Index by Analog Input:
`1 = 0x3020 0x00`
`2 = 0x3021 0x00`
`3 = 0x3022 0x00`
`4 = 0x3023 0x00`
	0.0 – 20.0 V
	0 – 2000
		3.0 V
		300
			For usage of the Virtual Switch input threshold.
			Voltages above this 'set point' are processed as = On
			Note: These input pins can also be used as the encoder (ENC) signals.
			X = 1, 2, 3, or 4 for the Analog Inputs 1-4: *Analog_Input_X_High_Threshold*

Low Threshold
Analog_Input_X_Low_Threshold
 	
Parameter's CAN Index by Analog Input:
`1 = 0x3010 0x00`
`2 = 0x3011 0x00`
`3 = 0x3012 0x00`
`4 = 0x3013 0x00`
	0.0 – 20.0 V
	0 – 2000
		1.0 V
		100
			For usage of the Virtual Switch input threshold.
			Voltages below this 'set point" are processed as = Off
			Note: These input pins can also be used as the encoder (ENC) signals.
			X = 1, 2, 3, or 4 for the Analog Inputs 1-4: *Analog_Input_X_Low_Threshold*

INPUTS — ANALOG INPUTS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Analog 5 through 11: X = Analog Input Number which is part of the VCL parameters' name			
Value *Analog_X_Volts* Variable's CAN Index by Analog Input: 5 = 0x3304 0x00 6 = 0x3305 0x00 7 = 0x3306 0x00 8 = 0x3307 0x00 9 = 0x3308 0x00 10 = 0x3309 0x00 11 = 0x330A 0x00	0.0 – 20.0 V *0 – 2000*	Read Only	The analog voltage at the specific analog input (See Figure 4) X = 5 – 11 for the Analog Input 5 – 11: *Analog_X_Volts*
VSW Status *Virtual_Switch_X* Variable's CAN Index by Analog Input: 5 = 0x3384 0x00 6 = 0x3385 0x00 7 = 0x3386 0x00 8 = 0x3387 0x00 9 = 0x3388 0x00 10 = 0x3389 0x00 11 = 0x338A 0x00	0 – 1 *0 – 1*	Read Only	Virtual Switch 1 (pin 1) status: 0 = Off 1 = On X = 5 - 11 for the Analog Inputs 5-11: *Virtual_Switch_X*
Filter Time Constant *Analog_Input_X_Filter_TC* Parameter's CAN Index by Analog Input: 5 = 0x3034 0x00 6 = 0x3335 0x00 7 = 0x3336 0x00 8 = 0x3337 0x00 9 = 0x3338 0x00 10 = 0x3339 0x00 11 = 0x333A 0x00	4 – 3000 ms *4 – 3000*	8 ms *8*	Time to 63% of the full voltage range. X = 5 – 11 for the Analog Input 5 – 11: *Analog_Input_X_Filter_TC*
High Threshold *Analog_Input_X_High_Threshold* Parameter's CAN Index by Analog Input: 5 = 0x3024 0x00 6 = 0x3325 0x00 7 = 0x3326 0x00 8 = 0x3327 0x00 9 = 0x3328 0x00 10 = 0x3329 0x00 11 = 0x332A 0x00	0.0 – 20.0 V *0 – 2000*	3.0 V *300*	For usage of the Virtual Switch input threshold. Voltages above this 'set point" are processed as = On Note the analog input's corresponding usage; *RTD 1, pin 16, Analog 5* *RTD 2, pin 17, Analog 6* *RTD 3, pin 18, Analog 7* *RTD 4, pin 19, Analog 8* *Wiper (potentiometer), pin 20, Analog 9* *Pot (potentiometer) High, pin 21, Analog 10* *Analog Out, pin 22, Analog 11* X = 5 – 11 for the Analog Input 5 – 11: *Analog_Input_X_High_Threshold*
Low Threshold *Analog_Input_X_Low_Threshold* Parameter's CAN Index by Analog Input: 5 = 0x3014 0x00 6 = 0x3315 0x00 7 = 0x3316 0x00 8 = 0x3317 0x00 9 = 0x3318 0x00 10 = 0x3319 0x00 11 = 0x331A 0x00	0.0 – 20.0 V *0 – 2000*	1.0 V *100*	For usage of the Virtual Switch input threshold. Voltages below this 'set point" are processed as = Off Note the analog input's corresponding usage; *RTD 1, pin 16, Analog 5* *RTD 2, pin 17, Analog 6* *RTD 3, pin 18, Analog 7* *RTD 4, pin 19, Analog 8* *Wiper (potentiometer), pin 20, Analog 9* *Pot (potentiometer) High, pin 21, Analog 10* *Analog Out, pin 22, Analog 11* X = 5 – 11 for the Analog Input 5 – 11: *Analog_Input_X_Low_Threshold*

INPUTS — POT INPUT

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Resistance *Pot_Resistance* `0x3319 0x00`	0 – 15000 Ohms *0 – 15000*	Read Only	The calculation of the wiper resistance to ground.
Wiper Position *Wiper_Position* `0x3318 0x00`	0 – 100 % *0 – 1000*	Read Only	The calculated position percentage from the 2-wire or 3-wire potentiometers.
Type *Pot_Type* `0x3040 0x00`	Voltage/3-Wire/2-Wire *0 – Voltage* *1 – 3 Wire Pot* *2 – 2 Wire Pot*	Voltage *0*	0 – Voltage: Analog 9 Input used as an analog voltage input 1 – 3-Wire: Analog inputs 9 & 10 used for a 3-wire pot wiper 2 – 2-Wire: Analog Input 9 used as a 2-wire pot. Note: if using pin 20 input as a virtual switch, set this parameter to voltage.
Nominal Resistance *Pot_Resistance_Nominal* `0x3041 0x00`	0 – 15000 Ohms *0 – 15000*	0	The resistance (Ohms) across the 2 or 3 wire potentiometer. This is the value for the 100% (maximum) position. *2-Wire: The external resistance between pin 20 and 8.* *3-Wire: The external resistance between pin 21 and 8.*
Tolerance *Pot_Resistance_Tolerance* `0x3042 0x00`	0 – 2000 Ohms *0 – 2000*	0	Sets the pot resistance variation of the potentiometer. The allowance beyond which a fault is declared.

RTD Input

The 1351's RTD 20kΩ resistance range will accommodate the many resistive sensing devices. The 1351 System Controller's 4-point parameter mapping of RTD inputs offers customized resistive values to specify a "value" (voltage) output for a given resistance, thus enabling a wide range of RTD usage for temperature, pressure, linear-motion, and similar applications. Four RTDs can be connected and each used in VCL (see Figure 4).

INPUTS — RTD INPUT

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
RTD 1 – 4: X = the RTD Input number which is part of the VCL parameters' name			
Resistance *RTD_X_R* Variable's CAN Index by RTD Input: 1 = 0x3310 0x00 2 = 0x3311 0x00 3 = 0x3312 0x00 4 = 0x3313 0x00	0 – 20000 Ohms *0 – 20000*	Read Only	The RTD's resistance at the corresponding voltage point. *Refer to the RTD X Map. This variable is also in the Monitor menu: Monitor/Inputs/RTD Input/RTDX/Resistance* X = 1, 2, 3, or 4 for the Resistance: *RTD_X_R*
Value *RTD_X_OUT* Variable's CAN Index by RTD Input: 1 = 0x3314 0x00 2 = 0x3315 0x00 3 = 0x3316 0x00 4 = 0x3317 0x00	– 32768 – 32767 *– 32768 – 32767*	Read Only	The "value" (voltage) at the RTD input corresponding to the RTD resistance point. The meaning of the output value depends on the type of RTD used: Pressure, Temperature, Etc. *Refer to the RTD X Map. This variable is also in the Monitor menu: Monitor/Inputs/RTD Input/RTDX/Resistance* X = 1, 2, 3, or 4 for the Value: *RTD_X_OUT*
RTD Enable *RTD_X_Enable* Parameter's CAN Index by RTD Input: 1 = 0x3050 0x00 2 = 0x3051 0x00 3 = 0x3052 0x00 4 = 0x3053 0x00	Float / Pull Up *0 – 1*	Float *0*	Pull Up = RTD Enabled at pin (RDT X) 0 = Float 1 = Pull Up X = 1, 2, 3, or 4 for the Value: *RTD_X_Enable*
RTD 1 – 4 Map: X = the RTD Map number which is part of the VCL parameters' name			
RTD Resistance: X = the RTD Resistance number which is part of the VCL parameters' name			
Point 1 *RTD_X_R1* Parameter's CAN Index by RTD Input: 1 = 0x3060 0x00 2 = 0x3080 0x00 3 = 0x30A0 0x00 4 = 0x30C0 0x00	0 – 20000 Ohms *0 – 20000*	0	The resistance of the RTD at the corresponding voltage point. Inputs below this **Point 1** are limited to "**Out 1**" (below). These resistive values MUST be in ascending/increasing order. *This parameter is the least (lowest) resistance of the 4 parameters.* X = 1, 2, 3, or 4 for the Value: RTD_X_R1
Point 2 *RTD_X_R2* Parameter's CAN Index by RTD Input: 1 = 0x3061 0x00 2 = 0x3081 0x00 3 = 0x30A1 0x00 4 = 0x30C1 0x00	0 – 20000 Ohms *0 – 20000*	0	The resistance of the RTD at the corresponding voltage point. This R-values map MUST be in ascending/increasing order. *This parameter is the next lowest resistance of the 4.* X = 1, 2, 3, or 4 for the Value: *RTD_X_R2*

INPUTS — RTD INPUT, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Point 3
RTD_X_R3
 	
Parameter's CAN Index by RTD Input:
`1 = 0x3062 0x00`
`2 = 0x3082 0x00`
`3 = 0x30A2 0x00`
`4 = 0x30C2 0x00`
	0 – 20000 Ohms
	0 – 20000
		0	The resistance of the RTD at the corresponding voltage point. This R-values map MUST be in ascending/increasing order.
			This parameter is the second highest resistance of the 4.
			
			X = 1, 2, 3, or 4 for the Value: *RTD_X_R3*

Point 4
RTD_X_R4
 	
Parameter's CAN Index by RTD Input:
`1 = 0x3063 0x00`
`2 = 0x3083 0x00`
`3 = 0x30A3 0x00`
`4 = 0x30C3 0x00`
	0 – 20000 Ohms
	0 – 20000
		0	The resistance of the RTD at the corresponding voltage point. Inputs above this **Point 4** are limited to "**Out 4**" (below).
			R values map in the MUST be in ascending/increasing order.
			This parameter is the highest (most) resistance of the 4 parameters.
			
			X = 1, 2, 3, or 4 for the Value: *RTD_X_R4*

Output Value: X = the RTD Output Value number which is part of the VCL parameters' name			

Point 1
RTD_X_OUT1
 	
Parameter's CAN Index by RTD Input:
`1 = 0x3070 0x00`
`2 = 0x3090 0x00`
`3 = 0x30B0 0x00`
`4 = 0x30D0 0x00`
	– 32768 – 32767
	– 32768 – 32767
		0	The "value" output for the corresponding **Point 1** resistance (above). The meaning of the output value depends on the type of RTD used: Pressure, Temperature, Etc. (datasheet), or as a customized sensor response.
			X = 1, 2, 3, or 4 for the Value: *RTD_X_OUT1*

Point 2
RTD_X_OUT2
 	
Parameter's CAN Index by RTD Input:
`1 = 0x3071 0x00`
`2 = 0x3091 0x00`
`3 = 0x30B1 0x00`
`4 = 0x30D1 0x00`
	– 32768 – 32767
	– 32768 – 32767
		0	The "value" output for the corresponding **Point 2** resistance (above). This is the voltage at the input that corresponds to the resistance. The meaning of the output value depends on the type of RTD used: Pressure, Temperature, Etc. (datasheet), or as a customized sensor response.
			X = 1, 2, 3, or 4 for the Value: *RTD_X_OUT2*

Point 3
RTD_X_OUT3
 	
Parameter's CAN Index by RTD Input:
`1 = 0x3072 0x00`
`2 = 0x3092 0x00`
`3 = 0x30B3 0x00`
`4 = 0x30D2 0x00`
	– 32768 – 32767
	– 32768 – 32767
		0	The "value" output for the corresponding **Point 3** resistance (above). This is the voltage at the input that corresponds to the resistance. The meaning of the output value depends on the type of RTD used: Pressure, Temperature, Etc. (datasheet), or as a customized sensor response.
			X = 1, 2, 3, or 4 for the Value: *RTD_X_OUT3*

Point 4
RTD_X_OUT4
 	
Parameter's CAN Index by RTD Input:
`1 = 0x3073 0x00`
`2 = 0x3093 0x00`
`3 = 0x30B2 0x00`
`4 = 0x30D3 0x00`
	– 32768 – 32767
	– 32768 – 32767
		0	The "value" output for the corresponding **Point 4** resistance (above). This is the voltage at the input that corresponds to the resistance. The meaning of the output value depends on the type of RTD used: Pressure, Temperature, Etc. (datasheet), or as a customized sensor response.
			
			X = 1, 2, 3, or 4 for the Value: *RTD_X_OUT4*

High Speed Digital Inputs

There are two high-speed digital inputs on the 1351. These inputs can be used to measure input frequency and pulse width or used as counters/accumulators. These inputs can also be used for a quadrature (encoder) input, but they cannot be used as both high-speed inputs and encoders simultaneously. Therefore, the function of each input (pins 14 and 15) must be setup prior to use by setting the following High Speed Digital Input parameters. See the Encoder 1 parameters, below, for this alternative usage.

INPUTS — HIGH SPEED DIGITAL INPUTS

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Count
HS_Input_X_Count
 	
Variable's CAN Index by Count Input:
`1 = 0x3422 0x00`
`2 = 0x3425 0x00`
	0 – 4294967295
	0 – 4294967295
		Read Only	Count is based on the **Type** parameter selection. The counter range is 0 – (2^{32}) – 1. It will overflow and reset to 0.
			X = 1 or 2 for the Count: *HS_Input_X_Count*
			---------- 	
			The counter function has a new VCL feature;
			Reset_Pulse_Counter(X)
			Description:
			Will reset the input counter to 0
			Parameter:
			X = the input counter to reset: 1 – 2
			Pin 14, X = 1
			Pin 15, X = 2
			Note, this is the same variable in the Monitor menu:
			Monitor\Inputs\High Speed Digital Input X\Count

Frequency
HS_Input_X_Frequency
 	
Variable's CAN Index by Frequency Input:
`1 = 0x3420 0x00`
`2 = 0x3423 0x00`
	0 – 50000 Hz
	0 – 50000
		Read Only	Measure the frequency of the incoming signal pulses.
			The frequency measurement is derived from the input signal's rising to rising edge.
			X = 1 or 2 for the Frequency: *HS_Input_X_Frequency*
			Note, this is the same variable in the Monitor menu:
			Monitor\Inputs\High Speed Digital Input X\Frequency

Pulse Width
HS_Input_X_Pulse_Wdith
 	
Variable's CAN Index by Pulse Width Input:
`1 = 0x3421 0x00`
`2 = 0x3424 0x00`
	0.0 – 100 %
	0 – 1000
		Read Only	This is the % width of either the high or low pulse, depending on the Type setting.
			X = 1 or 2 for the Frequency: *HS_Input_X_Pulse_Width*
			Note, this is the same variable in the Monitor menu:
			Monitor\Inputs\High Speed Digital Input X\Pulse Width

Type
High_Speed_Input_X_Type
 	
Parameter's CAN Index by Type Input:
`1 = 0x322B 0x00`
`2 = 0x322C 0x00`
	None / *0*
	Frequency / *1*
	Width – High / *2*
	Width – Low / *3*
	Count – Rising / *4*
	Count – Falling / *5*
		0	The **Type** selection will configure this High Speed Input to return the monitor variables Count, Frequency, or Pulse Width basis.
			X = 1 or 2 for the Frequency: *High_Speed_Input_X_Type*
			None = 0: Can be used for encoders. The Count, Frequency, or Pulse Width variables will not be calculated. Do not use these any frequency, count or width readings if this Type parameter is set to None.
			Values of 1-5 will set this input accordingly, where one-channel of the Encoder Input will be used (as long as the corresponding Encoder X Type is set to None. See the Encoder Inputs section).
			Frequency = 1: Always measured from the incoming signal's rising edge to rising edge, in Hertz (number of times per second). Each pulse/rise/drop is "1" count.
			Width – High = 2: Measures the Pulse Width of the signals high portion.
			Width – Low = 3: Measures the Pulse Width of the signals low portion.
			Counter – Rising: Count triggers on each rising edge of a pulse.
			Counter - Falling: Count triggers of each falling edge of a pulse

Encoder Input

The 1351 can connect and process two different position sensors inputs (ENC1 A/B and ENC2 A/B), with each encoder having two inputs (an A and B). Within this manual, the generic term '*encoder*' is used, yet the actual sensor can be either a Quadrature encoder, a Sine/Cosine sensor, or Sawtooth device. The two encoders, labeled Encoder 1 and Encoder 2, can be different. For example, in the wiring diagram, Figure 4, ENC1 A/B (pins 14 and 15) can be a quadrature encoder, while ENC2 A/B (pins 1 and 2) can be a Sine/Cosine sensor. Each encoder requires its own independent wiring and parameter setup. Both provide independent 0-360^0 position and RPM measurement. See Chapter 2, Encoder Inputs, for more information on these encoder/sensor types and 1351 usage.

The type of *encoder* wired to the 1351 must be selected and setup using the following parameters. The specific encoder parameter menus are context enabled, thus they are visible in CIT/1313 HHP based upon the "Type" selection. A quadrature encoder will display a different set of parameters than a Sine/Cosine sensor, for example.

Note that the 1351 is not a motor controller, so use VCL to process the resulting Position and RPM monitor variables.

INPUTS — ENCODER INPUT

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

RPM
Encoder_X_RPM
 	
Variable's CAN Index by RPM:
`1 = 0x3410 0x00`
`2 = 0x3412 0x00`
	– 2147483648 – 2147483647
	– 2147483648 – 2147483647
	{i.e., $-(2^{31})$ – $[(2^{31})-1]$}
		Read Only	Encoder X's speed in revolutions per minute (rpm).
			Motor speed based only on encoder channel A pulses
			X = 1 or 2 for the RPM: *Encoder_X_RPM*
			Positive RPM = Forward rotation
			Negative RPM = Reverse rotation
			See the Direction parameter, below.

Type
Encoder_X_Type
 	
Parameter's CAN Index by Type:
`1 = 0x3220 0x00`
`2 = 0x3230 0x00`
	None
	Quadrature
	Sin/Cos
	Sawtooth
	0
	1
	2
	3
		None
		0
			Set this parameter to the type of position feedback device connected to the input pins.
			The selections is Enumeration CIT/1313:
			None = *0*
			Quadrature = *1*
			Sin/Cos = *2*
			Sawtooth = *3*
			X = 1 or 2 for the Type: *Encoder_X_Type*
			Note, When an encoder is used for these inputs, the high-speed input options are not available, as described in the previous High Speed Digital Inputs section.

Direction
Encoder_X_Dir
 	
Parameter's CAN Index by Direction:
`1 = 0x3222 0x00`
`2 = 0x3232 0x00`
	A before B
	B before A
	0
	1
		A before B
		0
			Configure the positive direction of motion.
			X = 1 or 2 for the Type: *Encoder_X_Dir*
			The selection is Enumeration CIT/1313:
			The rising edge on A before B is forward (positive rotation) = 0
			The rising edge of B before A is forward (positive rotation) = 1

INPUTS — ENCODER INPUT, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Quadrature Encoder		This menu is Context Enabled: It is visible when **Type = Quadrature Encoder**	

Pull Up Enable
Encoder_1_Pull_Up
`0x322D 0x00`
Note, Encoder 2 VCL name:
Switch_13_14_Open_State
`0x31FC 0x00`
	0 – 1/
	Pull Down – Pull Up
	0 – 1
		0	Enables the encoder channels A and B 2k pull-up resistors to +5V. Used to interface with encoders with open collector outputs.
			0 = Off = Pull Down (not pulled up to +5V)
			1 = On = Pull Up (pulled-up to +5V w/2k pull-up resistor)

Steps
Encoder_X_PPR
Parameter's CAN Index by Steps:
`1 = 0x3221 0x00`
`2 = 0x3231 0x00`
	12 – 1024
	12 - 1024
		60	Sets the number of encoder pulses per revolution. This parameter must be set to match the quadrature encoder.
			X = 1 or 2 for the Type: *Encoder_X_PPR*

Sine/Cosine Encoder		This menu is Context Enabled: It is visible when **Type = Sine/Cosine Encoder**	
Note, the sin/cos sensor physical waveforms are not bipolar, but center around an offset voltage, typically around 2.5v. The Peak-Peak voltage swing may be as small a 1volt or up to 2.5 volts. If the sensor voltages go outside this range, position sensing will be faulty and not useable by the VCL functions. In this case, a fault will be detected.			

Sin Min Voltage
Sin_X_Min
Parameter's CAN Index for *Sin_X_Min:*
`1 = 0x3223 0x00`
`2 = 0x3233 0x00`
	0.50 – 4.50 V
	50 – 450
		3.00
		300
			The lowest voltage that the sine waveform normally reaches.
			X = 1 or 2 for the Type: *Sin_X_Min*

Sin Max Voltage
Sin_X_Max
Parameter's CAN Index for *Sin_X_Max:*
`1 = 0x3225 0x00`
`2 = 0x3235 0x00`
	0.50 – 4.50 V
	50 – 450
		3.00
		300
			The highest voltage that the sine waveform normally reaches.
			X = 1 or 2 for the Type: *Sin_X_Max*

Cos Max Voltage
Cos_X_Max
Parameter's CAN Index for *Cos_X_Max:*
`1 = 0x3224 0x00`
`2 = 0x3236 0x00`
	0.50 – 4.50 V
	50 – 450
		3.00
		300
			The highest voltage that the cosine waveform normally reaches.
			X = 1 or 2 for the Type: *Cos_X_Max*

Cosine Min Voltage
Cos_X_Min
Parameter's CAN Index for *Cos_X_Min:*
`1 = 0x3226 0x00`
`2 = 0x3234 0x00`
	0.50 – 4.50 V
	50 – 450
		3.00
		300
			The lowest voltage that the cosine waveform normally reaches.
			X = 1 or 2 for the Type: *Cos_X_Min*

INPUTS — ENCODER INPUT, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Saw Encoder		This menu is Context Enabled: It is visible when **Type = Saw Encoder**	
Sawtooth Max/Min voltage signal: If this analog signal goes outside the specified ranges for >60ms, a fault is declared.			

Saw Encoder Max Voltage
Saw_Enc_X_Max_Voltage
 	
Parameter's CAN Index for *Saw_Enc_X_Max_Voltage*
`1 = 0x3229 0x00`
`2 = 0x3239 0x00`
	0.0 – 10.0 V
	0 – 1000
		0.0
		0
			Sets the highest voltage that is normally output by the sawtooth encoder.
			X = 1 or 2 for the Type: *Saw_Enc_X_Max_Voltage*

Saw Encoder Min Voltage
Saw_Enc_X_Min_Voltage
 	
Parameter's CAN Index for *Saw_Enc_X_Min_Voltage*
`1 = 0x3228 0x00`
`2 = 0x3238 0x00`
	0.0 – 10.0 V
	0 – 1000
		0.0
		0
			Sets the lowest voltage that is normally output by the sawtooth encoder.
			X = 1 or 2 for the Type: *Saw_Enc_X_Min_Voltage*

Saw Encoder Offset
Saw_Enc_X_Offset
 	
Parameter's CAN Index for *Saw_Enc_X_Offset*
`1 = 0x3227 0x00`
`2 = -x3237 0x00`
	0.0 – 10.0 V
	0 – 1000
		0.0
		0
			Sets the voltage, which is interpreted as the 0 degree point on channel A.
			X = 1 or 2 for the Type: *Saw_Enc_X_Offset*
			If the position calculated using the channel A is outside a tolerance parameter compared to that of channel B for >60ms, a fault is declared

Saw Encoder Tracking Tolerance
Saw_Enc_X_Tracking_Tolerance
 	
Parameter's CAN Index for *Saw_Enc_1_Tracking_Tolerance*
`1 = 0x322A 0x00`
`2 = 0x323A 0x00`
	0 – 360 degrees
	0 – 3600
		0 degrees	Sets the voltage, below which, a fault is declared. Must exceed this value for >60ms.
			X = 1 or 2 for the Type: *Saw_Enc_X_Tracking_Tolerance*

OUTPUTS

OUTPUTS

PWM DRIVERS..... p. 55

—Driver 1
—*PWM*
—*Current*
—Control Mode
—Ramp Up
—Ramp Down
—Initial Level
—Initial Time
—Fault Check
—Test
—*Command*
. . . . .
—Driver 10

HALF-BRIDGE DRIVERS..... p. 64

—Driver 11
—PWM
—Current
—Control Mode
—Type
—Ramp Up
—Ramp Down
—Initial Level
—Initial Time
—Test
—Command
. . . . .
—Driver 12

DIGITAL DRIVERS..... p. 68

—*Digital Out 1 State*
. . . . .
—*Digital Out 3 State*
—Digital Out 1 Mode
. . . . .
—Digital Out 3 Mode
—Test
—Digital Out 1 command
. . . . .
—Digital Out 3 command

SAFETY OUTPUT..... p. 69

—*Safety Output State*
—Safety Out Command

ANALOG OUTPUT..... p. 70

—*Output Voltage*
—Command

PWM Drivers

The ten 1351 Drivers are PWM controlled and have current measurement, output state monitoring and fault detection functions. Each driver can operate in one of the following modes: Open, Direct PWM, Voltage Compensated and Constant Current. All drivers operate at >15 kHz PWM. Drivers are VCL (or CAN message) commanded by directly writing to the variable *Driver_X_Command*, where the value range of 0 – 1000 corresponds to 0.0% to 100.0%. For example,

```
Driver_1_Command = 500 ;50% command

0x3360 0x00
```

In any active driver mode, the driver command, PWM and current can be monitored.

If the Mode is changed while there is a non-zero *Driver_X_Command*, then the Driver PWM is shut off immediately (no ramping) and is held off until the *Driver_X_Command* is set to zero. After this, the new mode is in effect and the *Driver_X_Command* can be changed (to a non-zero value).

Driver 10 has the added selection of a Frequency Output mode. This function sets up the PWM Driver10 (pin 33) output to yield a frequency proportional to the input (source) variable at an execution rate of 16 ms. When this option is active, Driver 10's frequency is independent of Drivers 1-9 PWM frequency (above). This output can be used to drive an electronic speedometer or tachometer. This option has the two specific VCL functions, *Automate_Frequency_Output()*, and *Disable_Frequency_Output()*.

OUTPUTS — PWM DRIVERS

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Driver 1 through 4 X = 1 – 4 for the Driver number (X) which is part of the VCL parameters' and monitor-variable name			
PWM *Driver_X_PWM* Variable's CAN Index by *Driver_X_PWM* 1 = 0x3370 0x00 2 = 0x3371 0x00 3 = 0x3372 0x00 4 = 0x3373 0x00	0.0 – 100.0 % *0 – 1000*	Read Only	The driver's Pulse Width Modulation (PWM) percentage. X= 1 – 4 for the VCL name *Driver_X_PWM* 100% equates to the driver being fully ON, meaning 100 percent of the Coil Return voltage (B+) is applied to the load (e.g., a contactor coil). 50% equates to an applied voltage of 50% 0 % turns off the driver. This variable is also located in Monitor: *Monitor\Outputs\PWM Drivers\Driver X\PWM*
Current *Driver_X_Current* Variable's CAN Index by *Driver_X_Current* 1 = 0x3330 0x00 2 = 0x3331 0x00 3 = 0x3332 0x00 4 = 0x3333 0x00	0.000 – 4.000 Amps *0 – 4000*	Read Only	The Amperage through Driver X X= 1 – 4 for the VCL name *Driver_X_Current* This variable is also located in Monitor: *Monitor\Outputs\PWM Drivers\Driver X\Current*

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Control Mode
Driver_X_Mode
 	
Parameter's CAN Index by *Driver_X_Mode*
`1 = 0x30E0 0x00`
`2 = 0x30E1 0x00`
`3 = 0x30E2 0x00`
`4 = 0x30E3 0x00`
	Off
	Direct PWM
	Voltage Comp
	Current Cntrl
	0
	1
	2
	3
		Off
		0
			Sets the mode for Driver X
			X= 1 – 4 for the VCL name *Driver_X_Mode*
			0 = Off (Open mode)
			This mode is used when the driver output is disabled to allow the pin to be used as an Input (i.e., Switch 1)
			1 = Direct PWM (normal PWM percentage operation)
			The driver's FET and wiring diagnostics are performed in this mode.
			2 = Voltage Comp (Voltage Compensated)
			Regulates the driver PWM based on the command, the nominal voltage setting and the present battery voltage in an attempt to provide a constant average voltage at the output. As the battery voltage drops, the PWM percentage will increase to compensate for the lower voltage. The driver's FET and wiring diagnostics are performed in this mode. The initial and continuous PWM percent timing control is also applied in this driver control mode.
			3 = Current Cntrl (Current Control)
			Regulates the current through the driver. Typically used to control Proportional Valves (PV). This mode interprets the command as a load current request to regulate the PWM (via a PI controller) and thus the requested load current.

Ramp Up
Driver_X_Ramp_Up_Time
 	
Parameter's CAN Index by *Driver_X_Ramp_Up_Time*
`1 = 0x3130 0x00`
`2 = 0x3131 0x00`
`3 = 0x3132 0x00`
`4 = 0x3133 0x00`
	0 – 10000 ms
	0 – 10000
		8 ms
		8
			The time in milliseconds to go from the minimum to the maximum control mode type, as a linear ramp-up.
			X= 1 – 4 for the VCL name *Driver_X_Ramp_Up_Time*
			Direct PWM: This is the time (duration) to change the pulse width modulation percentage
			Voltage Comp: This is the time (duration) to change the applied average voltage.
			Current Cntrl: This is time (duration) to change the current between the minimum and maximum driver current setting.
			A value of 0 milliseconds disables any ramping, resulting in a step-function-change when the driver command increases.
			Note that this *Ramp Up* is not applied to the Initial Level parameter.

Ramp Down
Driver_X_Ramp_Down_Time
 	
Parameter's CAN Index by *Driver_X_Ramp_Down_Time*
`1 = 0x3140 0x00`
`2 = 0x3141 0x00`
`3 = 0x3142 0x00`
`4 = 0x3143 0x00`
	0 – 10000 ms
	0 – 10000
		8 ms
		8
			The time in milliseconds to go from the minimum to the maximum control mode type, as a linear ramp-down.
			X= 1 – 4 for the VCL name Driver_X_Ramp_Down_Time
			Direct PWM: This is the pulse width percentage.
			Voltage Comp: This is nominal voltage percentage.
			(Note, Coil Return = B+ voltage)
			Current Cntrl: This is percentage between the maximum and minimum driver current setting
			(See the Current Control parameters menu, below)
			A value of 0 millisecond disables any ramping resulting in a step-function-change when the driver command decreases or is stopped (Driver_1_Command = 0).

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Initial Level
Driver_X_Hold_Level
 	
Parameter's CAN Index by *Driver_X_Hold_Level*
`1 = 0x3150 0x00`
`2 = 0x3151 0x00`
`3 = 0x3152 0x00`
`4 = 0x3153 0x00`
	0.0 – 100.0 %
	0 – 1000
		0.0 %
		0
			The percentage command for the Initial Hold Level.
			X= 1 – 4 for the VCL name *Driver_X_Hold_Level*
			This is the percentage the driver will initially "jump to" without ramping. This is affectedly the traditional contactor pull-in parameter. For example, a high pull-in ensures contactor contact-tips closure or a fast EM Brake release.
			Direct PWM: This is the pulse width percentage.
			Voltage Comp: This is nominal voltage percentage. (Note, Coil Return = B+ voltage)
			Current Cntrl: This is percentage between the maximum and minimum driver current setting. (See the Current Control parameters menu, below)
			Note that the Initial level may be above or below the applied command.

Initial Time
Driver_X_Hold_Time
 	
Parameter's CAN Index by *Driver_X_Hold_Time*
`1 = 0x3160 0x00`
`2 = 0x3161 0x00`
`3 = 0x3162 0x00`
`4 = 0x3163 0x00`
	0 – 10000 ms
	0 – 10000
		0 ms
		0
			The time the Initial Level is applied.
			X= 1 – 4 for the VCL name *Driver_X_Hold_Time*
			This parameter allows customization of the driver's initial-level parameter duration, before any ramping to the longer-term command percentage.

Fault Check
Driver_X_Checks_Enable
 	
Parameter's CAN Index by *Driver_X_Checks_Enable*
`1 = 0x31B0 0x00`
`2 = 0x31B1 0x00`
`3 = 0x31B2 0x00`
`4 = 0x31B3 0x00`
	Off
	Static
	
	0
	1
		0
		0
			Off = 0
			Static = 1
			X= 1 – 4 for the VCL name *Driver_X_Checks_Enable*
			When ON, the 1351 will check the voltage at the output pin to see if it matches the current status of the driver. If the driver is Off, the output pin should read near battery voltage, otherwise, there is a fault from the load being disconnected/open or the FET being shorted

Current Control		This menu is Context Enabled: Visible only when **Control Mode = Current Cntrl**	

Max Current
Driver_X_Max_Current
 	
Parameter's CAN Index by *Driver_X_Max_Current*
`1 = 0x3110 0x00`
`2 = 0x3111 0x00`
`3 = 0x3112 0x00`
`4 = 0x3113 0x00`
	0 – 3000 mA
	0 – 3000
		0 mA
		0
			Maximum current that the 100% command will effectuate.
			X= 1 – 4 for the VCL name *Driver_X_Max_Current*
			Reference the *Driver_X_Command*.

Min Current
Driver_X_Min_Current
 	
Parameter's CAN Index by *Driver_X_Min_Current*
`1 = 0x3120 0x00`
`2 = 0x3121 0x00`
`3 = 0x3122 0x00`
`4 = 0x3123 0x00`
	0 – 3000 mA
	0 – 3000
		0 mA
		0
			Minimum current that a 0.1% command will effectuate. A value of 0 will turn off the driver.
			X= 1 – 4 for the VCL name *Driver_X_Min_Current*
			Reference the *Driver_X_Command*.

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Dither Period
Driver_X_Dither_Period
 	
Parameter's CAN Index by *Driver_X_Dither_Period*
`1 = 0x3180 0x00`
`2 = 0x3181 0x00`
`3 = 0x3182 0x00`
`4 = 0x3183 0x00`
	2 – 1000 ms
	2 – 1000
		1000 ms
		1000
			Periodicity (frequency) of the applied dither.
			X= 1 – 4 for the VCL name *Driver_X_Dither_Period*
			Dither provides a small amount of cyclically changing current to vibrate the solenoid and thus keep it from "sticking" in one position. Without dither, it is harder to make small adjustment in the proportional valve position. Dither has both a frequency and amount.

Dither Amount
Driver_X_Dither_Amount
 	
Parameter's CAN Index by *Driver_X_Dither_Amount*
`1 = 0x3170 0x00`
`2 = 0x3171 0x00`
`3 = 0x3172 0x00`
`4 = 0x3173 0x00`
	0 – 100 %
	0 – 32767
		0 %
		0
			The percent of the Driver_1_Max_Current parameter that is applied above and below its set point.
			X= 1 – 4 for the VCL name *Driver_X_Dither_Amount*
			Note, the driver measures current just before the PWM is shut off, therefore there is a lower limit where the current cannot be read. For the low side drivers, this is 8% (for HB drivers in low-side mode, this is 12%). The system designer should make sure that the load resistance, the coil return voltage supply, and minimum current will remain valid keeping the driver PWM above these values for good regulation.
			0% disables the dither function.

Kp
Driver_X_Kp
 	
Parameter's CAN Index by *Driver_X_Kp*
`1 = 0x3190 0x00`
`2 = 0x3191 0x00`
`3 = 0x3192 0x00`
`4 = 0x3193 0x00`
	1 – 100 %
	20 – 2048
		8 %
		163
			The proportional section of the PI (Proportional/Integral) current regulator within the 1351 system controller.
			X= 1 – 4 for the VCL name *Driver_X_Kp*
			The Kp term is the proportional gain, which acts immediately upon the error between the command the actual system state. Higher gains will tend to drive that error lower. The Ki term (below) is the integral gain. The integral gain will accumulate the current droop and attempt to bring the Driver current droop back to 0 amperes. Higher Ki gains will react more strongly and quickly.
			Therefore, this parameter determines how aggressively the PI controller attempts to match the driver current command. Larger values provide tighter control.
			If the gain is set too high, oscillations can occur as the controller tries to control the Driver current. If it is set too low, the Driver current may behave sluggishly and be difficult to control.

Ki
Driver_X_Ki
 	
Parameter's CAN Index by *Driver_X_Ki*
`1 = 0x31A0 0x00`
`2 = 0x31A1 0x00`
`3 = 0x31A2 0x00`
`4 = 0x31A3 0x00`
	1 – 100 %
	20 – 2048
		8 %
		163
			Integral section of the PI (Proportional/Integral) current regulator (controller).
			X= 1 – 4 for the VCL name *Driver_X_Ki*
			The integral term (Ki) forces zero steady state error so the Driver will run at the commanded current in conjunction with the Kp parameter (above). Larger values provide tighter control.
			If the gain is set too high, oscillations can occur as the controller tries to control the Driver current. If it is set too low, the Driver may take a long time to approach the exact commanded current.

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Test			

Command
Driver_X_Command
 	
Variable's CAN Index by *Driver_X_Command*
`1 = 0x3360 0x00`
`2 = 0x3361 0x00`
`3 = 0x3362 0x00`
`4 = 0x3363 0x00`
	0.0 – 100.0 %
	0 – 1000
		0 %	The Driver command. It corresponds to the driver's control mode type setting. This is the steady-state input command, ramping, and initial levels are applied after and upon this command.
			X= 1 – 4 for the VCL name *Driver_X_Command*
			This is labeled as "Test" here because this is not a settable parameter that persist over a KSI cycle.
			When set in Programmer, this is a "test" function for operating a driver during system setup/development. Its value is not retained over a key or power cycle. It re-sets to 0% (off).
			Note that this "*Driver_X_Command*" is the command to use in VCL for operating a driver. It must be used in the VCL program to operate the driver(s) as the normal basis. The 1351 also requires the Safety Out Command (state) be initialized (= 1) else there will be no voltage at the Coil Return/Safety Output (pins 11 and 12). This is part of the minimum VCL needed to operate the 1351 controller.

Driver 5 through 10 X = 5 – 10 for the Driver number (X) which is part of the VCL parameters' and monitor-variable name			

PWM
Driver_X_PWM
 	
Variable's CAN Index by *Driver_X_PWM*
`5 = 0x3374 0x00`
`6 = 0x3375 0x00`
`7 = 0x3376 0x00`
`8 = 0x3377 0x00`
`9 = 0x3378 0x00`
`10 = 0x3379 0x00`
	0.0 – 100.0 %
	0 – 1000
		Read Only	The driver's PWM percentage.
			X= 5 – 10 for the VCL name *Driver_X_PWM*
			100% equates to the driver being fully ON, meaning 100 percent of the Coil Return voltage (B+) is applied to the load (e.g., contactor coil).
			50% equates to an applied voltage of 50%
			0 % turns off the driver.
			This variable is also located in Monitor:
			Monitor\Outputs\PWM Drivers\Driver X\PWM

Current
Driver_X_Current
 	
Variable's CAN Index by *Driver_X_Current*
`5 = 0x3334 0x00`
`6 = 0x3335 0x00`
`7 = 0x3336 0x00`
`8 = 0x3337 0x00`
`9 = 0x3338 0x00`
`10 = 0x3339 0x00`
	0.000 – 4.000 Amps
	0 – 4000
		Read Only	The Amperage through Driver X
			X= 5 – 10 for the VCL address *Driver_X_Current*
			This variable is also located in Monitor:
			Monitor\Outputs\PWM Drivers\Driver X\Current

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Control Mode
Driver_X_Mode
 	
Parameter's CAN Index by *Driver_X_Mode*
`5 = 0x30E4 0x00`
`6 = 0x30E5 0x00`
`7 = 0x30E6 0x00`
`8 = 0x30E7 0x00`
`9 = 0x30E8 0x00`
`10 = 0x30E9 0x00`
	Off
	Direct PWM
	Voltage Comp
	Current Cntrl
	
	0
	1
	2
	3
		Off
		0
			Sets the mode for Driver X
			X= 5 – 10 for the VCL name *Driver_X_Mode*
			
			0 = Off (Open mode)
			This mode is used when the driver output is disabled to allow the pin to be used as an Input (i.e., Switch 2)
			1 = Direct PWM (normal PWM percentage operation)
			The driver's FET and wiring diagnostics are performed in this mode.
			2 = Voltage Comp (Voltage Compensated)
			Regulates the driver PWM based on the command, the nominal voltage setting and the present battery voltage in an attempt to provide a constant average voltage at the output. As the battery voltage drops, the PWM percentage will increase to compensate for the lower voltage. The driver's FET and wiring diagnostics are performed in this mode. The initial and continuous PWM percent timing control is also applied in this driver control mode.
			3 = Current Cntrl (Current Control)
			Regulates the current through the driver. Typically used to control Proportional Valves (PV). This mode interprets the command as a load current request to regulate the PWM (via a PI controller) and thus the requested load current.
			Driver 10 only:
			4 = Frequency Output (PWM Frequency Control)
			Regulates the driver's PWM Frequency, independently from drivers 1-9 (which have a common frequency)
			Typically used to control electronic speedometers or tachometers.

Ramp Up
Driver_X_Ramp_Up_Time
 	
Parameter's CAN Index by *Driver_X_Ramp_Up_Time*
`5 = 0x3134 0x00`
`6 = 0x3135 0x00`
`7 = 0x3136 0x00`
`8 = 0x3137 0x00`
`9 = 0x3138 0x00`
`10 = 0x3139 0x00`
	0 – 10000 ms
	0 – 10000
		8 ms
		8
			The time in milliseconds to go from the minimum to the maximum control mode type, as a linear ramp-up.
			X= 5 – 10 for the VCL name *Driver_X_Ramp_Up_Time*
			
			Direct PWM: This is the time to change the pulse width modulation percentage
			Voltage Comp: This is the time to change the nominal voltage percentage.
			(Note, Coil Return = B+ voltage)
			Current Cntrl: This is time to change the current between the minimum and maximum driver current setting.
			
			A value of 0 milliseconds disables any ramping, resulting in a step-function-change when the driver command increases.
			Note that this *Ramp Up* is not applied in the Initial Level parameter.

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Ramp Down
Driver_X_Ramp_Down_Time
 	
Parameter's CAN Index by
Driver_X_Ramp_Down_Time
`5 = 0x3144 0x00`
`6 = 0x3145 0x00`
`7 = 0x3146 0x00`
`8 = 0x3147 0x00`
`9 = 0x3148 0x00`
`10 = 0x3149 0x00`
	0 – 10000 ms
	0 – 10000
		8 ms
		8
			The time in milliseconds to go from the minimum to the maximum control mode type, as a linear ramp-down.
			X= 5 – 10 for the VCL name *Driver_X_Ramp_Down_Time*
			
			Direct PWM: This is the pulse width percentage.
			Voltage Comp: This is nominal voltage percentage. (Note, Coil Return = B+ voltage)
			Current Cntrl: This is percentage between the maximum and minimum driver current setting (See the Current Control parameters menu, below)
			
			A value of 0 millisecond disables any ramping resulting in a step-function-change when the driver command decreases or is stopped (Driver_X_Command = 0). The ramp down duration is not applied when the 1351 is turned off (switched off).

Hold Level
Driver_X_Hold_Level
 	
Parameter's CAN Index by
Driver_X_Hold_Level
`5 = 0x3154 0x00`
`6 = 0x3155 0x00`
`7 = 0x3156 0x00`
`8 = 0x3157 0x00`
`9 = 0x3158 0x00`
`10 = 0x3159 0x00`
	0.0 – 100.0 %
	0 – 1000
		0.0 %
		0
			The percentage command for the Initial Hold Level.
			X= 5 – 10 for the VCL name *Driver_X_Hold_Level*
			This is the percentage the driver will initially "jump to" without ramping. This is affectedly the traditional contactor pull-in parameter. For example, a high pull-in ensures contactor contact-tips closure or a fast EM Brake release.
			
			Direct PWM: This is the pulse width percentage.
			Voltage Comp: This is nominal voltage percentage. (Note, Coil Return = B+ voltage)
			Current Cntrl: This is percentage between the maximum and minimum driver current setting. (See the Current Control parameters menu, below)
			
			Note that the Initial level may be above or below the applied command.

Hold Time
Driver_X_Hold_Time
 	
Parameter's CAN Index by
Driver_X_Hold_Level
`5 = 0x3164 0x00`
`6 = 0x3165 0x00`
`7 = 0x3166 0x00`
`8 = 0x3167 0x00`
`9 = 0x3168 0x00`
`10 = 0x3169 0x00`
	0 – 10000 ms
	0 – 10000
		0 ms
		0
			The time the Initial Level is applied.
			X= 5 – 10 for the VCL name *Driver_X_Hold_Time*
			This parameter allows customization of the driver's initial-level parameter duration, before any ramping to the longer-term command percentage.

Fault Check
Driver_X_Checks_Enable
 	
Parameter's CAN Index by
Driver_X_Hold_Level
`5 = 0x31B4 0x00`
`6 = 0x31B5 0x00`
`7 = 0x31B6 0x00`
`8 = 0x31B7 0x00`
`9 = 0x31B8 0x00`
`10 = 0x31B9 0x00`
	Off
	Static
	0
	1
		0
		0
			Off = 0
			Static = 1
			X= 5 – 10 for the VCL name *Driver_X_Checks_Enable*

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Pull Down Enable
Switch_X_Open_State
 	
Parameter's CAN Index by *Switch_X_Open_State*
`9 = 0x31B8 0x00`
`10 = 0x31B9 0x00`
	Pull Down
	Pull Up
	Float
	0
	1
	2
		Pull Down
		0
			Drivers 9 and 10 offer a "float" state, where the input is neither pulled-up to +5V nor pulled-down to ground. These two drivers float option are for interfacing with TTL device.
			X= 9 – 10 for the VCL name *Switch_X_Open_State*

Current Control		This menu is Context Enabled: Visible only when **Control Mode = Current Cntrl**	

Max Current
Driver_X_Max_Current
 	
Parameter's CAN Index by *Driver_X_Max_Current*
`5 = 0x3114 0x00`
`6 = 0x3115 0x00`
`7 = 0x3116 0x00`
`8 = 0x3117 0x00`
`9 = 0x3118 0x00`
`10 = 0x3119 0x00`
	0 – 3000 mA
	0 – 3000
		0 mA
		0
			Maximum current that the 100% command will effectuate.
			X= 5 – 10 for the VCL name *Driver_X_Max_Current*
			Reference the *Driver_X_Command*.

Min Current
Driver_X_Min_Current
 	
Parameter's CAN Index by *Driver_X_Min_Current*
`5 = 0x3124 0x00`
`6 = 0x3125 0x00`
`7 = 0x3126 0x00`
`8 = 0x3127 0x00`
`9 = 0x3128 0x00`
`10 = 0x3129 0x00`
	0 – 3000 mA
	0 – 3000
		0 mA
		0
			Minimum current that a 0.1% command will effectuate. (a value of .05 will turn off the driver).
			X= 5 – 10 for the VCL name *Driver_X_Min_Current*
			Reference the *Driver_X_Command.*

Dither Period
Driver_X_Dither_Period
 	
Parameter's CAN Index by *Driver_X_Dither_Period*
`5 = 0x3184 0x00`
`6 = 0x3185 0x00`
`7 = 0x3186 0x00`
`8 = 0x3187 0x00`
`9 = 0x3188 0x00`
`10 = 0x3189 0x00`
	2 – 1000 ms
	2 – 1000
		1000 ms
		1000
			Periodicity (frequency) of the applied dither.
			X= 5 – 10 for the VCL name *Driver_X_Dither_Period*
			Dither provides a small amount of cyclically changing current to vibrate the solenoid and thus keep it from "sticking" in one position. Without dither, it is harder to make small adjustment in the proportional valve position. Dither has both a frequency and amount.

Dither Amount
Driver_X_Dither_Amount
 	
Parameter's CAN Index by *Driver_X_Dither_Amount*
`5 = 0x3174 0x00`
`6 = 0x3175 0x00`
`7 = 0x3176 0x00`
`8 = 0x3177 0x00`
`9 = 0x3178 0x00`
`10 = 0x3179 0x00`
	0 – 100 %
	0 – 32767
		0 %
		0
			The percent of the Driver_X_Max_Current parameter that which is applied above and below its set point.
			X= 5 – 10 for the VCL name *Driver_X_Dither_Amount*
			Note, the driver measures current just before the PWM is shut off, therefore there is a lower limit where the current cannot be read. For the low side drivers, this is 8% (for HB drivers in low-side mode, this is 12%). The system designer should make sure that the load resistance, the coil return voltage supply, and minimum current will remain valid keeping the driver PWM above these values for good regulation.
			0% disables the dither function.

OUTPUTS — PWM DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Kp
Driver_X_Kp
 	
Parameter's CAN Index by *Driver_X_Kp*
`5 = 0x3194 0x00`
`6 = 0x3195 0x00`
`7 = 0x3196 0x00`
`8 = 0x3197 0x00`
`9 = 0x3198 0x00`
`10 = 0x3199 0x00`
	1 – 100 %
	20 – 2048
		8 %
		163
			The proportional section of the PI (Proportional/Integral) current regulator within the 1351 system controller.
			X= 5 – 10 for the VCL name *Driver_X_Kp*
			This parameter determines how aggressively the PI controller attempts to match the driver current command. Larger values provide tighter control.
			If the gain is set too high, oscillations can occur as the controller tries to control the Driver current. If it is set too low, the Driver current may behave sluggishly and be difficult to control.

Ki
Driver_X_Ki
 	
Parameter's CAN Index by *Driver_X_Ki*
`5 = 0x31A4 0x00`
`6 = 0x31A5 0x00`
`7 = 0x31A6 0x00`
`8 = 0x31A7 0x00`
`9 = 0x31A8 0x00`
`10 = 0x31A9 0x00`
	1 – 100 %
	20 – 2048
		8 %
		163
			Integral section of the PI (Proportional/Integral) current regulator (controller).
			X= 5 – 10 for the VCL name *Driver_X_Ki*
			The integral term (Ki) forces zero steady state error so the Driver will run at exactly the commanded current in conjunction with the Kp parameter (above). Larger values provide tighter control.
			If the gain is set too high, oscillations can occur as the controller tries to control the Driver current. If it is set too low, the Driver may take a long time to approach the exact commanded current.

Driver 10		This menu is Context Enabled: Visible only when Control Mode = Frequency Output	
Frequency Output			
Output Frequency *Frequency_output* `0x549D 0x00`	0 – 4000 Hz *0 – 4000*	Read Only	The output PWM frequency, in Hertz, of Driver 10 This variable is also viewable in the Monitor menu *Monitor\Outputs\PWM Drivers\Driver 10\Output Frequency*
Duty Cycle *Frequency_Out_Duty_Cycle0* `x3273 0x00`	0.0 – 100.0 % *0 – 1000*	50 %	Sets the PWM duty cycle of Driver 10. Driver 10's PWM duty cycle between 0 and 100%. The default is 50%. The definition of the duty cycle percentage (D) is the time the pulse width (PW) is active over the total period (T). D = PW/T~ 100%
Test			

Command
Driver_X_Command
 	
Variable's CAN Index by *Driver_X_Command*
`5 = 0x3364 0x00`
`6 = 0x3365 0x00`
`7 = 0x3366 0x00`
`8 = 0x3367 0x00`
`9 = 0x3368 0x00`
`10 = 0x3369 0x00`
	0.0 – 100.0 %
	0 – 1000
		0 %	The Driver command. It corresponds to the driver's control mode type setting. This is the steady-state input command, ramping, and initial levels are applied after and upon this command.
			X= 5 – 10 for the VCL name *Driver_X_Command*
			This is labeled as "Test" here because this is not a settable parameter that persists over a KSI cycle.
			When set in Programmer, this is a "Test" function for operating a driver during system setup/development. Its value is not retained over a key or power cycle. It re-sets to 0% (off).
			Note that this "*Driver_X_Command*" is the command to use in VCL for operating a driver. It must be used in the VCL program to operate the driver(s) as the normal basis. The 1351 also requires the Safety Out Command (state) be initialized (= 1) else there will be no voltage at the Coil Return/Safety Output (pins 11 and 12). This is part of the minimum VCL needed to operate the 1351 controller.

Half-Bridge Drivers

The Half Bridge drivers can be set to operate in two different topologies: Low or High side.

- In Low side topology, the load must be connected to the driver pin and B+, Keyswitch or Coil Return.
- In the High-side topology, the load must be connected to the driver pin and B– .

The topology mode of the Half-Bridge Drivers is set using its **Type** parameter. The Half-Bridge (HB) Drivers use the same control modes as the PWM Drivers. The HB drivers can be used to drive high-side connected (Coil Return) or low-side connected (B–) loads in constant current, constant voltage or PWM % modes.

- Driver 11 = HB Driver 1 (pin 23)
- Driver 12 = HB Driver 2 (pin 35)

OUTPUTS — HALF-BRIDGE DRIVERS

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Driver 11			

PWM
Driver_X_PWM
 	
Variable's CAN Index by *Driver_X_PWM*
`11 = 0x337A 0x00`
`12 = 0x337B 0x00`
	0.0 – 100.0 %
	0 – 1000
		Read Only	The driver's Pulse Width Modulation (PWM) percentage.
			X= 11 or 12 for the VCL name *Driver_X_PWM*
			100% equates to the driver being fully ON, meaning 100 percent of the Coil Return voltage (B+) is applied to the load (e.g., a contactor coil).
			50% equates to an applied voltage of 50%
			0 % turns off the driver.
			This variable is also located in Monitor:
			Monitor\Outputs\Half-Bridge Drivers\Driver X\PWM

Current
Driver_X_Current
 	
Variable's CAN Index by *Driver_X_Current*
`11 = 0x333A 0x00`
`12 = 0x333B 0x00`
	0.000 – 4.000 Amps
	0 – 4000
		Read Only	The Amperage through HB Drivers 1 or 2.
			X= 11 or 12 for the VCL name *Driver_X_Current*
			This variable is also located in Monitor:
			Monitor\Outputs\Half-Bridge Drivers\Driver X\Current

Control Mode
Driver_X_Mode
 	
Parameter's CAN Index by *Driver_X_Mode*
`11 = 0x30EA 0x00`
`12 = 0x30EB 0x00`
	Off
	Direct PWM
	Voltage Comp
	Current Cntrl
	
	0
	1
	2
	3
		Off
		0
			Sets the mode for HB Driver X
			X= 11 or 12 for the VCL name *Driver_X_Mode*
			0 = Off (Open mode)
			This mode disables the driver.
			1 = Direct PWM (normal PWM percentage operation)
			The driver's FET and wiring diagnostics are performed in this mode.
			2 = Voltage Comp (Voltage Compensated)
			Regulates the driver PWM based on the command, the nominal voltage setting, and the present battery voltage in an attempt to provide a constant average voltage at the output. As the battery voltage drops, the PWM percentage will increase to compensate for the lower voltage. The driver's FET and wiring diagnostics are performed in this mode. The initial and continuous PWM percent timing control is also applied in this driver control mode.
			3 = Current Cntrl (Current Control)
			Regulates the current through the driver. Typically used to control Proportional Valves (PV). This mode interprets the command as a load current request to regulate the PWM (via a PI controller) and thus the requested load current.

OUTPUTS — HALF-BRIDGE DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Type
Half_Bridge_X_Mode
 	
Parameter's CAN Index by *Half_Bridge_X_Mode*
`11 = 0x30F0 0x00`
`12 = 0x30F1 0x00`
	Low-Side
	High-Side
	0
	1
		Low-Side
		0
			0 = Low-side driver (connects the load to B– /ground)
			1 = High-side driver (connects the load to B+/battery)
			X= 11 or 12 for the VCL name *Half_Bridge_X_Mode*
			In the Low-side mode, the load must be connected between the driver and B+. The driver "sinks" current.
			In the High-side mode, the load must be connected between the driver and B– . The driver "sources" current.

Ramp Up
Driver_X_Ramp_Up_Time
 	
Parameter's CAN Index by *Driver_X_Ramp_Up_Time*
`11 = 0x313A 0x00`
`12 = 0x313B 0x00`
	0 – 10000 ms
	0 – 10000
		8 ms
		8
			The time in milliseconds to go from the minimum to the maximum control mode type, as a linear ramp-up.
			X= 11 or 12 for the VCL name *Driver_X_Ramp_Up_Time*
			Direct PWM: This is the time to change the pulse width modulation percentage
			Voltage Comp: This is the time to change the average voltage.
			Current Cntrl: This is time to change the current between the minimum and maximum driver current setting.
			A value of 0 milliseconds disables any ramping, resulting in a step-function-change when the driver command increases.
			Note that this *Ramp Up* is not applied in the Initial Level parameter.

Ramp Down
Driver_X_Ramp_Down_Time
 	
Parameter's CAN Index by *Driver_X_Ramp_Down_Time*
`11 = 0x314A 0x00`
`12 = 0x314B 0x00`
	0 – 10000 ms
	0 – 10000
		8 ms
		8
			The time in milliseconds to go from the minimum to the maximum control mode type, as a linear ramp-down.
			X= 11 or 12 for the VCL name *Driver_X_Ramp_Down_Time*
			Direct PWM: This is the pulse width percentage.
			Voltage Comp: This is nominal voltage percentage. (Note, Coil Return = B+ voltage)
			Current Cntrl: This is percentage between the maximum and minimum driver current setting (See the Current Control parameters menu, below)
			A value of 0 millisecond disables any ramping resulting in a step-function-change when the driver command decreases or is stopped (Driver_11_Command = 0).

OUTPUTS — HALF-BRIDGE DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Initial Level
Driver_X_Hold_Level
 	
Parameter's CAN Index by Driver_X_Hold_Level
`11 = 0x315A 0x00`
`12 = 0x315B 0x00`
	0.0 – 100.0 %
	0 – 1000
		0.0 %
		0
			The percentage command for the Initial Hold Level.
			This is the percentage the driver will initially "jump to" without ramping. This is affectedly the traditional contactor pull-in parameter. For example, a high pull-in ensures contactor contact-tips closure or a fast EM Brake release.
			X= 11 or 12 for the VCL name Driver_X_Ramp_Down_Time
			
			Direct PWM: This is the pulse width percentage.
			Voltage Comp: This is nominal voltage percentage. (Note, uses the nominal B+ voltage as the reference)
			Current Cntrl: This is percentage between the maximum and minimum driver current setting. (See the Current Control parameters menu, below)
			
			Note that the Initial level may be above or below the applied command.

Initial Time
Driver_X_Hold_Time
 	
Parameter's CAN Index by *Driver_X_Hold_Time*
`11 = 0x316A 0x00`
`12 = 0x316B 0x00`
	0 – 10000 ms
	0 – 10000
		0 ms
		0
			The time the Initial Level is applied.
			X= 11 or 12 for the VCL name *Driver_X_Hold_Time*
			This parameter allows customization of the driver's initial-level parameter duration, before any ramping to the longer-term command percentage.

Current Control		This menu is Context Enabled: Visible only when **Control Mode = Current Cntrl**	

Max Current
Driver_X_Max_Current
 	
Parameter's CAN Index by *Driver_X_Max_Current*
`11 = 0x311A 0x00`
`12 = 0x311B 0x00`
	0 – 3000 mA
	0 – 3000
		0 mA
		0
			Maximum current that the 100% command will effectuate.
			X= 11 or 12 for the VCL name *Driver_X_Max_Current*
			Reference the *Driver_X_Command*.

Min Current
Driver_X_Min_Current
 	
Parameter's CAN Index by *Driver_X_Min_Current*
`11 = 0x312A 0x00`
`12 = 0x312B 0x00`
	0 – 3000 mA
	0 – 3000
		0 mA
		0
			Minimum current that a 0.1% command will effectuate. (A value of 0 will turn off the driver).
			X= 11 or 12 for the VCL name *Driver_X_Min_Current*
			Reference the *Driver_11_Command.*

Dither Period
Driver_X_Dither_Period
 	
Parameter's CAN Index by *Driver_X_Dither_Period*
`11 = 0x318A 0x00`
`12 = 0x318B 0x00`
	2 – 1000 ms
	2 – 1000
		1000 ms
		1000
			Periodicity (frequency) of the applied dither.
			X= 11 or 12 for the VCL name *Driver_X_Dither_Period*
			Dither provides a small amount of cyclically changing current to vibrate the solenoid and thus keep it from "sticking" in one position. Without dither, it is harder to make small adjustment in the proportional valve position. Dither has both a frequency and amount.

OUTPUTS — HALF-BRIDGE DRIVERS, cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Dither Amount
Driver_X_Dither_Amount
 	
Parameter's CAN Index by *Driver_X_Dither_Amount*
`11 = 0x317A 0x00`
`12 = 0x317B 0x00`
	0 – 100 %
	0 – 32767
		0 %
		0
			The percent of the HB Driver's *Driver_X_Max_Current* parameter that is applied above and below its set point.
			X= 11 or 12 for the VCL name *Driver_X_Dither_Amount*
			Note that the driver measures current just before the PWM is shut off, therefore there is a lower limit where the current cannot be read. For the low side drivers, this is 8% (for HB drivers in low-side mode, this is 12%). The system designer should make sure that the load resistance, the coil return voltage supply, and minimum current will remain valid keeping the driver PWM above these values for good regulation.
			0% disables the dither function.

Kp
Driver_X_Kp
 	
Parameter's CAN Index by *Driver_X_Kp*
`11 = 0x319A 0x00`
`12 = 0x319B 0x00`
	1 – 100 %
	20 – 2048
		8 %
		163
			The proportional section of the PI (Proportional/Integral) current regulator within the 1351 system controller.
			X= 11 or 12 for the VCL name *Driver_X_Kp*
			Therefore, this parameter determines how aggressively the PI controller attempts to match the driver current command. Larger values provide tighter control.
			If the gain is set too high, oscillations can occur as the controller tries to control the Driver current. If it is set too low, the Driver current may behave sluggishly and be difficult to control.

Ki
Driver_11_Ki
 	
Parameter's CAN Index by *Driver_X_Ki*
`11 = 0x31AA 0x00`
`12 = 0x31AB 0x00`
	1 – 100 %
	20 – 2048
		8 %
		163
			Integral section of the PI (Proportional/Integral) current regulator (controller).
			X= 11 or 12 for the VCL name *Driver_X_Ki*
			The integral term (Ki) forces zero steady state error so the Driver will run at exactly the commanded current in conjunction with the Kp parameter (above). Larger values provide tighter control.
			If the gain is set too high, oscillations can occur as the controller tries to control the Driver current. If it is set too low, the Driver may take a long time to approach the exact commanded current.

Test			

Command
Driver_X_Command
 	
Parameter's CAN Index by *Driver_X_Command*
`11 = 0x336A 0x00`
`12 = 0x336B 0x00`
	0.0 – 100.0 %
	0 – 1000
		0	The HB Driver command. It corresponds to the driver's control mode type setting. This is the steady-state command, similar to the traditional contactor "holding voltage" parameter, following the Driver's Initial Level and Time parameters settings.
			X= 11 or 12 for the VCL name *Driver_X_Command*
			This is also the "ramp-to" and "ramp-from" reference.
			This is labeled as "Test" here because this is not a settable parameter that persists over a KSI cycle.
			When set in Programmer, this is a "Test" function for operating a driver during system setup/development. Its value is not retained over a key or power cycle. It re-sets to 0% (off).
			Note that this "*Driver_X_Command*" is the command to use in VCL for operating a driver. It must be used in the VCL program to operate the driver(s) as the normal basis.

Digital Drivers

The three digital drivers are low-side drivers, each with a 3-ampere current (sink) limit. These outputs can only be commanded by the VCL constants ON or OFF. (or 1 and 0). The control modes for PWM, constant current or voltage are not available. For inductive loads, these digital drivers have an internal fly-back diode to Coil Return. Resistive and capacitive (RC) loads must not exceed the 3-amp current-sink rating. These three drivers use the same VCL function as used for the Safety Output (refer to that section, below).

These drivers can be commanded by the constants ON or OFF. (or 1 and 0) using the *Digital Out_X_Commmand* or the VCL function `Put_Digital_Out()`. For example,

```
Digital_Out_2_Command = ON       ;fully turns-on the driver Digital Out 2 (pin 25).
```

The switch inputs related to these digital drivers, with their pull-up/downs are completely separate and can perform independently from the digital output. The switch operation is not related to the Enable or Off state of the driver. In fact, the driver and the switch can be used at the same time.... BUT do not switch to B+ if the digital driver is ON.

OUTPUTS — DIGITAL OUTPUTS

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION

Digital Out X State
Digital_Output_X_State
 	
Variable's CAN Index for *Digital_Output_X_State*
`1 = 0x3403 0x00`
`2 = 0x3404 0x00`
`3 = 0x3405 0x00`
	OFF – ON
	0 – 1
		Read Only	Present state of the digital output. When it is the driver and is commanded on, the pin will be pulled to B-/ground and the state will read ON (1).
			X= 1, 2, or 3 for this monitor variable (Digital Out X State) and the corresponding VCL name: *Digital_Output_X_State*
			This monitor variable is also available in the Monitor menu:
			Monitor\Outputs\Digital Drivers\Digital X State

Digital Out 1 Mode *Digital_Out_1_Mode* `0x30EC 0x00`	Off – Enabled *0 – 1*	0 *0*	Enables the Driver to be used as an output, otherwise this is the Analog 1 input or Encoder 2B input (See the example wiring diagram). When enabled, the digital driver will turn on and off according to the `Digital Output X Command` variable or `Put_Driver()` VCL functions. 0 = Digital Out disabled 1 = Digital Out can be controlled
Digital Out 2 Mode *Digital_Out_2_Mode* `0x30ED 0x00`	Off – Enabled *0 – 1*	0 *0*	Enables the Driver to be use as an output, otherwise this is the Switch 11 input (See the example wiring diagram). 0 = Digital Out disabled 1 = Digital Out can be controlled
Digital Out 3 Mode *Digital_Out_3_Mode* `0x30EE 0x00`	Off – Enabled *0 – 1*	0 *0*	Enables the Driver to be use as an output, otherwise this is the Switch 12 input (See the example wiring diagram). 0 = Digital Out disabled 1 = Digital Out can be controlled
Test		Same as the above drivers, this is labeled "Test" (here) because this is not a settable parameter that persists over a KSI cycle. Use in VCL to retain.	
Digital Out 1 Command *Digital_Out_1_Command* `0x3409 0x00`	0 – 1 *0 – 1*	0 *0*	Sets the driver action. 0 = Off 1 = On (driver must be enabled to actually be affected by the command)
Digital Out 2 Command *Digital_Out_2_Command* `0x340A 0x00`	0 – 1 *0 – 1*	0 *0*	Sets the driver action. 0 = Off (see the corresponding Mode parameter, above) 1 = On (See the corresponding Mode parameter, above)
Digital Out 3 Command *Digital_Out_3_Command* `0x340B 0x00`	0 – 1 *0 – 1*	0 *0*	Sets the driver action. 0 = Off (see the corresponding Mode parameter, above) 1 = On (See the corresponding Mode parameter, above)

Safety Output

The Safety Output has three distinct roles:

1. It provides controllable load power for the PWM and Digital drivers
2. Loads connected to this are protected against reversed battery connections
3. It can act as a high side driver

The Safety Output is off by default. In order to use this output as the coil return/battery supply side connection for the loads (as shown in the example wiring diagram), it must be first turned ON by VCL or over the CAN bus. If this is not turned on, the loads and drivers connected to these pins cannot operate properly. Setting the Safety Output Command parameter to 1 (ON) in Programmer is only valid while the keyswitch is ON, useful for testing during setup/commissioning the controller. It re-sets to 0 (OFF) upon a key/power cycle.

The Safety Output is controlled by its parameter Safety_Out_Command and/or by the VCL function `Put_Driver(SAFETY_OUT,x)` (where x = 0 for Off or 1 for On).

VCL or CAN bus SDO command should set this output ON after all safety checks are made and the system is ready, but it must be on before any connected load is required to operate. VCL or CAN SDO command can turn the Safety Output off to protect the loads in case of a system fault. The VCL code examples are:

```
Safety_Out_Command = ON          ;Turns ON (or OFF) the Safety
                                 ;Output. Shuts down all drivers
                                 ;if OFF.

Put_Driver(SAFETY_OUTPUT, 1)     ;Turns ON the Safety Output.
                                 ;Enables all drivers as per their
                                 ;individual settings and/or VCL
                                 ;commands.
```

OUTPUTS — SAFETY OUTPUT

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Safety Output State *Safety_Output_State* `0x3402 0x00`	0 – 1 *0 – 1*	Read Only	The present state of the output. This variable is also available in the Monitor menu, *Monitor\Outputs\Safety Output\Safety Output State* 0 = ON 1 = OFF
Safety Out Command *Safety_Out_Command* `0x3408 0x00`	0 – 1 *0 – 1*	0	Turns ON or OFF the safety Output. Changing this parameter in CIT is only viable while KSI is On. Cycling KSI to Off will always revert the Safety Out Command to 0 (zero), which is Off. **0 = Off.** All the drivers supplied by the Coil Return (pins 11 and 12) are shut off. No B+ supply voltage is applied to the Coil Return (pins 11 and 12). The Safety Output command can shut down all drivers at once. **1 = On.** B+ voltage is applied to the Coil Return supply (pins 11 and 12). All the drivers supplied by the Coil Return can be individually setup/used. Maximum of 23 Amps. Reference the example wiring diagram (Figure 4).

Analog Output

This adjustable 0-10V output is intended to drive high-impedance loads, such as a battery discharge indicator or hour meter. The Analog Output can swing from 0-10 volts. The following settings and monitor value are available for the Analog Output pin.

OUTPUTS — ANALOG OUTPUT

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Output Voltage *Analog_11_Volts* `0x330A 0x00`	0.0 – 20.0 V *0 – 2000*	Read Only	The present output voltage (state). This variable is also available in the Monitor menu *Monitor\Outputs\Analog Output\Output Voltage*
Analog Out Command *Analog_Out_Command* `0x3401 0x00`	0 – 100 % *0 – 1000*	0.0 *0*	This adjustable 0-10V output is intended to drive high-impedance loads, such as a battery discharge indicator or hour meter. The maximum output (supply) current is approximately 10mA. 0% = Off 100% = 10V

CAN

Each CAN port is separate from the other offering different Node IDs and an independent baud rate. Yet, they can be combined (connected together) via a parameter setting. When combined, each port's communications are received and sent out on the other port. The circuits are not isolated, sharing the 1351 internal power supply and I/O ground (B–) reference.

- CAN 1 (pins 3 and 4): CANopen(11-bit)/J1939(29-bit)
- CAN 2 (pins 5 and 6): CANopen(11-bit)/J1939(29-bit)

Appendix A includes detailed instructions (with examples) on setting up the PDO mapping in CIT (Programmer) and by SDO write messages.

CAN

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Port 1			
NMT State *port_1_nmt_state* `0x5490 0x00`	0 – 127 *0 – 127*	Read Only	Controller CAN1 NMT state: 0 = Initialization 4 = Stopped 5 = Operational 127 = pre-operational VCL related: Setup_NMT_State(PORT, state)
Node ID *Port_1_Node_ID* `0x2000 0x01`	0x01 – 0x7F (hex) *1 – 127 (decimal)*	0x11 17	Displays the controller's Node ID, in hexadecimal. For example: 0x11 = 17d Note: Setup/assignment is via the Curtis Integrated Toolkit™: Launchpad, add a device. Do not assign the same CAN Node ID to the 2 ports.
Baud Rate *Port_1_Baud_Rate* `0x2001 0x01`	100K – 1M (kbps) *–1 – 4*	125k 0	Sets the CAN baud rate for the CANopen Ancillary system: –1 = 100Kbps, 0 = 125Kbps, 1 = 250Kbps, 2 = 500Kbps, 3 = 800Kbps, 4 = 1000Kbps. Note that the 2 CAN ports can have different Baud Rates.
Heartbeat Rate *Port_1_Heartbeat_Rate* `0x1017 0x00`	0 – 4000 ms *0 – 4000*	100	Sets the rate at which the CAN heartbeat messages are sent from the CANopen (port 1) system.
Termination *CAN_1_Termination* `0x3002 0x00`	Off – Enabled *0 – 1*	Off	Enables the electronically controlled termination resistor (120 Ω) for CAN Port 1. This termination is internal to the 1351, no additional or external hardware is required.
RPDO 1			
Timeout (RPDO 1) *can_rpdo_1_event_timer* `0x1400 0x05`	*0 – 65535* *0 – 65535*		Receive (Rx) device, time before the PDO Timeout fault. (PDO Timeout period) 1-unit = 4 ms (e.g., 5 = 4ms x 5 = 20 ms timeout)
COB ID (RPDO 1) *can_rpdo_1_cob_id* `0x1400 0x01`	1h – FFFFFFFFh 0 – 4294967295	80000211h	Application's 11-bit COB-ID (Communication Object ID) Note: Use VCL for 29-bit COB ID Note: Hexadecimal 80000211h = 0x80000211
Length (RPDO 1) *can_rpdo_1_length* `0x1600 0x00`	0 – 8 *0 – 8*	0	Number of CAN objects in map (not the number of bits or bytes).
1 *can_rpdo_1_map_1* `0x1600 0x01`	0h – FFFFFFFFh 0 – 4294967295	0h 0	Map 1st object variable and bit length (8,16, 24, or 32)

CAN — cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
2 *can_rpdo_1_map_2* 0x1600 0x02	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
3 *can_rpdo_1_map_3* 0x1600 0x03	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
4 *can_rpdo_1_map_4* 0x1600 0x04	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
5 *can_rpdo_1_map_5* 0x1600 0x05	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
6 *can_rpdo_1_map_6* 0x1600 0x06	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
7 can_rpdo_1_map_7 0x1600 0x07	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
8 *can_rpdo_1_map_8* 0x1600 0x08	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
TPDO 1			
Timeout (TPDO 1) *can_tpdo_1_event_timer* 0x1800 0x05	*0 – 65535* *0 – 65535*	40	Transmit (Tx) device, time before the PDO Timeout fault. (PDO Timeout period) 1-unit = 4 ms (e.g., 5 = 4ms x 5 = 20 ms timeout)
COB ID (TPDO 1) *can_tpdo_1_cob_id* 0x1800 0x01		C0000191h	Application's 11-bit COB-ID (Communication Object ID) Note: Use VCL for 29-bit COB ID Note: Hexadecimal C0000191h = 0xC0000191
Length (RPDO 1) *can_tpdo_1_length* 0x1A00 0x00	0 – 8 *0 – 8*	0	Number of CAN objects in map (not the number of bits or bytes).
1 *can_tpdo_1_map_1* 0x1A00 0x01	0h – FFFFFFFFh 0 – 4294967295	0h 0	Map 1st object variable and bit length (8,16, 24, or 32)
2 *can_tpdo_1_map_2* 0x1A00 0x02	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
3 *can_tpdo_1_map_3* 0x1A00 0x03	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
4 *can_tpdo_1_map_4* 0x1A00 0x04	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
5 *can_tpdo_1_map_5* 0x1600 0x05	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
6 *can_tpdo_1_map_6* 0x1A00 0x06	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
7 *can_tpdo_1_map_7* 0x1A00 0x07	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)

CAN — cont'd

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
8 *can_tpdo_1_map_8* `0x1A00 0x08`	0h – FFFFFFFFh 0 – 4294967295	50008h 0	Map 1st object variable and bit length (8,16, 24, or 32)
RPDO {2, 3, 4} Byte Map			See RPDO 1 (above), adjusted for CAN Index/Etc. See CIT Programmer
TPDO {2, 3, 4} Byte Map			See TPDO 1 (above), adjusted for CAN Index/Etc. See CIT Programmer
Port 2			
NMT State *port_2_nmt_state* `0x5491 0x00`	0 – 127 (0 – 127) *0 – 127*	Read Only	Controller CAN1 NMT state: 0 = Initialization 4 = Stopped 5 = Operational 127 = pre-operational VCL related: Setup_NMT_State(PORT, state)
Node ID *Port_2_Node_ID* `0x2000 0x02`	0x01 – 0x7F (hex) *1 – 127 (decimal)*	0x12 18	Displays the controller's Node ID, in hexadecimal. For example: 0x12 = 18d Note: Setup/assignment is via the Curtis Integrated Toolkit™: Launchpad, add a device. Do not assign the same CAN Node ID to the 2 ports.
Baud Rate *Port_2_Baud_Rate* `0x2001 0x02`	100K – 1M (kbps) *–1 – 4*	125k 0	Sets the CAN baud rate for the CANopen Ancillary system: –1 = 100Kbps, 0 = 125Kbps, 1 = 250Kbps, 2 = 500Kbps, 3 = 800Kbps, 4 = 1000Kbps. Note that the 2 CAN ports can have different Baud Rates.
Heartbeat Rate *Port_2_Heartbeat_Rate* `0x3240 0x00`	0 – 4000 ms *0 – 4000*	100	Sets the rate at which the CAN heartbeat messages are sent from the J1939 (port 2) system.
Termination *CAN_2_Termination* `0x3003 0x00`	Off – Enabled *0 – 1*	Off	Enables the electronically controlled termination resistor (120 Ω) for CAN Port 2. This termination is internal to the 1351, no additional or external hardware is required.
Cross Connection *CAN_Port_Cross_Connection* `0x3004 0x00`	Off – Enabled *0 – 1*	Off	Enable the 2 CAN Ports to be connected together. Any message received on either Port 1 or Port 2 will be received by both CAN1 and CAN2. Any message send by CAN 1 or CAN 2 will be sent out both ports. This is often used when CANopen SRDOs are required in a system. Note To use this feature, the baud rates of both CAN ports must be set the same in in order to prevent buss errors and port shutdowns. The Node IDs of each port must be different.

ACCELEROMETER

PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Accelerometer			
Accelerometer X Filter *Accelerometer_X_Filter_TC* `0x3270 0x00`	2 – 3000 ms *2 – 3000*	128 ms	The time-constant of a first order low-pass filter. Increasing this parameter's time (ms) will result in larger filtering of the monitor valuable Accel X Axis's value Reference: *Monitor » Accelerometer » Accel X Axis (Accel_X_Axis 0x3430 0x00)*
Accelerometer Y Filter *Accelerometer_Y_Filter_TC* `0x3271 0x00`	2 – 3000 ms *2 – 3000*	128 ms	The time-constant of a first order low-pass filter. Increasing this parameter's time (ms) will result in larger filtering of the monitor valuable Accel Y Axis's value Reference: *Monitor » Accelerometer » Accel Y Axis (Accel_Y_Axis 0x3431 0x00)*
Accelerometer Z Filter *Accelerometer_Z_Filter_TC* `0x3272 0x00`	2 – 3000 ms *2 – 3000*	128 ms	The time-constant of a first order low-pass filter. Increasing this parameter's time (ms) will result in larger filtering of the monitor valuable Accel Z Axis's value Reference: *Monitor » Accelerometer » Accel Z Axis (Accel_Z_Axis 0x3432 0x00)*

SUPERVISOR UPDATE

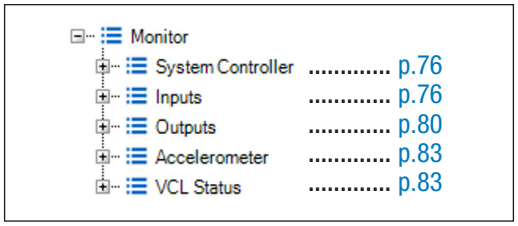
PARAMETER	ALLOWABLE RANGE	DEFAULT	DESCRIPTION
Supervisor Update			
Notice: These parameters are for updating the processors, only. Do not use/change these parameters in setting-up the 1351. Contact the local Curtis support engineering office before using the parameters. Incorrect usage may lead the 1351 in an un-responsive state requiring specific SDOs to recover.			
Primary State *primary_state* `0x5080 0x00`	0 – 127 *0 – 127*	Read Only	For updating the Primary processor, ONLY **Follow the Update Instruction for relevance.***
Supervisor State *supervisor_state* `0x5081 0x00`	0 – 127 *0 – 127*	Read Only	For updating the Supervisor processor, ONLY **Follow the Update Instruction for relevance.***
Force Primary Enter CAN BSL state *force_primary_enter_bsl_command* `0x5500 0x00`	0 – 1 *0 – 1*	0	For updating the Primary processor START MANAGER, ONLY Forces the primary to enter CAN BSL mode to update PSM. Note: Be sure to close CIT in 10S after evoking the command. **Follow the Update Instruction for relevance.***
Supervisor Update Control *Supervisor_Flash_Control* `0x3263 0x00`	Update None Update supervisor SM Update supervisor OS and App	0 1 2	For updating the Supervisor processor, ONLY **Follow the Update Instruction for relevance.***

** When a firmware update is released by Curtis, instructions will be available/included for using these variables.*
Do not use these variables when only updating the controller's CDEV version (e.g., for future cedev beyond this manual's cdev 2.11.14.0.).

4 — MONITOR VARIABLES

Monitor variables are read only items intended to assist with setting-up the 1351 system controller parameters. Use the monitor variables for 1351 faults and system diagnostic. As noted in Chapter 3, monitor variables shown in the Programmer App's parameter menus are the same as those in the Monitor menu. The menu structure generally follows the parameter menu layout in terms of System, Inputs and Outputs. The 3-axis accelerometer readout is located in the Monitor menu.

Table 20 Monitor Variables Menus



As displayed using the Curtis Integrated Toolkit™ or the 1313 HHP

SYSTEM CONTROLLER..... p. 76
- —Keyswitch Voltage
- —Ext 5V
- —Ext 5V Current
- —Ext 12V
- —Ext 12V Current
- —Module Temperature

INPUTS..... p. 76

SWITCH INPUT..... p. 76
- —Virtual Switch 1
- —Virtual Switch 2
- . . . . .
- —Virtual Switch 11

—Switch 1..... p. 77
- —Switch 2
- . . . . .
- —Switch 14

ANALOG INPUT..... p. 77
- —Analog 1
- —Analog 2
- . . . . .
- —Analog 11

POT INPUT..... p. 78
- —Wiper Position
- —Resistance

RTD INPUT..... p. 78
- —RTD1
  - —Resistance
  - —Value
- —RTD2
  - —Resistance
  - —Value
- —RTD3
  - —Resistance
  - —Value
- —RTD4
  - —Resistance
  - —Value

HIGH SPEED DIGITAL INPUT.... p. 79
- —High Speed Input 1
  - —Count
  - —Frequency
  - —Pulse Width
- —High Speed Input 2
  - —Count
  - —Frequency
  - —Pulse Width

ENCODER INPUT..... p. 79
- —Encoder 1
  - —Position
  - —RPM
- —Encoder 2
  - —Position
  - —RPM

OUTPUTS..... p. 80

PWM DRIVERS..... p. 80
- —Driver 1
  - — PWM
  - — Current
- —Driver 2
  - — PWM
  - — Current
- . . . . .
- —Driver 10
  - — PWM
  - — Current
  - — Output Frequency

HALF-BRIDGE DRIVER... p. 81
- —Driver 11
  - — PWM
  - — Current
- —Driver 12
  - — PWM
  - — Current

DIGITAL DRIVERS..... p. 81
- —Digital Out 1 State
- —Digital Out 2 State
- —Digital Out 3 State

SAFETY OUTPUT..... p. 81
- —Safety Output State

ANALOG OUTPUT..... p. 81
- —Output Voltage

ACCELEROMETER..... p. 83
- —Accel X Axis
- —Accel Y Axis
- —Accel Z Axis

VCL STATUS..... p. 83
- —Master Timer
- —VCL Error Module
- —VCL Error Code
- —VCL Error Function

MONITOR VARIABLES: SYSTEM CONTROLLER

VARIABLE	DISPLAY RANGE	DESCRIPTION
Keyswitch Voltage *Keyswitch_Voltage* `0x331C 0x00`	0.0 – 140.0 V *0 – 14000*	Voltage at the keyswitch (KSI, pin 7). This will be at/near the battery voltage during operation.
Ext 5V *Ext_5V* `0x3325 0x00`	0.0 – 10.0 V *0 – 1000*	The measured output voltage at the 5V supply (pin 10) This variable is also available in the parameter External Supplies menu, see: *Configuration\System Controller\External Supplies\Ext 5V*
Ext 5V Current *Ext_5V_Current* `0x3327 0x00`	0 – 1000 mA *0 – 1000*	The measured output current at the 5V supply (pin 10) Note, the combined load current between the +5V and +12V outputs cannot exceed 300 mA This variable is also available in the parameter External Supplies menu, see: *Configuration\System Controller\External Supplies\Ext 5V Current*
Ext 12V *Ext_12V* `0x3326 0x00`	0.0 – 20.0 V *0 – 2000*	The measured output voltage at the 12V supply (pin 9) This variable is also available in the parameter External Supplies menu, see: *Configuration\System Controller\External Supplies\Ext 12V*
Ext 12V Current *Ext_12V_Current* `0x3328 0x00`	0 – 1000 mA *0 – 1000*	The measured output current at the 12V supply (pin 9) Note, the combined load current between the +5V and +12V outputs cannot exceed 300 mA This variable is also available in the parameter External Supplies menu, see: *Configuration\System Controller\External Supplies\Ext 12V Current*
Module Temperature *Module_Temperature* `0x3329 0x00`	– 50.0 – 100.0°C *– 500 – 1000*	The internal temperature of the 1351 System Controller. If excessive or unexpected high temperature, reconsider mounting method and location.

MONITOR VARIABLES: INPUTS → Switch Input/Virtual Switches

VARIABLE	DISPLAY RANGE	DESCRIPTION

Virtual Switch X
Virtual_Switch_X
 	
Monitor Variable CAN Index
Virtual_Switch_X
`1 = 0x3380 0x00`
`2 = 0x3381 0x00`
`3 = 0x3382 0x00`
`4 = 0x3383 0x00`
`5 = 0x3384 0x00`
`6 = 0x3385 0x00`
`7 = 0x3386 0x00`
`8 = 0x3387 0x00`
`9 = 0x3388 0x00`
`10 = 0x3389 0x00`
`11 = 0x338A 0x00`
	0 – 1
	0 – 1
		0 = Off
		1 = On
		The **Virtual Switch X** state is based upon the *Analog 1 Input* voltage (pin 1), the high and low thresholds set for the analog input and the **VSW X Active Level** parameter settings.
		Note: Replace X with 1 – 11 for the specific VSW.
		See Programmer: *Configuration\Inputs\Virtual Switches\VSW X\Active Level*
		Configuration\Inputs\Analog Inputs\Analog X\Value

MONITOR VARIABLES: INPUTS → Switch Input/Switches

VARIABLE	DISPLAY RANGE	DESCRIPTION
Switch X *Switch_X* Monitor Variable CAN Index *Switch_X* 1 = 0x338B 0x00 2 = 0x338C 0x00 3 = 0x338D 0x00 4 = 0x338E 0x00 5 = 0x338F 0x00 6 = 0x3390 0x00 7 = 0x3391 0x00 8 = 0x3392 0x00 9 = 0x3393 0x00 10 = 0x3394 0x00 11 = 0x3395 0x00 12 = 0x3396 0x00 13 = 0x3397 0x00 14 = 0x3398 0x00	0 – 1 *0 – 1*	0 = Off 1 = On **Switch 1** is based upon the corresponding SW X parameter settings. Use VCL to implement the value of *Switch_1* in software. Note: Replace X with 1 – 14 for the specific SW. See Programmer: *Configuration\Inputs\Switches\SW X\ \\Status Monitor\Inputs\Switch Input\Switch X* Note: **Switch X** in the Monitor menu is the same as **Status** in the Configuration menu
Input X State *Input_X_State* Monitor Variable CAN Index *Input_X_State* 1 = 0x33BB 0x00 2 = 0x33BC 0x00 3 = 0x33BD 0x00 4 = 0x33BE 0x00 5 = 0x33BF 0x00 6 = 0x33C0 0x00 7 = 0x33C1 0x00 8 = 0x33C2 0x00 9 = 0x33C3 0x00 10 = 0x33C4 0x00 11 = 0x33C5 0x00 12 = 0x33C6 0x00 13 = 0x33C7 0x00 14 = 0x33C8 0x00	0 – 1 *0 – 1*	The Input State is the state of the voltage at the switch input pin. If the pin has B+ on it, it will read 1, if it has 0 volts on it, it will read 0. If the active state on a switch is set High, then the Input State and Switch Status will be the same. If the active state is set Low, then the input state and Switch Status will be opposite Note: Replace X with 1 – 14 for the specific switch state. See Programmer: *Configuration\Inputs\Switches\SW X\ \\Status Monitor\Inputs\Switch Input\Switch X*

MONITOR VARIABLES: INPUTS → Analog Input

VARIABLE	DISPLAY RANGE	DESCRIPTION
Analog X *Analog_X_Volts* Monitor Variable CAN Index *Analog_X_Volts* 1 = 0x3300 0x00 2 = 0x3301 0x00 3 = 0x3302 0x00 4 = 0x3303 0x00 5 = 0x3304 0x00 6 = 0x3305 0x00 7 = 0x3306 0x00 8 = 0x3307 0x00 9 = 0x3308 0x00 10 = 0x3309 0x00 11 = 0x330A 0x00	0.0 – 20.0 V *0 – 2000*	The voltage at Analog X input. Note: Replace X with 1 – 11 for the specific Analog Input. This is the same Read-Only variable located with the Programmer: *Configuration\Inputs\Analog Inputs\Analog X\Value*

MONITOR VARIABLES: INPUTS → Pot Input

VARIABLE	DISPLAY RANGE	DESCRIPTION
Wiper Position *Wiper_Position* `0x3318 0x00`	0 – 100 % *0 – 1000*	The calculated position percentage from the 2-wire or 3-wire potentiometers. This read-only variable is repeated in the Pot Input parameters. *Configuration\Inputs\Pot Input\Wiper Position*
Resistance *Pot_Resistance* `0x3319 0x00`	0 – 15000 Ohms *0 – 15000*	The calculation of the wiper resistance to ground. The 1351 System Controller's dynamically calculated resistance of a connected potentiometer. Pin 20 is the wiper (between pin 21, Pot Hi and pin 8, I/O Gnd). This read-only variable is repeated in the Pot Input parameters. *Configuration\Inputs\Pot Input\Resistance*

MONITOR VARIABLES: INPUTS → RTD Input

VARIABLE	DISPLAY RANGE	DESCRIPTION
RTD X		

Resistance
RTD_X_R
 	
Monitor variable CAN Index
RTD_X_R
`1 = 0x3310 0x00`
`2 = 0x3311 0x00`
`3 = 0x3312 0x00`
`4 = 0x3313 0x00`
	0 – 20000 Ohms
	0 – 20000
		The measured RTD's resistance at the input pin
		Note: Replace X with 1 – 4 for the specific RTD Input (see example wiring diagram).
		This read-only variable is repeated in the RTD Input parameters.
		Configuration\Inputs\RTD Input\RTD X\Resistance

Value
RTD_X_OUT
 	
Monitor variable CAN Index
RTD_X_Out
`1 = 0x3314 0x00`
`2 = 0x3315 0x00`
`3 = 0x3316 0x00`
`4 = 0x3317 0x00`
	–32768 – 32767
	–32768 – 32767
		The "value" at the RTD input corresponding to the RTD resistance point. The meaning of the output value depends on the type of sensor used and the mapping.
		Note: Replace X with 1 – 4 for the specific RTD Input (see example wiring diagram).
		This read-only variable is repeated in the Pot Input parameters.
		Configuration\Inputs\RTD Input\RTD X\Value

MONITOR VARIABLES: INPUTS → High Speed Digital Input

VARIABLE	DISPLAY RANGE	DESCRIPTION

Count
HS_Input_X_Count
 	
Monitor variable CAN Index
HS_Input_X_Count
`1 = 0x3422 0x00`
`2 = 0x3425 0x00`
	0 – 4294967295
	0 – 4294967295
		Count is based on the Type parameter selection. The counter range is 0 – (2^{32}) – 1.
		It will overflow and reset to 0.
		X = 1 = High Speed Input 1 = pin 15 (see example wiring diagram)
		X = 2 = High Speed Input 2 = pin 14 (see example wiring diagram)
		This read-only variable is repeated in the Input parameters:
		Configuration\Inputs\High Speed Digital Input\High Speed Input X\Count

Frequency
HS_Input_X_Frequency
 	
Monitor variable CAN Index
HS_Input_X_Frequency
`1 = 0x3420 0x00`
`2 = 0x3423 0x00`
	0 – 50000 Hz
	0 – 50000
		Measure of the frequency of the incoming signal pulses.
		The frequency measurement is derived from the input signal's rising to rising edge.
		X = 1 = High Speed Input 1 = pin 15 (see example wiring diagram)
		X = 2 = High Speed Input 2 = pin 14 (see example wiring diagram)
		This read-only variable is repeated in the Input parameters:
		Configuration\Inputs\High Speed Digital Input\High Speed Input X\Frequency

Pulse Width
HS_Input_X_Pulse_Wdith
 	
Monitor variable CAN Index
HS_Input_X_Pulse_Wdith
`1 = 0x3421 0x00`
`2 = 0x3424 0x00`
	0.0 – 100 %
	0 – 1000
		Based upon the High Speed Input Type parameter selection, either the high or low (voltage) duration of the incoming PWM is used to generate the Pulse Width percentage, where:
		0% = Off (no signal)
		100% = fully On (no PWM, yet rather a constant state)
		X = 1 = High Speed Input 1 = pin 15 (see example wiring diagram)
		X = 2 = High Speed Input 2 = pin 14 (see example wiring diagram)
		This read-only variable is repeated in the Input parameters:
		Configuration\Inputs\High Speed Digital Input\High Speed Input X\Pulse Width

MONITOR VARIABLES: INPUTS → ENCODER INPUT

VARIABLE	DISPLAY RANGE	DESCRIPTION
Encoder 1		
Position *Encoder_1_Position* `0x3414 0x00`	– 2147483648 – 2147483647 – 2147483648 – 2147483647 {i.e., – (2^{31}) – [(2^{31})– 1]}	Rotor (Encoder 1) position since last KSI = On. Rotor revolutions in the forward direction are positive (count up), while revolutions in the reverse direction are negative (count down).
RPM *Encoder_1_RPM* `0x3410 0x00`	– 2147483648 – 2147483647 rpm – 2147483648 – 2147483647 {i.e., – (2^{31}) – [(2^{31})– 1]}	Encoder 1's speed in revolutions per minute (rpm). Positive RPM = Forward rotation Negative RPM = Reverse rotation Reference the Direction parameter for whether forward or reverse: *Configuration\Inputs\Encoder Input\Encoder 1\Direction* Note that these monitor variables are common to the three-encoder devices: Quadrature, Sin/Cos, and Sawtooth.
Encoder 2		
Position *Encoder_2_Position* `0x3415 0x00`	– 2147483648 – 2147483647 – 2147483648 – 2147483647 {i.e., – (2^{31}) – [(2^{31})– 1]}	Rotor (Encoder 2) position since last KSI = On. Rotor revolutions in the forward direction are positive (count up), while revolutions in the reverse direction are negative (count down).
RPM *Encoder_2_RPM* `0x3412 0x00`	– 2147483648 – 2147483647 rpm – 2147483648 – 2147483647 {i.e., – (2^{31}) – [(2^{31})– 1]}	Encoder 2's speed in revolutions per minute (rpm). Positive RPM = Forward rotation Negative RPM = Reverse rotation Reference the Direction parameter for whether forward or reverse: *Configuration\Inputs\Encoder Input\Encoder 2\Direction* Note that these monitor variables are common to the three-encoder devices: Quadrature, Sin/Cos, and Sawtooth.

MONITOR VARIABLES: OUTPUTS → PWM Drivers

VARIABLE	DISPLAY RANGE	DESCRIPTION
Driver 1		

PWM
Driver_X_PWM
 	
Monitor variable CAN Index
Driver_X_PWM
`1 = 0x3370 0x00`
`2 = 0x3371 0x00`
`3 = 0x3372 0x00`
`4 = 0x3373 0x00`
`5 = 0x3374 0x00`
`6 = 0x3375 0x00`
`7 = 0x3376 0x00`
`8 = 0x3377 0x00`
`9 = 0x3378 0x00`
`10 = 0x3379 0x00`
	0.0 – 100.0 %
	0 – 1000
		The driver's PWM percentage.
		Note: Replace X with 1 – 10 for the specific *Driver_X_PWM* (See example wiring diagram).
		100% equates to the driver being fully ON, meaning 100 percent of the Coil Return voltage (B+) is applied to the load (e.g., contactor coil). See the example wiring diagram.
		50% equates to an applied voltage of 50%
		0 % turns off the driver.
		This variable is also located in the Configuration Outputs menu: *Configuration\Outputs\PWM Drivers\Driver X\PWM*

Current
Driver_X_Current
 	
Monitor variable CAN Index
Driver_X_Current
`1 = 0x3330 0x00`
`2 = 0x3331 0x00`
`3 = 0x3332 0x00`
`4 = 0x3333 0x00`
`5 = 0x3334 0x00`
`6 = 0x3335 0x00`
`7 = 0x3336 0x00`
`8 = 0x3337 0x00`
`9 = 0x3338 0x00`
`10 = 0x3339 0x00`
	0.000 – 4.000 Amps
	0 – 4000
		The average current through the Driver.
		Note: Replace X with 1 – 10 for the specific *Driver_X_Current* (See example wiring diagram).
		This variable is also located in the Configuration Outputs menu: *Configuration\Outputs\PWM Drivers\Driver X\Current*

Driver 10		
Output Frequency *Frequency_output* `0x549D 0x00`	0 – 4000 Hz *0 – 4000*	The output PWM frequency, in Hertz, of Driver 10 This variable is also located in the Configuration Outputs menu: *Configuration\Outputs\PWM Drivers\Driver 10\Output Frequency*

MONITOR VARIABLES: OUTPUTS → Half-Bridge Drivers

VARIABLE	DISPLAY RANGE	DESCRIPTION
Driver 11 and 12		

PWM
Driver_X_PWM
 	
Monitor variable CAN Index
Driver_X_PWM
`11 = 0x337A 0x00`
`12 = 0x337B 0x00`
	0.0 – 100.0 %
	0 – 1000
		The driver's PWM percentage.
		Note: Replace X with 11 or 12 for the specific *Driver_X_PWM*
		Driver 11 = the HB Driver 1 (pin 23)
		Driver 12 = the HB Driver 2 (pin 35)
		100% equates to the driver being fully ON, meaning 100 percent of the voltage (B+) is applied to the load (e.g., contactor coil or a motor).
		50% equates to an applied voltage of 50%
		0 % turns off the driver.
		This variable is also located in the Configuration Outputs menu:
		Configuration\Outputs\Half-Bridge Drivers\Driver X\PWM

Current
Driver_X_Current
 	
Monitor variable CAN Index
Driver_X_Current
`11 = 0x333A 0x00`
`12 = 0x333B 0x00`
	0.000 – 4.000 Amps
	0 – 4000
		The average current through the HB Driver.
		Note: Replace X with 11 or 12 for the specific *Driver_X_Current*
		Driver 11 = the HB Driver 1 (pin 23)
		Driver 12 = the HB Driver 2 (pin 35)
		This variable is also located in the Configuration Outputs menu:
		Configuration\Outputs\ Half-Bridge Drivers \Driver X\Current

MONITOR VARIABLES: INPUTS → Digital Drivers

VARIABLE	DISPLAY RANGE	DESCRIPTION

Digital Out 1 State
Digital_Output_X_State
 	
Monitor variable CAN Index
Digital_Output_X_State
`1 = 0x3403 0x00`
`2 = 0x3404 0x00`
`3 = 0x3405 0x00`
	OFF – ON
	0 – 1
		Present state of the digital output. When it is the driver and is commanded on, the pin will be pulled to B-/ground and the state will read ON (1).
		X= 1, 2, or 3 for this monitor variable (Digital Out X State) and the corresponding VCL name: *Digital_Output_X_State*
		This monitor variable is also available in the configuration outputs menu:
		Configuration\Outputs\Digital Drivers\Digital Out X State\Outputs

MONITOR VARIABLES: INPUTS → Safety Output

VARIABLE	DISPLAY RANGE	DESCRIPTION
Safety Output State *Safety_Output_State* `0x3402 0x00`	0 – 1 *0 – 1*	The present state of the output. 0 = ON 1 = OFF This variable is also available in the parameter configuration menu, *Configuration \Outputs\Safety Output\Safety Output State*

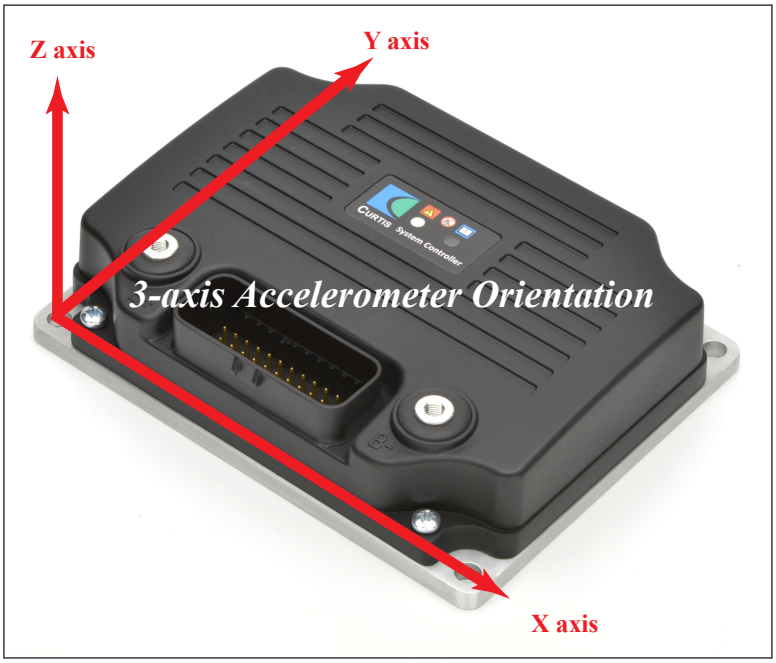
MONITOR VARIABLES: INPUTS → Analog Output

VARIABLE	DISPLAY RANGE	DESCRIPTION
Output Voltage *Analog_11_Volts* `0x330A 0x00`	0.0 – 20.0 V *0 – 2000*	The present output voltage (state). This variable is also available in the parameter configuration menu *Configuration \Outputs\Analog Output\Output Voltage*

ACCELEROMETER

3-axis Accelerometer (X Y Z axis)

The 1351 comes standard with a 3-axis accelerometer. The 3-axis accelerometer can be used in many situations, such as recording abuse (shock), to limit vehicle speed, lift or reach based upon the grade (tilt), monitor vehicle acceleration/time, or vehicle sway (roll). Implement the accelerometer variables using VCL.



X-axis The accelerometer's x-axis is from left-to-right across the 1351. If the 1351 is placed on its left side (B+ terminal down, B– terminal up), the accelerator's x-axis (monitor variable `X`) will read positive 1 g and approximately 0 in the other two.

Y-axis The accelerometer's y-axis is from front-to-back on the 1351. If the 1351 is placed on its rear side (35-pin connector down), the accelerator's y-axis (monitor variable `Y`) will read positive 1 g and approximately 0 in the other two.

Z-axis The accelerometer's z-axis is from bottom to top on the 1351. If the 1351 is placed flat on its cold plate (connections up), the accelerator's z-axis (monitor variable `Z`) will read positive 1 g and approximately 0 in the other two.

MONITOR VARIABLES: ACCELEROMETER

VARIABLE	DISPLAY RANGE	DESCRIPTION
Accel X Axis *Accel_X_Axis* `0x3430 0x00`	– 16.000 – 16.000 *– 16000 – 16000*	The 3-axis accelerometer X-axis output signal. The value is in g in one mg increments (e.g., the force of gravity = 1.000 g). Note, this is a rounded value due to the filter setting and the rounding (to two decimal places) of the sensor sensitivity value (where raw mg/32767 = sensitivity).
Accel Y Axis *Accel_Y_Axis* `0x3431 0x00`	– 16.000 – 16.000 *– 16000 – 16000*	The 3-axis accelerometer Y-axis output signal. The value is in g (e.g., the force of gravity = 1.00 g). Note, this is a rounded value due to the filter setting and the rounding (to two decimal places) of the sensor sensitivity value (where raw mg/32767 = sensitivity).
Accel Z Axis *Accel_Z_Axis* `0x3432 0x00`	– 16.000 – 16.000 *– 16000 – 16000*	The 3-axis accelerometer Z-axis output signal. The value is in g (e.g., the force of gravity = 1.00 g). Note, this is a rounded value due to the filter setting and the rounding (to two decimal places) of the sensor sensitivity value (where raw mg/32767 = sensitivity).

MONITOR VARIABLES: VCL STATUS

VARIABLE	DISPLAY RANGE	DESCRIPTION
Master Timer *master_timer* `0x4E14 0x00`	0 – 42949673.0 Sec *0 – 4294967295*	The Manger Timer is a timer of the total time the controller has been powered (keyswitch = On time). The OS software controls the manger timer. The manger Timer cannot be reset.
VCL Error Module *last_vcl_error_module* `0x4E0B 0x00`	0 – 65535 *0 – 65535*	A VCL Runtime Error (fault code 0x41) will store additional information about the cause of a VCL runtime error in the VCL Error Module variable. The resulting non-zero values can be compared to the runtime VCL Module ID definitions of the controller (listed in the 1351 manual, VCL Error Module Definitions table), which should help pinpoint the VCL error that caused the runtime error.
VCL Error Code *last_vcl_error_code* `0x4E0A 0x00`	0 – 65535 *0 – 65535*	A VCL Runtime Error (fault code 0x41) will store additional information about the cause of a VCL runtime error in the VCL Error Code variable. The resulting non-zero values can be compared to the runtime VCL Error Code definitions of the controller (listed in the 1351 manual, VCL Error Code Definitions table), which should help pinpoint the VCL error that caused the runtime error.
VCL Error Function *last_vcl_error_function* `0x549C 0x00`	0 – 1000 *0 – 1000*	A VCL Runtime Error (fault code 0x41) will store additional information about the cause of a VCL runtime error in the VCL Error Function variable. The resulting non-zero values can be compared to the runtime VCL function table of the controller (listed in 1351 manual, Appendix A, or the Sys Info), which should help pinpoint the VCL error that caused the runtime error. The variable value is the index of VCL function table. For example, value 1 indicates that function MAP_TWO_POINT() cause the VCL error, value 2 indicate the CONTROL_EXTERNAL_POWER() function, and a value 3 indicates the AUTOMATE_ABS() function … etc.

5 — VEHICLE CONTROL LANGUAGE (VCL)

VCL OVERVIEW

The 1351 System Controller fully utilizes the Curtis Vehicle Control Language (VCL). VCL is an easy to program "C-like" language providing all the power and flexibility to control any application. VCL programs run protected and securely; monitored for proper operation to protect the system from erratic behaviors.

The 1351 carries over most of the VCL from the Curtis E and F-Series controllers and will be familiar to skilled developers. Several new features have been added to enhance the 1351's ability to act as a system controller and manger. Since the 1351 does not have any motor controller functions, VCL runs substantially faster on the 1351 than on any of the VCL controllers to date.

This chapter summarizes VCL as applicable to the 1351. It describes many of the functions specific to the 1351 system controller. For a more complete understanding of the functions and capabilities of VCL, see the original VCL Programmer's Guide, and VCL Common Functions Manual, available from Curtis. Also consult the Application Note: *Migrating VCL projects from E to F-series**

All VCL functions applicable to the 1351 are listed in Appendix A. The functions are also available in the *System Information* (sys info) an html-file that is available in the CIT's VCL Studio app (VCL Studio/Help menu/*System Information*). The *System Information* file is always the most up-to-date source of VCL functions, variables, constants, faults, parameters, CAN objects, etc. pertaining to the 1351. Its basis is from the actual software version of the 1351 System Controller.

SUMMARY OF VCL BASICS

- VCL is not case-sensitive:

  `put_driver(), Put_Driver(), and PUT_DRIVER()` are identical.
- Spaces in variable names are not allowed in VCL; use underscores in place of spaces.

  Example: `Nominal_Voltage` is the VCL name for the CIT/1313 HHP parameter Nominal Voltage.
- Functions are followed by parentheses; for example:

  `Reset_Controller()` is a function

  `Reset_Voltage` is a variable.
- Logical statements must be inside parentheses; examples:

```
IF (setpoint >50)
ELSE IF ((setpoint <20) & (temperature >100)).
```

- Comments are preceded by semicolons.

```
; This comment is invisible to VCL
```

** VCL programming assistance (and the Application Note: Migrating VCL projects from E to F-series) is available from Curtis. Contact your Curtis distributor or support engineer for help or training with the setup and VCL programming of the 1351 System Controller and its application.*

The original VCL manuals (principles of VCL) are embedded within the Curtis WinVCL program. WinVCL is the controller programming software for the Curtis serial-programmable controllers (e.g., the E/SE controllers). WinVCL is not applicable to the CAN-programmable 1351 System Controller and F-series controllers.

See Appendix C for the CANbus software and hardware applicable to program parameters, setup, and troubleshoot the 1351 System Controller.

Variables and Constants

All 1351 VCL user variables and parameters are 32-bit signed integers. This greatly simplifies many mathematical operations, as the risk of integer overflow is greatly reduced. Note that Constants are limited to 16 bits.

VCL VARIABLE TYPES AND MEMORY

VCL provides dedicated memory space to store custom variables. The variable type determines whether they are stored and used in volatile memory or stored non-volatile memory.

VCL uses 2 types of memory.

1. RAM (Random Access Memory) which is volatile memory for the VCL run-time variables.
2. FRAM, (Flash Random Access Memory) is non-volatile memory (NVM) for retaining variable values for power-up/down logs.

RAM variables are stored only while power is on; they are lost at power-down. They must be initialized upon power-up by explicit VCL assignments. The *user* and *autouser* variables are in this category (i.e., `User1 = 12`).

FRAM variables can be saved at power-down and restored at power-on. The P_Users (parameters) and NVUser variables are in this category (i.e., `P_User12` and `NVUser12`).

At power-up, all variables values are loaded in to RAM for run-time operation.

P_User parameters are a special type of variable that is intended to be used to create OEM defined CIT/1313 HHP programmer parameters. These parameters can be defined up to 32-bit by using the `P_User` (1-300) variables. These variables are typically written to FRAM through the CIT or 1313 HHP Programmer app (i.e., when a dealership technician changes these created parameter settings using the 1313 HHP). These `P_User` variables can be used/changed in the VCL code, but changing a `P_User` value with VCL will only change the variable value in RAM and will not change the value in FRAM. Thus, these variables are intended for creating and defining CIT and 1313 HHP Programmer app parameters only.

VCL can modify the control mode parameters in RAM by using the VCL variable name for the programmable parameter. For example,

```
Driver_1_Ramp_Up_Time = 300    ; Change Ramp-up Rate to 300 ms
```

will change the RAM value of the Driver 1 Ramp Up; the new value will be used in determining the ramp-up rate during operation. However, the value of the parameter's stored in FRAM remains unchanged; so when the controller is turned off, the run-time RAM value will be lost. The next time the 1351 system controller is powered back on, the "old" value of Ramp Up will be restored from FRAM memory. To save this RAM value in VCL, the VCL writer must use the *NVM_Write_Parameter* function ((i.e., `NVM_Write_Parameter(Driver_1_Ramp_Up_Time)`).

Once again, Parameter values that are changed by using the CIT or 1313 HHP programmer are saved directly to FRAM memory. The CIT and 1313 HHP changes will be retained and restored the next time the 1351 system controller is powered back on.

Table 21 summaries the available 1351 system controller VCL variables by memory type.

Table 21 VCL variables by memory type

Memory	Quantity	Name	Object Range	Save Type
FRAM	300	P_User1 – P_User300	0x4000 – 0x412B	Manual Save
RAM	300	Autouser1 – Autouser300	0x4200 – 0x432B	Lost at shutdown
FRAM	250	NVUser1 – NVUser160	0x4400 – 0x44F9	Auto save
RAM	120	User1 – User120	0x4500 – 0x4577	Lost at shutdown

SDO Write Message

To retain parameter values in non-volatile memory (NVM) via CANopen SDO write messages, write a non-zero value to SDO_NVM_Write_Enable (Object Index 0x2008, sub-index 0x00) before changing parameter values. This will cause parameter changes to be written to non-volatile FRAM memory. Note, having the SDO_NVM_Write_Enable set to zero only saves the parameter changes to ephemeral (RAM) memory.

Quick Link:
SDO Write and NVM (info) p. 33

VCL Function Examples

These functions are applicable to the 1351 System Controller. See the *System Information* (sys info) file that is available in the CIT, and listed in Appendix A.

`Set_Watchdog_Timeout()` p. 87

`Set_Watchdog_Fault_Action()` p. 88

`Kick_Watchdog()` p. 89

`MAP_TWO_POINTS()` p. 90

`Automate_Frequency_Output ( )` p. 91

`Set_Accelerometer_Range ( )` p. 92

VCL, Watchdog Timer and Faults

Using VCL code, watchdog timers can be setup to ensure that specific parts of the VCL code are operating properly. The watchdog can be started, stopped and "kept alive" in up to five separate places in the code. A Watchdog fault action definition will allow the VCL designer to select what happens when the active Watchdog times out.

FUNCTION `Set_Watchdog_Timeout()` *Sets the time-out in milliseconds for any one of five watchdog timers.*

This function will set the time-out in milliseconds for any one of five watchdog timers available. When the function is called, the specific watchdog timer is cleared and the time-out threshold is updated.

The watchdog must not be running when this function is called or the Watchdog fault and fault actions will be activated. After the first `Kick_Watchdog()` has been called for a timer, the watchdog cannot be stopped or set to a new timeout. This is to prevent faulty code from re-enabling the watchdog constantly and thus subverting its function.

When a Watchdog timer runs out (not kicked in time) VCL will halt and the appropriate Watchdog fault (1 through 5) will be called. The system will then activate the particular fault actions defined by the VCL programmer.

SYNTAX `Set_Watchdog_Timeout(WD#, Time)`

ARGUMENTS:

`WD#`	One of the 5 watchdog timers, WD1 through WD5
`Time`	The watchdog's time threshold, 20 ms (min.) to 60 s (max.) *20 – 60000*

RETURNS: None

REPORTED ERRORS:

`BAD_ID`	WD# out of range
`PARM_RANGE`	Parameter (Time) out of range

FUNCTION `Set_Watchdog_Fault_Action()` *User defined actions when the watchdog timer times-out.*

When a watchdog timer times-out, the user-defined actions are launched by the OS. These watchdog fault actions are setup in identical fashion to the User Fault Action in VCL.

SYNTAX `Set_Watchdog_Fault_Action(WD#, Action_Bits)`

ARGUMENTS:

`WD#` One of the 5 watchdog timers, WD1 through WD5

`Action_Bits` The following table shows the available action bits and the predefined labels that can be used directly or combined to make new actions.

Fault Action Name	Action Code (Bits)
No Action	0x00000
Shut Down Driver 1	0x00001
Shut Down Driver 2	0x00002
Shut Down Driver 3	0x00004
Shut Down Driver 4	0x00008
Shut Down Driver 5	0x00010
Shut Down Driver 6	0x00020
Shut Down Digital Out 1	0x00040
Shut Down Digital Out 2	0x00080
Shut Down Driver 7	0x00100
Shut Down Driver 8	0x00200
Shut Down Driver 9	0x00400
Shut Down Driver 10	0x00800
Shut Down Driver 11	0x01000
Shut Down Driver 12	0x02000
Shut Down Digital Out 3	0x04000
Shut Down Primary Driver	0x000FF
Shut Down Driver Supervisor	0x07F00
Shut Down Driver All	0x07FFF
Shut Down Ext 5V	0x08000
Shut Down Ext 12V	0x10000*
Shut Down Safety Out	0x20000*
Shut Down All	0x3FFFF*

RETURNS: None

REPORTED ERRORS:

`BAD_ID` WD# out of range
`PARM_RANGE` Parameter (Time) out of range
`PT_RANGE` Unknown Parameter

**Note: VCL `CONSTANTS` are limited to 16-Bit, although OS processing is 32-bits. These VCL fault actions, with bits in the* upper word*, must be either bit shifted or split & added during VCL runtime, e.g.:*

```
Shut_Down_Ext12V = 0x1 << 16          ; Results in 0x10000
Shut_Down_Safety_Out = 0x2 << 16      ; Results in 0x20000
Shut_Down_All = (0x3<<16) + 0xFFFF    ; Results in 0x3FFFF
```

FUNCTION `Kick_Watchdog()` *Starts and reset the specified watchdog timer.*

The first call of this starts the watchdog timer (required). Succeeding calls will reset (re-starts) the watchdog timer timing.

SYNTAX `Kick_Watchdog(WD#)`

ARGUMENTS:

`WD#`	One of the 5 watchdog timers, WD1 through WD5

RETURNS: None

REPORTED ERRORS:

`Bad_ID`	WD# out of range

Code Example: Two watchdogs setup.

```
Set_watchdog_timeout(WD1, 200)   ; 200 ms timer)
Set_watchdog_timeout(WD2, 1000)  ; 1000 ms (1 min) timer
Set_watchdog_fault_action(WD1, ShutdownAll)
Set_watchdog_fault_action(WD2, ShutdownDriver1)

; Do anything else to get the system setup, the watchdog(s)
; is not running yet

MainLoop:
;do some functions
Call Update_Everything
kick_watchdog(WD1)  ; Set, Reset WD1 in the main loop
Goto MainLoop

Update_Everything:
;do some functions
kick_watchdog(WD2)  ; Set, reset WD2
return
```

FUNCTION `MAP_TWO_POINTS()`

This function interpolates values between two points, Y1 and Y2; based upon an X input parameter.

Interpolating based on X1 and X2.

Typical Usage:

1. Calculate a value framed between two X-axis points projected across two Y points, where the functions' output is the value of the Y-axis intersection.
2. Extend the 7 pair limitation of the *Setup_Map* function by the use of multiple If, If Else statements on segments of an XY array.

SYNTAX `Map_Two_Points(X,X1,X2,Y1,Y2)`

ARGUMENTS:

`X`	Input.
`X1`	Input point X1.
`X2`	Input point X2.
`Y1`	Output point Y1.
`Y2`	Output point Y2.

RETURNS:

`n`	Mapped value.

REPORTED ERROR:

`None`

Code Example: Convert a controller value, such as Steer_Angle, to a voltage.
The input (X) is the Steer_Angle variable, where X1-X2 is an angle between 0 and 90 Degrees. The output is the interpolated value across Y-axis points in Volts. Enable the expansion of the Setup_Map(16) function beyond the seven (7) point-pair by the use of multiple If, If Else statements on segments of a two point XY array. For Example:

```
If(X < 1)
{
Y = Map_2_Points(X, X1, X2, Y1, Y2)
}
Else If(0<= X <= 1)
{
Y = Map_two_Points(X, X1, X2, Y1, Y2)
}
Else If(1< X <= 2)
{
Y = Map_Two_Points(X, X1, X2, Y1, Y2)
}
Else If(2< X <= 3)
{
Y = Map_Two_Points(X, X1, X2, Y1, Y2)
}
Etc.
```

FUNCTION `automate_frequency_output()`

This function sets up the Driver 10 (pin 33) PWM output to yield a frequency proportional to the input variable at an execution rate of 16 ms. This output can be used to drive an electronic speedometer or tachometer.

An additional parameter, **Duty Cycle** (VCL variable, *Frequency_Output_Duty_Cycle*), works in conjunction with the Automate_Frequency_Output() function to modify the duty cycle of the output. The default value of *Frequency_Output_Duty_Cycle* is 50%.

SYNTAX `automate_frequency_output (Source,MinInput, MaxInput, MinOutput, MaxOutput)`

ARGUMENTS:

`Source`	Index of source value to be the input.
`MinInput`	Holds the minimum value for the input.
`MaxInput`	Holds the maximum value for the input.
`MinOutput`	Holds the minimum value for the output in Hz.
`MaxOutput`	Holds the maximum value for the output in Hz.

NOTE: The frequency range is 0–4000 Hz, with 4 Hz being the minimum active frequency, and 0–3 Hz = Off.

RETURNS:

`0` = PWM not automated.

`1` = PWM automated.

ERROR CODES: Param_Range is returned when a parameter is out of range.

PT_Range is returned when the Parameter Table Index is out of range.

Example: To set up Driver10 to output a 500 Hz to 1500 Hz PWM signal, at duty cycle = 25%, for motor speed in the range 100 rpm to 4000 rpm:

```
Create ABS_Motor_Speed variable   ;1351 is not a motor controller,
                                  ;e.g., Obtain the absolute (±) value from a CAN
                                  ; message or PDO mapping
Frequency_Output_Duty_Cycle = 25  ;set duty cycle = 25%.
                                  ;Alternatively, set in the parameter menu.
Automate_Frequency_Output(ABS_Motor_Speed,100,4000,500,1500)
```

FUNCTION `Set_Accelerometer_Range()`

This function is used to set the scale (from 2g to 16g) of the accelerometer.

SYNTAX `Set_Accelerometer_Range(Accel#, Scale)`

ARGUMENT:

Accel#:	Default to 1 now (only one accelerometer installed)
Scale:	0 = 2g
	1 = 4g
	2 = 6g
	3 = 8g
	4 = 16g

RETURNS:

`0` = Set accelerometer fail (Not supported Accel# or Scale value)

`1` = Set accelerometer successfully

ERROR CODES:

`BAD_ID`	Invalid Accel#
`PT_RANGE`	Scale is out of range (should be 0 - 4)

Example:

CAN FUNCTIONS (VCL SETUP)

Beyond the parameters in the Programmer app CAN menu, setting up the 1351 to use CAN messages *requires* a VCL program. The applicable 1351 CAN functions are described by topic in this section. An application's requirements will determine whether a function will be used, as not all applications will utilize every VCL function.

- Port1(primary) has 30 receive and 30 transmit mailboxes that can be assigned/used by VCL.
- Port2(supervisor), has 10 receive and 10 transmit mailboxes that can be assigned/used by VCL
- Attempting to assign more that this (using the `Assign_Mailbox()` function will return an error.
- There are four TPDO and four RPDO mailboxes that can be used and are shared between Port 1 and Port 2.

NMT Control

When a 1351 CAN port is enabled to be a manger, it can then send Network Management (NMT) commands to ancillary controllers. The 1351 has two CAN ports, CAN1 (*Port_1_Node_Id*) and CAN2 (*Port_2_Node_ID*). The NMT state of each port is available by its specific read-only monitor variable, *port_2_nmt_state* and *port_2_nmt_state*.

FUNCTION `send_NMT()`

Control the state of the CANopen ancillary(s).

When a port is enabled to be a manger, this function is used to control the state of salve(s).

NOTES

1: This function can control a single device or all devices on can bus.

SYNTAX `send_NMT (Port, Node_ID, NMT_Command)`

ARGUMENT:

`Port`	constant CAN_PORT_1, (0, 0X0), or constant CAN_PORT_2 , (1, 0X1)
`Node ID`	Node_ID of CAN ancillary device(use constant GLOBAL to control all device on the bus)
`NMT Command`	Following constant can use: GO_OPERATIONAL GO_PREOPERATIONAL STOP_COOMUNINCATIONS RESET_COMMUNINCATION RESET_DEVICE

RETURNS:

`0` = Function did not successfully execute

`1` = Function successfully executed

ERROR CODES:

`BAD_ID`	PORT# or Node ID# out of range.

EXAMPLE:

FUNCTION `Setup_NMT_State()`

This function is used to control the NMT state of two CAN ports.

NOTES

1: Control the NMT state of two CAN port separately

SYNTAX `Setup_NMT_State(Port, State)`

ARGUMENT:

`Port`	constant CAN_PORT_1 (0, 0x0) or constant CAN_PORT_2 (1,0x1)
`State`	NMT state to be set

RETURNS:

`0` = Setup NMT state successfully

`1` = Setup NMT fail

ERROR CODES:

EXAMPLE:

Node Guarding

A controller can be setup to monitor the heartbeats of every node on the system. It will calculate the time between heartbeats and generate error flags when they are lost. It will also record the CANopen State of the Node.

The 1351 can monitor up to 16 Nodes.

FUNCTION `Enable_Node_Guarding()` *Turns On/Off heartbeat monitoring.*

`Disable_Node_Guarding()` *Disable will clear out ALL nodes from the guarding list.*

SYNTAX `Enable_Guarding (Port)`

`Disable_Node_Guarding (Port)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].

RETURNS:

`0` = function did not successfully execute

`1` = function successfully executed

REPORTED ERRORS:

`BAD_ID`	PORT# out of range

FUNCTION `Setup_Heartbeat()` *Sets the expected rate for a required Node ID's heartbeat.*

If the Heartbeat from this node exceeds 2 timeout periods (one missed heartbeat) an error flag is set. Time is set in milliseconds

SYNTAX `Setup_Heartbeat(Port, Node_ID, Time)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].
`Node ID`	Node_ID of CAN ancillary device
`Time`	Expected heartbeat rate in ms

RETURNS:

`0` = function did not successfully execute

`1` = function successfully executed

REPORTED ERRORS:

`BAD_ID`	PORT# or Node ID # out of range
`PT_RANGE`	No more monitoring slots available (16 maximum)
`ERROR_ASSIGN_MB`	assign the receive mailbox for heartbeat monitor fail (no more mailboxes available or the identifier has been assigned to another mailbox)

FUNCTION `State = Get_NMT_State()` *State of the device requested*

Returns the state of the device requested by monitoring the Heartbeat messages

SYNTAX `Get_NMT_State (Port, Node_ID)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].
`Node_ID`	The Ancillary Node ID

RETURNS:

`-2` = Function did not successfully execute

`-1` = Never receive the heartbeat message

`State` return the latest state of ancillary device:

`0` = initialization,

`4` = stopped

`5` = operational

`127` = pre-operational

REPORTED ERRORS:

`BAD_ID`	The ancillary is not included in node guarding lists.

FUNCTION `Rate = Get_Heartbeat_Time()` *If the heartbeat has not been lost, this function will return the time between the last two heartbeats.*

SYNTAX `Get_Heartbeat_Time(Port, Node_ID)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].
`Node_ID`	The Ancillary Node ID

RETURNS:

`-1` = The specified device is not included in node guarding slots

`0` = Never receive the heartbeat message from specified device

`Time`	If normal, return the time between last two heartbeats; If the heartbeat has been lost, return the time elapsed from last heartbeat.

REPORTED ERRORS:

`BAD_ID`	The ancillary is not included in node guarding lists.

FUNCTION `Status = Check_Heartbeat_Status()` *Returns the status of the heartbeat timeout check the time between the last two heartbeats.*

SYNTAX `Check_Heartbeat_Status (Port, Node_ID)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].
`Node_ID`	The Ancillary Node ID

RETURNS:

`-1` = The specified device is not included in node guarding slots

`1` = faulted

`0` = no fault on this Node ID.

If the GLOBAL ID is used, the function will return 1 if ANY Node is beyond it set timeout

REPORTED ERRORS:

`BAD_ID`	The ancillary is not included in node guarding lists.

Emergency Message Monitoring

Emergency messages are collected by the CAN manger (when enabled) and placed into an indexed buffer. Messages can be retrieved by the NODE ID or the depth of the buffer can be retrieved and messages pulled out one-by-one.

The buffer is 16 emergency messages deep.

When a "code" is retrieved from the buffer, its data is placed into pre-defined global VCL variables and the message is pulled from the stack. The user can then check for the next message until the index returned is null (valueless).

FUNCTION `Enable_Emergency_Message_Monitor()` *Starts Emergency Message monitoring.*

`Disable_ Emergency_Message_Monitor()` *Disable will clear out ALL nodes from the monitoring list.*

SYNTAX `Enable_Emergency_Message_Monitor(Port)`

`Disable_ Emergency_Message_Monitor(Port)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6]

RETURNS:

`0` = did not execute.
`1` = successfully executed

REPORTED ERRORS:

`BAD_ID`	PORT out of range

FUNCTION `Setup_Emergency Message_Monitor()` *Monitor a Node for Emergency messages*

Place this Node ID into the list of devices that will be monitored for Emergency messages.

SYNTAX `Setup_Emergency Message_Monitor(Port, Node_ID)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6]
`Node_ID`	The Ancillary Node ID

RETURNS:

`0` = Set up emergency message monitor fail
`1` = Setup emergency message success

REPORTED ERRORS:

`BAD_ID`	PORT# or Node ID # out of range
`PT_RANGE`	No more monitoring slots available
`ERROR_ASSIGN_MB`	Assign the receive mailbox for emergency message monitor fail (no more mailboxes available or the identifier has been assigned to another mailbox)

FUNCTION `Error_Index = Check_CANopen_Emergency()` *Checks for message on port & device*

Check if there was an emergency message active on this port and ancillary device.

If the Node ID = GLOBAL (0) than any emergency message will be found.

Note, this "Error_Index" example is a variable name chosen by the VCL writer that is set equal to the function in order to identify the emergency message by the ancillary Node ID. The term "Error_Index" is used here to indicate how to use this function in VCL ("Error_Index" it can be any VCL appropriate variable name).

SYNTAX `Check_CANopen_Emergency(Port, Node_ID)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6]
`Node_ID`	The Ancillary Node ID or Global for all ancillary nodes

RETURNS:

`-1` = The specified device is not included in the emergency message monitor slots

`0` = no Emergency Messages

ERROR_INDEX:

`1 or greater` = the number of messages buffered for that Node ID, (Basically, the length of the buffer is returned)

REPORTED ERRORS:

`BAD_ID`	The ancillary is not included in emergency message monitor lists.

FUNCTION `Status = Get_CANopen_Emergency_Message()` *Full 8 bytes of the message, and ID*

Returns the first 8 bytes of an emergency message and Node_ID and places them into the following pre-defined VCL variable(s) as illustrated, as Status. The buffer is 16 messages, e.g., *Status_01* through *Status_16*. These are the messages as indexed by "Error_Index", described above.

SYNTAX `Get_CANopen_Emergency_Message(Error_Index)`

ARGUMENTS:

Each call to the function will re-fill these variables.

Or, specify the VCL variable, Status (as illustrated here), with the "Error-Index" argument.

RETURNS:

`1` = get data and update into predefined VCL variable.

```
Emergency_Message_Error_Code       (bytes 0 and 1)
Emergency_Message_Error_Register   (byte 2)
Emergency_Message_Data_1           (byte 3)
Emergency_Message_Data_2           (byte 4)
Emergency_Message_Data_3           (byte 5)
Emergency_Message_Data_4           (byte 6)
Emergency_Message_Data_5           (byte 7)
Emergency_Message_Node_ID
```

`0` = the buffer is empty at that location

REPORTED ERRORS:

`BAD_ID`	Error_index out of range

FUNCTION `Status = Clear_Emergency_Message_Buffer()` *Remove messages from the emergency message buffers.*

This function can be used to remove messages from the emergency message buffers. This might be needed if messages have not been removed in a long time and the buffer is full. Note that buffers will roll over and the oldest messages are lost/ replace with the latest messages.

SYNTAX `Clear_Emergency_Message_Buffer(Port, Node_ID, Index)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].
`Node_ID`	The Ancillary Node ID or Global for all ancillary nodes
`Index`	Specified index of emergency message. If the Index = 0 , then all messages are removed

RETURNS:

`0` = function did not successfully execute
`1` = function successfully executed

REPORTED ERRORS:

`BAD_ID`	The ancillary is not included in emergency message monitor lists.

SDO Management

Ten VCL functions are described here to manage Service Data Objects (SDOs)*. SDOs are used to either read data from another device to send data to the device. In either case, the 1351 OS will send an SDO and then wait for and buffer the return. The VCL needs only to poll the SDO buffer for the response. Each SDO sent will be provided a Reception_ID. This ID is used to check the status of the request and to pull the data from the input reception buffers. The buffer is 16 messages deep. Reading from the buffer clears out that message.

FUNCTION `Reception_ID = Request_SDO_Read()` *Send an SDO read request to a specific node*

This function sends an expedited SDO request to read a value from a specific device identified by its Node ID. The function will return a handle (i.e., *Reception_ID*) to the 1351's receive buffer location where the data will be placed. Each buffer location is 32 bits (4 bytes) wide.

The *Reception_ID* is used to check the status of the particular request and pulls the data from the input reception buffers. Once a message is read, it is pulled from the request buffer. The buffer is 16 messages deep. If a buffer is full, the Request function will return an invalid ID of 0 (valid IDs are 1-16), meaning no more messages can be requested until the pending messages (1-16) are checked and pulled from the buffer, or the buffer is cleared.

When each request is made, an individual timer is started. If the timer expires, a fault will be set. Checking the status of a timed out value will return a fault code (see the error code table for *Check_SDO_Read*)

SYNTAX `Request_SDO_Read (Port, Node_ID, Index, Sub-Index, Timeout)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].
`Node ID`	Node_ID of the CAN ancillary device
`Index`	The CANopen Index of the parameter to read
`Sub-Index`	The CANopen sub-Index of the parameter to read
`Timeout`	Specify the maximum time (in milliseconds) to receive a response.

RETURNS:

`0`	Function did not successfully execute
`Handle:`	SDO Read Handle

REPORTED ERRORS:

`BAD_ID`	PORT# or Node ID # out of range
`PT_RANGE`	No more SDO read slots available
`ERROR_ASSIGN_MB`	Assign the receive mailbox for SDO read (no more mailboxes available or the identifier has been assigned to another mailbox)

** SDO Service Data Object (SDO). A SDO is a low priority message used to transfer data sets from a client to a server and vice versa. Several types of data transfer are available, with the Client (manger controller) taking the initiative for a transfer. The SDO process is used primarily to read or write to an object index of a Server (ancillary controller).*

FUNCTION `Check_SDO_Read()` *Check if a specific request for SDO data has been fulfilled*

SYNTAX `Check_SDO_Read(handle)`

ARGUMENTS:

`handle` Specify the buffer handle provided by the `Request_SDO_Read` function

RETURNS:

`-1` = Invalid read handle or wait for messsage timeout

`0` = waiting for the message to be received. (i.e., cannot send another read request until the previous request has populated the buffer). Needs a check and/or delay in the VCL main loop to prevent over-requesting.

`1` = Message Received

`2 - 11` = Abort code (see table)

Error Message	Return	CANopen Code	Description
BAD_SDO_COMMAND	2	0504 0001h	Anytime the SDO command byte is unknown or the SDO structure is incorrect, including wrong length (not 8 bytes, as required by the CCIS-V2 device)
CRC_FAILED	3	0504 0004h	The CRC failed on the file.
UNSUPPORTED_ACCESS	4	0601 0000h	Attempt to access an object using an incorrect access level.
OBJECT_IS_WRITE_ONLY	5	0601 0001h	Try to read a write-only object
OBJECT_IS_READ_ONLY	6	0601 0002h	Try to write a read-only object
UNSUPPORTED_OBJECT	7	0602 0000h	Object is not in the device, or an error occurred while in a block, or segmented SDO transfer.
PDO_MAPPING_ERROR	8	0604 0041h	Any PDO mapping failure, including a non-map-able parameter, the device is not in Pre-Operational mode, the length byte is wrong or there are too many bytes in the PDO mapping.
VALUE_OUT_OF_RANGE	9	0609 0030h	The value is out-of-range
GENERAL_ERROR	10	0800 0000h	Any other generic reason not listed elsewhere.
INVALID_DEVICE_STATE	11	0800 0022h	Some parameters cannot be changed if the system is moving or is engaged (interlock off) or if the fault has not occurred

REPORTED ERRORS:

`BAD_ID` argument handle out of range

FUNCTION `Variable = Get_SDO_Length()` *Return the number of valid bytes in the SDO reception*

SYNTAX `Get_SDO_Length(Handle)`

ARGUMENTS:

`handle` Specify the buffer handle provided by the request function

RETURNS:

`-1 =` Invalid read handle

LENGTH: Number of valid bytes that were returned by the SDO request

REPORTED ERRORS:

`BAD_ID` argument handle out of range

FUNCTION `Variable = Get_SDO_Data()` *Pull the data from the request receive buffer*

SYNTAX `Get_SDO_Data(handle)`

ARGUMENTS:

`handle` Specify the buffer handle provided by the request function

RETURNS:

`-1 =` Invalid read handle

DATA: Returns a single 32 bit value and removes the message for the buffer

REPORTED ERRORS:

`BAD_ID` argument handle out of range

Code Example

```
If (SDO_Send_State = 0)   ; check if SDO_Send_State = 0, request if only = 0
  {
  Request_Switches_ID = Request_SDO_Read (CAN_PORT_1,38,3226,00,60)
                        ;Read request handle = Request_Switches_ID
                        ;CAN_PORT_1 = CAN1
                        ;Ancilary Node ID = 38d, 0x26h
                        ;Switches variable = 0x3226 0x00.
                        ;See E/SE series Sys Info.
                        ;e.g., ancillary is an E/SE series controller
                        ;60 millisecond timeout
  SDO_Send_State = 1    ;reset SDO_Send_State = 1
  }
Else
  If((Check_SDO_Read(Request_Switches_ID) = 1)
  {
  Get_SDO_Data(Request_Switches_ID)      ;get the Switches data
  SDO_Send_State = 0                     ;reset SDO_Send_State = 0
  }                                      ;to allow re-entry when
                                         ;message data is removed from
                                         ;the buffer.
```

FUNCTION `Write_ID = Write_SDO()` *Each SDO write will provide a Write_ID handle*

This function sends an expedited SDO request to write to change a parameter in a specific device identified by its Node ID. The function will return a handle (i.e., *Write_ID*) to the 1351's write buffer location where the data will be placed. Each buffer location is 32 bits (4 bytes) wide. The VCL can have up to 16 pending SDO writes.

The Write ID is used to check the status of the SDO write.

Once a message is responded to from the ancillary and checked, it is pulled from the buffer. The buffer is 16 messages deep. If a buffer is full, the Write SDO function will return an invalid ID of 0 (valid IDs are 1-16), meaning no more messages can be sent until the pending messages are checked and pulled from the buffer, or the write buffer is cleared.

When each write is made, an individual timer is started. If the timer expires before the write is successfully executed, a fault will be set. Checking the status of a timed-out value will return a fault code (see the error code table for *Check_SDO_Write*, next page).

SYNTAX `Write_SDO(Port,Node_ID,Index,Sub-Index,Length, Variable,Timeout)`

ARGUMENTS:

`Port`	constant CAN_PORT_1 (0, 0x0) [pins 3 & 4]. or, constant CAN_PORT_2 (1, 0x1) [pins 5 & 6].
`Node ID`	The Ancillary Node ID
`Index`	The Index of the parameter to read
`Sub-Index`	The sub-Index of the parameter to read
`Length`	The length of the parameter (16 bit or 32 bit)
`Timeout`	Specify the maximum response time (in milliseconds)

RETURNS:

`0 =` function did not successfully execute

HANDLE: SDO write handle.

REPORTED ERRORS:

`BAD_ID`	PORT# or Node ID # out of range
`PT_RANGE`	No more SDO write slots available or variable source is out of parameter table range or invalid data length
`ERROR_ASSIGN_MB`	Assign the receive mailbox for SDO write (no more mailboxes available or the identifier has been assigned to another mailbox)

FUNCTION `Check_SDO_Write()` *Check if a specific write for SDO data has been fulfilled.*

SYNTAX `Check_SDO_Write(handle)`

ARGUMENTS:

`handle` Specify the buffer handle provided by the *SDO_Write* function

RETURNS:

`-1` = Invalid SDO write handle or wait for response messsage timeout

`0` = Waiting for the response from the ancillary device to be received (i.e., cannot send another write request until the previous request has been removed the buffer). Needs a check and/or delay in the VCL main loop to prevent over-requesting.

`1` = SDO Write was successful

`2 - 11` = Abort code (see table)

Error Message	Return	CANopen Code	Description
BAD_SDO_COMMAND	2	0504 0001h	Anytime the SDO command byte is unknown or the SDO structure is incorrect, including wrong length (not 8 bytes, as required by the CCIS-V2 device)
CRC_FAILED	3	0504 0004h	The CRC failed on the file.
UNSUPPORTED_ACCESS	4	0601 0000h	Attempt to access an object using an incorrect access level.
OBJECT_IS_WRITE_ONLY	5	0601 0001h	Try to read a write-only object
OBJECT_IS_READ_ONLY	6	0601 0002h	Try to write a read-only object
UNSUPPORTED_OBJECT	7	0602 0000h	Object is not in the device, or an error occurred while in a block, or segmented SDO transfer.
PDO_MAPPING_ERROR	8	0604 0041h	Any PDO mapping failure, including a non-mapable parameter, the device is not in Pre-Operational mode, the length byte is wrong or there are too many bytes in the PDO mapping.
VALUE_OUT_OF_RANGE	9	0609 0030h	The value is out-of-range
GENERAL_ERROR	10	0800 0000h	Any other generic reason not listed elsewhere.
INVALID_DEVICE_STATE	11	0800 0022h	Some parameters cannot be changed if the system is moving or is engaged (interlock off) or if the fault has not occurred

FUNCTION `Status = Clear_SDO_Read_Buffer()` *Clears the read buffer*

`Status = Clear_SDO_Write_Buffer()` *Clears the read buffer*

These functions can be used to remove all messages from the respective buffers. This might be needed if messages have not been removed in a long time and the buffer is full. Note that once the buffers are full, the Request and Read VCL command will return an invalid ID (0)

SYNTAX `Clear_SDO_Read_Buffer()`

`Clear_SDO_Write_Buffer()`

ARGUMENTS:

`None`	Clear all buffers associated with both CAN ports.

RETURNS: None

REPORTED ERRORS:

FUNCTION `Status = Remove_SDO_Read()` *Clears a single read buffer*

`Status = Remove_SDO_Write()` *Clears a single read buffer*

Allow the VCL program to remove a single Read or Write SDO from the appropriate buffer.

SYNTAX `Remove_SDO_Read(handle)`

`Remove_SDO_Write(handle)`

ARGUMENTS:

`handle`	Specify the buffer handle provided by Request_SDO_Read/write_SDO function

RETURNS:

`-1` = Invalid SDO read handle

`1` = function successfully executed

REPORTED ERRORS:

`BAD_ID`	Argument handle out of range

CREATING AND USING CAN MAILBOXES (VCL SETUP)

CAN Mailboxes are where the 1351 operating system (OS) stores, sends and receive information over the CANbus. Since there is a limited number of Transmit (TX)[1] and receive (RX)[2] buffers and masks, CAN mailboxes must be first allocated. The OS provides a handle for each allocated mailbox, which allows the VCL program to define the mailbox, set it up and use it.

The 1351's CAN mailbox functions are the same as for the Curtis F-series controllers, and are thus different than the E/SE series of controllers. The 1351 System Controller differs in the fact it has two CAN ports that share the available mailboxes and PDO setups between CAN port 1 and CAN port 2.

Quick Link:
PDO Setups p. 71

There are four configurable TPDOs[3] and four RPDOs[4] on CAN Port 1.

In Addition to the normal F series CAN mailbox setup functions, the 1351 offers specific functions to simplify setting up mailboxes for the CANopen protocol. (The VCL writer may still use the normal VCL mailbox functions for other protocols or if you require features outside these CANopen specific functions)

If a CANopen function is used to start the setup of a CAN mailbox, the VCL writer must use only these CANopen functions for that mailbox. Likewise, these CANopen functions cannot be used with the normal CAN mailbox function

---

[1] RX (Server to Client), as per CANopen nomenclature. Messages from the manger controller to the ancillary controllers/devices

[2] TX (Client to Server), as per CANopen nomenclature. Messages from the ancillary controllers/devices to the manger controller

[3] RPDO Receive Process Data Object (RPDO). Data received by the Consumer from Producer communication, (e.g., the manger controller receives PDO data from the ancillary controller).

[4] TPDO Transmit Process Data Object (TPDO). Data transmission by the PDO Producer to PDO Consumer, (e.g., the ancillary controller transmits PDO data to the manger controller).

Reminder: TX and RX are from the perspective of the ancillary-device, as per CANopen specifications.

Configuration of a Transmit Mailbox

FUNCTION `Setup_CANopen_Transmit_Mailbox()` *Sets up the initial parameters for a CANopen mailbox*

This CANopen specific function sets up the initial parameters for a CANopen mailbox.

It serves the same purpose as the generic setup function, but simplifies the setup for CANopen users.

This function MUST BE called before defining the data for the mailbox or trying to use the mailbox.

SYNTAX `Setup_CANopen_Transmit_Mailbox(Handle,Message_type,Node_ID,Method)`

ARGUMENTS:

`Handle`	The mailbox ID
`Message type`	NMT, SYNC, EMR, SDO_TX, SDO_RX, TPDO, RPDO
`Node ID`	Node ID to include in the message
`Method`	If set to EVENT (=0) then this mailbox with send on the send command (One-time, or non-recurring message) If set from 4-16,000, it will send every 4mS to 16sec, based on this value

RETURNS:

`0`	Mailbox not setup
`1`	Mailbox setup

REPORTED ERRORS:

`Bad_ID`	Handle out of Range

FUNCTION

`Define_CANopen_Transmit_Data()` *Used to define the data sent from the mailbox.*

This function is used to define the data sent from the mailbox.

Up to 64 bits can be loaded into a single message.

Multiple `Define_CANopen_Transmit_Data` calls are used to add additional data to the message, up to the 64 bit limit (no more than 16 defines per mailbox).

The data is ordered in the same order as the functions are called starting at Bit 1 of Byte 1.

The Mailbox must be disabled to use the *Define Data* function (automatically done by the any setup transmit mailbox function)

SYNTAX

`Define_CANopen_Transmit_Data(Handle,Data_Source,Start_Bit,Num_Of_Bits)`

ARGUMENTS:

`Handle`	Holds the ID of the mailbox
`Data_Source`	Address of Data to be sent, such as P_USER_11
`Start_Bit`	The first, lowest order bit of data source to be packed (0 = bit1 and 31 = bit32)
`Num_OF_Bits`	How many bits are packed by this function into the message data 1 – 64

RETURNS:

`0`	Data format failed (too many bits or not enough bits in the data source
`1`	Data format set

REPORTED ERRORS:

`Bad_ID`	Handle out of Range
`PT_RANGE`	Parameter index out of range
`PARAM_RANGE`	Invalid start bits or length (exceed the parameter length) or the mailbox has been defined too may times (over 16)
`UNIT_BUSY`	The mailbox has been enabled
`BAD_MO_LEN`	The data bits to define exceed the number of undefined bits of the mailbox

Configuration of a Receive Mailbox

FUNCTION

`Setup_CANopen_Receive_Mailbox()` *Sets up the initial parameters for a CANopen mailbox.*

This CANopen specific function sets up the initial parameters for a CANopen mailbox.

It serves the same purpose as the generic setup function, but simplifies the setup for CANopen users.

This function MUST BE called before defining the data for the mailbox or trying to use the mailbox.

SYNTAX

`Setup_CANopen_Receive_Mailbox(Handle,Messsge_type,Node_ID,Mask,Service)`

ARGUMENTS:

`Handle`	The mailbox ID
`Message type`	NMT, SYNC, EMR, SDO_TX, SDO_RX, TPDO1-4, RPDO1-4
`Node ID`	Node ID to include in the message
`Mask`	Set the bits in the 11 Bit ID that must match, 1 = must match, 0 = don't care. Default set to MATCH_ALL = 0x07FF … vs RECEIVE_ALL = 0x0000
`Service`	HALT_ON_RECEPTION = the Mailbox will stop receiving any more data until the buffer is read and Cleared by command. The Receive flag is set and counter = 1 OVERWRITE_ON_RECEPTION = the mailbox will contain the last data, old data is lost. The Receive flag is set and counter will increment on each new reception

RETURNS:

`0`	Mailbox not setup
`1`	Mailbox setup

REPORTED ERRORS:

`Bad_ID`	Handle out of Range
`ERROR_SETUP_MB`	Set up the mailbox fail. The Node ID/Mask has been configured in another mailbox.

FUNCTION

`Define_CANopen_Receive_Data()` *Used to define the data received in a mailbox is arranged*

This function is used to define how the data received in a mailbox is arranged.

Up to 64 bits can be loaded into a single message.

Multiple define_receive_data calls are used to add retrieve data from the message, up to the 64 bit limit.

No more than 16 Define Data functions per mailbox.

The data is ordered in the same order as the functions are called starting at bit 1 of Byte 1.

The Mailbox must be disabled (automatically done by the any setup transmit mailbox function)

SYNTAX

`Define_CANopen_Receive_Data(Handle,Data_Source,Start_Bit,Num_Of_Bits)`

ARGUMENTS:

`Handle`	Holds the ID of the mailbox
`Data_Destination`	Address of Data where the bits will be stored, such as P_USER_11
`Start_Bit`	The first, lowest order bit of the data_source to be packed into (0 = bit1 and 31 = bit32)
`Num_OF_Bits`	How many bits are packed by this function into the message data. 1 – 64

RETURNS:

`0`	Data format failed... too many bits or not enough bits in the data source
`1`	Data format set

REPORTED ERRORS:

`Bad_ID`	Handle out of Range
`PT_RANGE`	Parameter index out of range or try to match to an OS parameter with unmatched data bits.
`PARAM_RANGE`	Invalid start bits or length (exceed the parameter length) or the mailbox has been defined too may times (over 16)
`UNIT_BUSY`	The mailbox has been enabled
`BAD_MO_LEN`	The data bits to define exceed the number of undefined bits of the mailbox
`PARAM_RO`	Variable is read-only or not accessible

FUNCTION `Setup_Receive_Mailbox_Auto_Reply()` *Mailbox to be sent upon receipt of data*

Configure a transmit mailbox to be sent upon receipt of data in this RX mailbox.

The RX and TX mailboxes should be fully defined and configured before using this function.

If the TX mailbox is not enabled, it cannot be sent.

Note, basically, any transmit mailbox can be used as the auto replay. The number one usage is to send a RPDO when a TPDO is received (reminder: TX and RX are from the perspective of the ancillary-device, as per CANopen specifications)

SYNTAX `Setup_Receive_Mailbox_Auto_Reply(Handle,Reply_Mailbox)`

ARGUMENTS:

`Handle`	the ID of the receiving (RX) mailbox that triggers the auto-reply (i.e., 1351).
`Reply_Mailbox`	Holds the Handle ID of the transmitting (TX) mailbox which will be send to the ancillary controller/device.

RETURNS:

`0`	Mailbox not setup
`1`	Mailbox setup

REPORTED ERRORS:

`Bad_ID`	Handle out of Range

FUNCTION `Setup_Receive_Mailbox_Timeout()` *Setup a timeout fault, trigger*

Setup a timeout fault that is active if this mailbox does not receive new data within the Timeout period

Usage example: set a fault if an ancillary controller/device fails to send data to the 1351 manger controller. Critical data that is not received within a given time period triggers a fault.

SYNTAX `Setup_Receive_Mailbox_Timeout(Handle,First_Timeout,Cycle_Timeout)`

ARGUMENTS:

`Handle`	Holds the ID of the mailbox
`First_Timeout`	0 = NO_TIMEOUT Non-zero is the maximum time (in milliseconds) allowed for the first message to be received after the mailbox is enabled. After this first timeout is triggered, a timeout flag is set in the OS. This flag can be viewed by the VCL Function, *Check_RX_Mailbox_Status()* If the mailbox has timed out, it will no longer receive data.
`Cycle_Timeout`	0 = N0_TIMEOUT Non-zero is the maximum time (in milliseconds) allowed between receptions After this timeout is triggered, a timeout flag is set in the OS. This flag can be viewed by the VCL Function, *Check_RX_Mailbox_Status()* If the mailbox has timed out, it will no longer receive data.

RETURNS:

`0`	Mailbox not setup
`1`	Mailbox setup

REPORTED ERRORS:

`Bad_ID`	Handle out of Range

Controlling a Receive Mailbox

The receive mailbox processing is a bit more complex than the transmit mail box, although there are many similarities, such as the mailbox must be enabled for any actions to occur and must be disabled before any setup functions.

Since there are two modes of operation, halt and over-write, there are multiple way to check and pull data from the receive mailbox.

If the halt-on-reception method is used, the VCL program should check the status of the receive flag. If it is set, the data is new and can be used. The flag must be cleared before any new data is received.

If the over-write method is used, the VCL program may check the counter to see how many messages have been received. This is an incrementing counter than can be cleared. If the VCL program code does this after each check of the mailbox, then this counter will say how many message have been missed (>1). The receive buffer is double buffered such that the incoming data is fully loaded before being set as the active receive mailbox.

The receive mailboxes can also be set to time-out if they do not get new data. This timeout works on the incoming data, not the data buffer, thus, even the mailboxes that are set to halt-on-reception are checked for new messages coming in at or below the timeout rate. If the timeout is exceeded, the flag is set and the message buffer will not be updated further until the timeout flag is cleared. The data in the mailbox is the last data received before the timeout. It is important to note that the message counter will continue to count if new messages matching the mailbox start coming in again, but they will not be received into the mailbox until the timeout flag is cleared. But, the counter could be used to determine that messaging has started again.

Configuration of the non-CANopen Mailbox functions

The following VCL Functions, including 29-bit identifier CAN messages do not change the total quality of the available mailboxes. There remains 30 Transmit and 30 Receive mailbox on CAN1 (CAN_PORT_1) and 10 each of Transmit and Receive on CAN2 (CAN_PORT_2). Reference the *System Information* file for further details on the available VCL CAN functions (available in the CIT VCL Studio app, within the "help" tap pull-down).

Assign_CAN_Mailbox( )

Setup_CAN_Receive_Mailbox( ) *Setup_CAN_Transmit_Mailbox( )*

Setup_CAN_Receive_Data( ) *Setup_CAN_Transmit_Data( )*

Enable_Receive_Mailbox( ) *Enable_Transmit_Mailbox( )*

FUNCTION `Assign_CAN_Mailbox()`

Use this function for setting up both the Receive and Transmit handles. The handle is the VCL program's reference for the mailbox. This function assigns the VCL CAN Receive or Transmit Mailbox, storing the mailbox handle in a VCL variable. This way, the correct mailbox can be referenced when checking the receive status or other mailbox properties. The CAN mailbox handles are numbered separately for the receive and transmit mailboxes; the type of mailbox always needs to be specified when referencing a mailbox handle.

SYNTAX `Assign_CAN_Mailbox(Port, Function)`

ARGUMENTS:

`Port`	The desired hardware port of the 1351 controller (CAN_PORT_1 or CAN_PORT_2)
`Function`	C_XMT for transmit operations. C_RCV for receive operations.

Example:

```
Switch_1_Status= assign_can_mailbox(CAN_PORT_1, C_RCV)
          ;CAN1, Receive Mailbox
Switch_2_Status= assign_can_mailbox(CAN_PORT_2, C_XMT)
          ;CAN2, Transmit Mailbox
```

FUNCTION `Setup_CAN_Receive_Mailbox()`

Use this function for setting up the initial parameters for a receive mailbox. This function MUST BE called before defining the data for the mailbox (see Setup_CAN_Receive_Data) or trying to use the mailbox.

SYNTAX

```
Setup_CAN_Receive_Mailbox(Handle, ID_High, ID_
Low, Mask_High, Mask_Low, Extended, Reply,
Startup timeout, Cyclic Timeout, Handshake)
```

ARGUMENTS:

`Handle`	Holds the ID of the mailbox (returned by Assign_CAN_Mailbox)
`ID_High`	High 13 bits of ID
`ID_Low`	Low 16 bits of ID
`Mask_High`	High 13 bits of mask ID
`Mask_Low`	Low 16 bits of mask ID
`Extended`	Normal Id (11bit) = 0, Extended Id (29bit) = 1

`Reply`	Handle of reply mailbox
`Startup timeout`	Maximum duration in which first message must be received in 4ms increments (e.g., 5 = 20ms timeout)
`Cyclic Timeout`	Maximum duration between cyclic messages in 4ms increments (e.g., 5 = 20ms timeout)
`Handshake`	Handshake enabled

Example:

```
Setup_CAN_Receive_Mailbox(Switch_1_Status, 0, 0xFDDB,
                          0x7FF,0x7FF,1,0,200,200,0)
   ; 1 => 29-bit identifier (the 6th argument)
```

FUNCTION `Setup_CAN_Receive_Data()`

Maps variables for a receive mailbox. Define the pointers used with a Message Buffer. This function is used to define the 'data pointers' for a given receive mailbox. Notes,

1. The *Setup_CAN_Receive_Mailbox* function MUST HAVE BEEN Called before defining the data with this function.
2. The data is ordered in the same order as the functions are called.
3. All functions must be called in sequence of transmission.
4. Calling the function after a mailbox has been enabled results in the data pointers being reset

SYNTAX `Setup_CAN_Receive_Data(Handle, Data, Length, Endian)`

ARGUMENTS:

`Handle`	Holds the ID of the mailbox (returned by Assign_CAN_Mailbox)
`Data`	Address of Data
`Length`	Data length, in **Bytes (from 1 to 4 bytes of data)** (i.e., allows any combination of messages up to four bytes) For example: Four 1-byte messages, one 4-byte message, two 2-byte messages, etc.
`Endian`	Endianness of data, 0 = Little Endian, 1 = Big Endian

Example:

```
Setup_CAN_Receive_Data(Switch_1_Status, DIRECTION_SWITCH_1, 2, 0)
                            ; DIRECTION_SWITCH_1 is 2 Bytes of data.
Enable_Receive_Mailbox(Switch_1_Status)     ; Enable this Receive mailbox
```

Transmit

The CAN Transmit functions are similar, including the argument "Length" which is the number of bytes for the Setup_CAN_Transmit_Data (Handle, Data, **Length,** Endian) function.

SRDOs

CANopen standard DS304 defines the method to produce Safety Related Date Objects. When enabled, the Primary will create the first message and the Supervisor will follow with the data inverted second message. If the CAN ports are linked, these messages will go out the same ports and thus conform to the CANopen standard. If the ports are not linked, each message will go out its own port, but in the same format and timing.

For SRDO transmission, a mailbox in the Primary must be set up and set to transmit on command and a mailbox must be setup up to match in the supervisor. The 11 bit identifier must be set according to the DS304 definition, which also means the *Node_IDs* must be set to the proper standard as well.

SRDOs may also be received. To do this, a mailbox in the Primary (port1) must be setup to receive one message and mailbox to receive the inverted data message in the Supervisor (port 2).

The following VCL functions can then be used to send SRDOs and validate incoming SRDOs

FUNCTION `SRDO_Handle = Setup_Transmit_SRDO()` *Links two mailboxes to be transmitted in a linked fashion.*

SYNTAX `Setup_Transmit_SRDO (Mailbox_Handle_1, Mailbox_Handle_2, Rate, Interval)`

ARGUMENTS:

`Mailbox_Handle_1`	The Primary mailbox *(Primary_Handler)*
`Mailbox_Handle_2`	The Supervisory mailbox *(Supervisor_Handler)*
`Rate`	If set to EVENT (=0) then this mailbox will send on the send command otherwise is it the rate between sending each primary mailbox in milliseconds
`Interval`	The time, in milliseconds, between sending the Primary and Supervisor messages.

RETURNS:

`0`	function did not successfully execute
`handle`	The handle to the setup SRDO function

REPORTED ERRORS:

`Bad_ID`	The primary mailbox or supervisor mailbox is not set up as SRDO type correctly.

FUNCTION `Enable_Transmit_SRDO()` *Starts the SRDO*

`Enable_Receive_SRDO()` *Starts the SRDO*

These two functions start the SRDO. If set to a cyclic rate, this will start transmissions

SYNTAX `Enable_Transmit_SRDO(SRDO_Handle)`

`Enable_Receive_SRDO(SRDO_Handle)`

ARGUMENTS:

`SRDO_Handle`	See RETURN: SRDO_*Handle* = *Setup_Transmit_SRDO()*

RETURNS:

`0`	Invalid SRDO handler
`1`	Function successfully executed

REPORTED ERRORS:

`Bad_ID`	Invalid SRDO handle

FUNCTION `Disable_Transmit_SRDO()` *Stops the SRDO*

`Disable_Receive_SRDO()` *Stops the SRDO*

These two functions stop the SRDO. If set to a cyclic rate, this will stop transmissions

SYNTAX `Disable_Transmit_SRDO(SRDO_Handle)`

`Disable_Receive_SRDO(SRDO_Handle)`

ARGUMENTS:

`SRDO_Handle`	See RETURN: SRDO_*Handle* = *Setup_Transmit_SRDO()*

RETURNS:

`0`	Invalid SRDO handler
`1`	Function successfully executed

REPORTED ERRORS:

`Bad_ID`	Invalid SRDO handle

FUNCTION `Send_SRDO()` *Initiates TX of the two SRDO messages*

Initiates TX of the two messages associated with this SRDO with an interspace timing.

SYNTAX `Send_SRDO(SRDO_Handle)`

ARGUMENTS:

`SRDO_Handle`	See RETURN: SRDO_*Handle* = *Setup_Transmit_SRDO()*

RETURNS:

`0`	Invalid SRDO handler
`1`	Function successfully executed

REPORTED ERRORS:

`Bad_ID`	Invalid SRDO handle

Example Transmit SRDO Setup

The example below assumes the variables have been setup previously for the mailbox handles, Node IDs and message data. After this set up, the main loop of VCL must call the *`Send_SRDO(SRDO_XMT_Handle)`* command to cause the 1351 to send the two linked SRDO messages.

Note that only the Primary port mailbox Data is defined. In an SRDO transmit, the Supervisor will use the same data definition as the Primary, but will invert the data when sent from the Supervisor.

```
; ==================== SETUP SRDO Transmit =============

; Set the CAN cross connect parameter ON by CIT or the 1313MH
; Setup two mailbox, one is for primary (first message), another is for supervisor (inverted data/second message)
        Primary_Mailbox  = assign_can_mailbox(CAN_PORT_1, C_XMT);
        Supervisor_Mailbox = assign_can_mailbox(CAN_PORT_2, C_XMT);

; Configure the two mailboxes as the SRDO type
        setup_canopen_transmit_mailbox(Primary_Mailbox, SRDO, SRDO_FIRST_MESSAGE_ID, 0)
        setup_canopen_transmit_mailbox(Supervisor_Mailbox, SRDO, SRDO_SECOND_MESSAGE_ID, 0)

; Configure mailbox data
        define_canopen_transmit_data(Primary_Mailbox, SRDO_XMT_Data_1, 0, 8)
        define_canopen_transmit_data(Primary_Mailbox, SRDO_XMT_Data_2, 0, 8)
        define_canopen_transmit_data(Primary_Mailbox, SRDO_XMT_Data_3, 0, 8)
        define_canopen_transmit_data(Primary_Mailbox, SRDO_XMT_Data_4, 0, 8)

; Setup SRDO to event driven with a 20mS interval between first and second messages
        SRDO_XMT_Handle = Setup_transmit_SRDO(Primary_Mailbox, Supervisor_Mailbox, 0, 20)

; Enable SRDO
; Enable_transmit_mailbox(Primary_Mailbox)
; Enable_transmit_mailbox(Supervisor_Mailbox)
; Enable_Transmit_SRDO(SRDO_XMT_Handle)

; ======================================================
```

FUNCTION `SRDO_Rcv_Handle = Setup_Receive_SRDO()` *Sets up the timing detection and fault functions for a receive SRDO*

The receive mailboxes must already be setup prior to using this function. Since the SRDO consists of two messages and is received and checked by both CAN ports, there are four mailboxes used total. Note that Only the Primary First mailbox must be defined since the Second message is identical with the data inverted.

SYNTAX `Setup_Receive_SRDO(Primary_Mailbox_Handle_first, primary_Mailbox_Handle_second, Supervisor_mailbox_handle_first, supervisor_mailbox_handle_second, Cycle_Timeout, Interval_Timeout)`

ARGUMENTS:

`Primary_Mailbox_Handle_first`	The Primary receive mailbox handle1 with odd node ID.
`Primary_Mailbox_Handle_second`	The Primary receive mailbox handler2 with even node ID.
`Supervisor_Mailbox_Handle_first`	The Supervisor receive mailbox handler1 with odd node ID.
`Supervisor_Mailbox_Handle_second`	The Supervisor receive mailbox handler2 with even node ID.
`Cycle_Timeout`	The max time allowed between Primary Mailbox first messages receives
`Interval_Timeout`	The max time, in milliseconds, allowed between reception of first and second mailbox receives

RETURNS:

`0`	function did not successfully execute
`handle`	The Handle to the SRDO reception function. It is used to enable and to check the status of the SRDO receive function

REPORTED ERRORS:

`Bad_ID`	The primary mailbox or supervisor mailbox is not set up as SRDO type correctly.

FUNCTION `Status = check_SRDO_receive()` *Check for proper timing and data integrity of the SRDO reception*

SYNTAX `check_SRDO_receive(SRDO_Rcv_Handle)`

SRDO_Rcv_Handle is the handle provided by the SRDO Receive Setup function (above). It is used to enable the SRDO function (after the mailboxes have all be setup) and to check the SRDO timing and data integrity.

If it returns a non-zero value into Status, the SRDO timing or address is wrong. The data will not be placed into the defined variables for the RSDO data.

Example Receive SRDO Setup

The example below assumes the variables have been setup previously for the mailbox handles, Node IDs and message data. After this set up, the main loop of VCL must check for the valid operation of the SRDO reception using the VCL function `check_SRDO_receive(SRDO_RCV_Handle)`. If it returns a non-zero value, there is a SRDO fault and the data received nay not be valid.

Note that only the first message Primary port Receive mailbox Data is defined. In an SRDO Receive, the Supervisor and Primary will use the same data definition for both messages, but will check for the inverted the data and timing automatically

```
; ==================== SETUP SRDO Receive =============

; Assign four mailboxes, two for primary and two for supervisor:
        SRDO_Primary_First_Mailbox          = assign_can_mailbox(0, C_RCV)
        SRDO_Primary_Second_Mailbox         = assign_can_mailbox(0, C_RCV)
        SRDO_Supervisor_First_Mailbox       = assign_can_mailbox(1, C_RCV)
        SRDO_Supervisor_Second_mailbox      = assign_can_mailbox(1, C_RCV)

; Configure these four mailboxes as SRDO type
        setup_canopen_receive_mailbox(SRDO_Primary_First_Mailbox, SRDO, SRDO_RCV_ODD_ID, 0x7FF, 0)
        setup_canopen_receive_mailbox(SRDO_Primary_Second_Mailbox, SRDO, SRDO_RCV_EVEN_ID, 0x7FF, 0)
        setup_canopen_receive_mailbox(SRDO_Supervisor_First_Mailbox, SRDO, SRDO_RCV_ODD_ID, 0x7FF, 0)
        setup_canopen_receive_mailbox(SRDO_Supervisor_Second_mailbox, SRDO, SRDO_RCV_EVEN_ID, 0x7FF, 0)

; Define the received data for the mailboxes
; The dual message SRDO data is compared in the firmware,
; Only valid data will be received into variables defined below:
        define_canopen_receive_data(SRDO_Primary_First_Mailbox, CAN_1_RPDO1_byte_1, 0, 8)
        define_canopen_receive_data(SRDO_Primary_First_Mailbox, CAN_1_RPDO1_byte_2, 0, 8)
        define_canopen_receive_data(SRDO_Primary_First_Mailbox, CAN_1_RPDO1_byte_3, 0, 8)
        define_canopen_receive_data(SRDO_Primary_First_Mailbox, CAN_1_RPDO1_byte_4, 0, 8)

; Setup this SRDO: 800 ms between message packages, 300ms between first and second (inverted data) messages
        SRDO_RCV_Handle = setup_receive_SRDO(SRDO_Primary_First_Mailbox,
        SRDO_Primary_Second_Mailbox, SRDO_Supervisor_First_Mailbox,
        SRDO_Supervisor_Second_mailbox, 800, 300)

; Enable SRDO
        Enable_Receive_SRDO(SRDO_RCV_Handle)
```

6 — INITIAL SETUP & COMMISSIONING

INITIAL SETUP

The 1351 System Controller can be used in a variety of vehicles which differ widely in characteristics. Consequently, not all parameters or parameter modes/types may be appropriate to a given application, or the default parameter values. Therefore, before operating the vehicle, it is imperative that these initial setup steps be carefully considered and followed to ensure that the system controller's parameter settings are suited to the application. To gain a better insight into the parameters and their settings, completely read this chapter before conducting the individual commissioning steps.

BEFORE YOU START

⚠ WARNING

The 1351 is not a motor controller. It is often used as the manger controller on a vehicle where there are motor controllers and gauges. Communication between devices is via the CANbus. Before beginning vehicle commissioning (command & control), jack the vehicle drive wheels up off the ground so that they spin freely and the vehicle is stable—especially when the drive wheels will accelerate, decelerate, or spin at high speeds during commissioning. Reference the application's wiring diagram when assigning the controller's inputs, outputs and control functions, as well as those of the motor controllers. Double-check all wiring to ensure it is consistent with the wiring guidelines described in *all* the controller's manuals. Finally, ensure all electrical and mechanical connections are properly torqued before proceeding with these setup and commissioning steps.

TO BEGIN

Use the example-wiring diagram (Figure 4, page 13) to complete a similar diagram for the application, including an overall wiring diagram for the vehicle's other controllers. Document how the 1351 will interface to the vehicle's hydraulics, sensors, electric motor controllers (traction, pump, steer, etc.), any ICE (Internal Combustion Engine), transmissions, or generators, as well as the vehicle/operator controls and gauges.

When everything is installed, turn on the 1351 system controller by closing the keyswitch circuit. The 1351 will power-up and blink the status LEDs (see Table 22). Using a PC, connect the Curtis Integrated Toolkit ™ (CIT) to the CANbus (i.e., establish a CANbus connection). Within the toolkit's Launchpad window, either create a new project, or scan for devices and then highlight the 1351 system controller that is being-setup, then create or modify the application's CIT Project. Ensure the Controller and Project have matching device profiles. When the application's project is established*, then click-to-open the Programmer application (app) tool. Programmer is where the parameters are configured (i.e., programmed). These steps can also be accomplished using the Curtis 1313 handheld CANbus programmer (1313 HHP*). The 1313 HHP does not have the VCL Studio app, which enables changes to the VCL program.

Note: It is recommended NOT to assert the motor controllers' interlock(s) during the initial 1351 parameter setup. This will prevent motor operation, which can come later when the 1351 is setup and then the focus can be on the motor controller(s)'s operation and tuning (using the motor controller-commissioning guide described with *their* manuals).

** See Appendix C for the Curtis tools to configure the 1351 System Controller.*

Contact your Curtis distributor or support engineer for help or training with the setup and using the Curtis Integrated Toolkit™.

PARAMETER SETTINGS – METHOD OVERVIEW

It is the user's preference what parameters to set first. It is recommended to set the hardware related parameters first, starting with the CAN communication parameters. Finish by setting the software-based parameters. For example,

CAN menu parameters

System Controller menu parameters

Inputs menu parameters

Outputs menu parameters

Accelerometer parameters

Step 1: CAN parameters

Determine if the system will use either or both CAN ports. Do not assign the same CAN Node ID to the two ports. Refer to the CAN menu (p. 71). The Node ID setup/assignment is via the Curtis Integrated Toolkit™ : Launchpad, *add a device.*

Step 2: System Controller parameters

Power menu. Set the system battery voltage, the minimum, maximum, and brownout trip levels to match the application (p. 35). Do not set the Brown Out Voltage below the 1351's default voltage.

BDI menu. If using Lead-acid batteries, setup the Battery Discharge Indicator (BDI) parameters. For optimum accuracy of the battery state of charge (SOC) calculations, use the battery type's charged and discharged specifications for setting the full and empty volts/cell parameters. Use the battery pack's kWh ratings and the vehicle's duty cycle to pre-calculate the discharge time parameter. Barring this information, experimenting with the operational vehicle in its normal usage is suggested to then finalize the BDI parameters. Refer to the BDI parameters and BDI explanation in Chapter 3 (p. 35).

Note, if not using lead-acid batteries, leaving the BDI parameters at their default settings is recommended. Simply ignore the BDI output monitor variable, *BDI Reading* in the VCL program.

For non-lead-acid batteries, including Lithium-Ion battery packs, use the battery pack or cell manufacturer's approved Battery Management System (BMS) for determining the SOC. The BMS aspect will almost always be via the CANbus.

External Supplies menu. Based upon the application, the 5 and 12-volt supplies can be enabled or left turned off. To help setup these parameters when powering external devices, this menu includes the monitor variables for voltage and current (p. 37). Note, the combined load current between the +5V and +12V outputs cannot exceed 300 mA (i.e., the *external power supplies* are not designed to power high wattage loads, only devices such as encoders, potentiometers, and RTD).

Temperature menu. If the application requires a minimum or maximum 1351 module temperature limit that is different from the – 20°C to 100°C default limits, make the adjustments to these parameters (p. 37).

Step 3: Input parameters

The Inputs menu contains seven sub menus, each of which pertains specifically to the user's application. If a particular input does not apply or will not be used, either leave the settings at their default value, or disable them (i.e., select "none" based upon the parameter). The active settings and pin-connection of common I/O always takes persistence. Use the application's wiring diagram (*created as recommended*) to setup the switch inputs, and whether or not any of the analog inputs will utilize the virtual switch feature. When using the virtual switch feature, be sure to set the corresponding analog input's high

and low threshold settings. Immediate feedback of the virtual switch status (monitor variable *VSW Status*) is included in each of the analog input sub-menus. It is recommended each input be verified as part of the initial setup.

Pot Inputs require setting the Type parameter. Do this first, and then set the resistance and tolerance parameters. Adjust these parameters based upon the in-menu Resistance and Wiper Position monitor variables values. Figures 5 and 6 illustrate the wiring for 3-Wire and 2-Wire potentiometers.

Figure 5
Wiring for 3-Wire Potentiometer

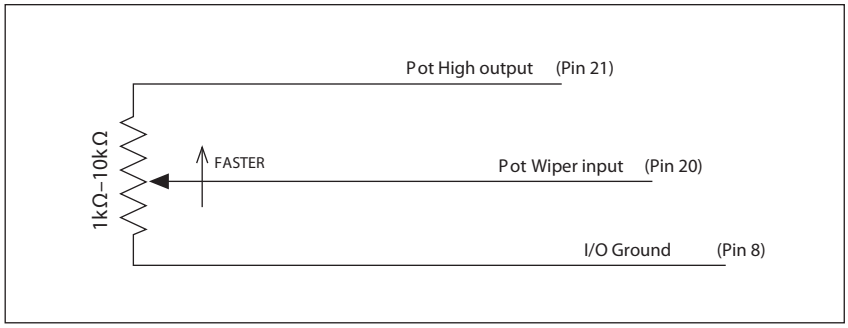
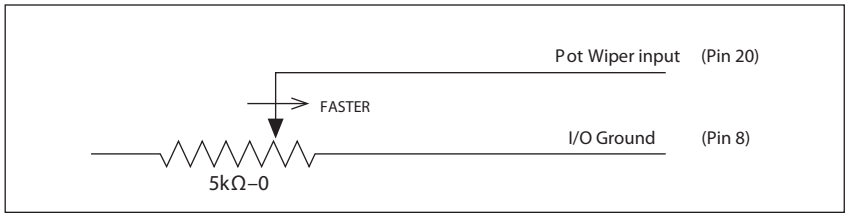


Figure 6
Wiring for 2-Wire Potentiometer



RTD Inputs require their type be enabled and their input-resistance versus output-value mapped. Once the system is operational, check the RTD's corresponding monitor variables Resistance and Value, making adjustments to the map values to achieve the correct response.

The two High Speed Digital Inputs parameters have a Type and Direction setting. Verify the RPM output variable is correct once the system is operational. Often, hand spinning of an encoder's device (e.g., motor) can be used to finalize the parameter settings.

Refer to these pages for the Input sub-menus.

Switches:	p. 39 – 42
Virtual Switches:	p. 43 – 44
Analog Inputs:	p. 45 – 46
Pot Inputs:	p. 47
RTD Inputs:	p. 48 – 49
High Speed Digital Inputs:	p. 50
Encoder Inputs:	p. 51 – 53

Step 4: Output parameters

The Outputs menu contains five sub menus, each of which pertains specifically to the user's application. If a particular output does not apply or will not be used, either leave the settings at their default value, or disable them (i.e., select to "Off" based upon the parameter). The active settings and pin-connection of common I/O always takes persistence. Use the application's wiring diagram (*created as recommended*) to setup the driver outputs, and whether or not the safety output feature will be utilized. Note: the "drivers" are low-side drivers; they must be connected with the Coil Return (B+) on the high-side of a coil or similar load. For inductive loads (coils), the connection at the Coil Return provides a common-to-all fly-back diode to B+ internal to the 1351. Driving capacitive loads, such as gauge or LED indicators, may cause non-intended results, and often an inline resistive element may be required to increase the RC time-constant on the powered device (i.e., the load) to obtain acceptable results.

The PWM Driver outputs have a "Test" mode "Command" parameter which resets to OFF following a key/power cycle (also see Safety Output Command). Use this during setup to verify the driver's operation when evoked in the VCL program or via a CANbus command. Test all drivers to ensure the load remains within the driver's current limits.

The Half-Bridge Drivers also have the "Test" mode parameter. Test the operation during setup to remain within the current limits of the drivers, before enabling the driver using the VCL command "Driver_X_Command = "the percentage" as per the driver.

The digital Driver's default setting is Off. Enable the output by setting the individual *Digital_Out_X_Mode* parameters from Off to Enable (in Programmer). The output can be 'tested' using the Programmer's parameter Digital Out X Command, which re-cycles to Off (0) following a KSI/power cycle. To enable the digital drivers output during operation, include in the VCL program the individual *Digital_Out_x_Command* = 1. X = 1, 2, or 3 for the particular driver.

To use the "Drivers" the Safety Output (Safety Output/Coil Return, pins 11 & 12) must be enabled. The Coil Return/Safety Output is the B+ voltage supply to the low-side drivers. The Safety Out Command in the CIT/1313 Programmer app's menu is valid for testing, returning to 0 (off) following a KSI/power cycle. In the VCL program, use the Safety_Out_Command = 1 to enable the output across KSI/Power cycles.

The Analog Output also has a test command parameter.

Refer to these pages for the Output sub-menus.

PWM Drivers:	p. 55 – 63
Half-Bridge Drivers:	p. 64 – 67
Digital Drivers:	p. 68
Safety Output:	p. 69
Analog Output:	p. 70

Step 5: Clear Faults

After configuring the parameter settings, cycle the keyswitch, and then use the Programmer app to check for faults. Clear and resolve all faults, including those in the Fault History (Programmer » Faults » History » Clear History) before continuing.

Use Chapter 7 for help in troubleshooting. Contact your Curtis customer support engineer to resolve any fault issues.

7 — DIAGNOSTIC AND TROUBLESHOOTING

The controller has two status LEDs, one red and one yellow LED. The present status of the controller can be ascertainable within seconds by observing the status LEDs. Each fault will flash a specific code.

THE DIAGNOSTICS PROCESS

Diagnostics information can be obtained in either of three ways: (1) by observing the fault codes flashed by the controller's status indicator, (2) by reading the indicated fault (🔴) in the *Curtis Integrated Toolkit*™ Programmer app, or (3) the CAN Emergency Messages.

Table 22 specifies the condition and associated flash pattern for the 1351 System Controller LEDs.

Table 22 LED flash patterns

Condition	Red	Yellow	Pattern
OK	Off	Flashing at a 1.5 sec cycle. ON = 500 ms Off = 1000 ms	
Programming	ON	ON	
Dead	ON	OFF	
Error	Flash Code digit 1 250 ms ON 250 ms OFF	Flash Code digit 2 250 ms ON 250 ms OFF	Each code is followed by a 2-second pulse, and the next code is started.
Invalid Software	Fast Flash	OFF	

The Normal/OK state is shown by only on the yellow LED flashing a 1 at a 1.5 sec cycle rate. Additionally, the On-time is lengthen to a full 500 ms, thus further differentiates it from any Error flash code.

Programming is shown using two formats:

- Initiate Programming and End of Programming = Both LEDs ON 100%,
- While data is incoming = Red LED ON and Yellow LED fast flashing

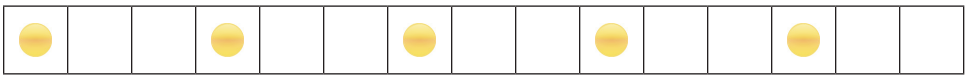
Non-operation (dead) state is shown by the RED Led 100% on and the Yellow LED off.

Error Codes (faults) are shown on the Status LEDs using a sequence of flashes and pauses. The technique used is a departure from the previous Curtis motor controller standard (e.g., E/SE controllers). The 1351 uses a format where the Red LED flashes the upper Tens digit number and the Yellow LED flashes the lower Ones digit number. The flash cycle is 500 ms with a 250 ms On time. If there are multiple codes, they are flashed in succession with a 2-second pause between each. There is an additional 2 seconds added to the last flash code in the list (4-second total pause) to denote the last Error code has be flashed.

Examples: (Each block = 500 ms)

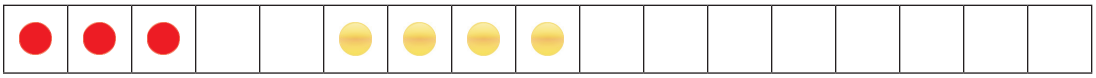
Normal/Ok/No errors

- On/Off cycle within each block = 100% (thus, the On time is 500 ms)

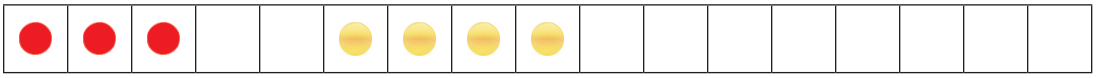


Error Flash Code

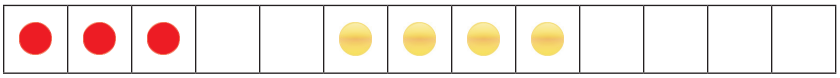
- On/Off cycle within each block = 50% (this the on time is 250 ms)
- Single code = 34 (note long pause at the end, showing the end of the fault code list)



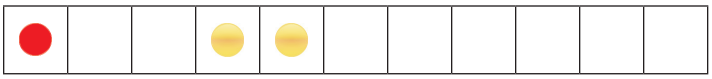
Repeats….



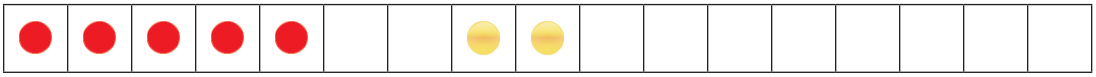
Multiple codes = 34, 12 and 52



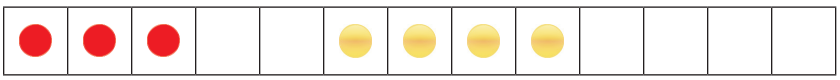
Followed by….



Followed by…. With the long pause at the end to signal the end of the fault code list



Repeats….



Each Fault Record has the following features defined. (Table 23)

Table 23 Fault Record (sub-Index & bytes)

Fault Record		
Sub-index	**Size**	**Description**
01h	1 byte	Status
02h	1 byte	Associated flash code. See Table 24
03h	3 bytes	Fault Detection Count
04h	2 bytes	Time of Last Detection
05h	4 bytes	Time of First Detection
06h	4 bytes	Fault Type

Table 24 lists all the 1351 System Controller Faults. Use these to diagnose system faults.

- The Flash Code sequence (Hex, *and decimal equivalents for values A-F*)
- The Fault Name and Type
- The Set and Clear triggers
- The Fault action

Table 24 Fault Table: The Set/Clear Conditions & Fault Actions

Flash Code	Fault Name	Set/Clear Conditions	Fault Actions
0x14	Under Voltage Fault Type(s): (1)	**Set:** KSI voltage dropped below the **Min Voltage** limit. **Clear:** Bring KSI voltage above the **Min Voltage** limit	Shutdown all driver outputs/ digital outputs.
0x15	Over Voltage Fault Type(s): (1)	**Set:** KSI voltage exceeded the **Max Voltage** limit. **Clear:** Bring KSI voltage below the **Max Voltage** limit	Shutdown all driver outputs/ digital outputs.
0x16	Under Temp Fault Type(s): (1)	**Set:** Controller's Heatsink temperature dropped below **Min Temp** limit. **Clear:** Bring heatsink temperature above **Min Temp** limit.	Shutdown all driver outputs/ digital outputs.
0x17	Over Temp Fault Type(s): (1)	**Set:** Controller's Heatsink temperature exceeded **Max Temp** limit. **Clear:** Bring heatsink temperature below **Max Temp** limit.	Shutdown all driver outputs/ digital outputs.
0x21	Driver 1 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 1 is either open or shorted. Or Driver 1 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 1.
0x22	Driver 2 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 2 is either open or shorted. Or Driver 2 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 2.
0x23	Driver 3 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 3 is either open or shorted. Or Driver 3 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 3.
0x24	Driver 4 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 4 is either open or shorted. Or Driver 4 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 4.
0x25	Driver 5 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 5 is either open or shorted. Or Driver 5 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 5.
0x26	Driver 6 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 6 is either open or shorted. Or Driver 6 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 6.
0x27	Driver 7 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 7 is either open or shorted. Or Driver 7 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 7
0x28	Driver 8 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 8 is either open or shorted. Or Driver 8 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 8.

Table 24 Fault Table: The Set/Clear Conditions & Fault Actions, cont'd

Flash Code	Fault Name	Set/Clear Conditions	Fault Actions
0x29	Driver 9 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 9 is either open or shorted. Or Driver 9 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 9.
0x2A (2 x 10)	Driver 10 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 10 is either open or shorted. Or Driver 10 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 10.
0x2B (2 x 11)	Driver 11 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 11 is either open or shorted. Or Driver 11 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 11.
0x2C (2 x 12)	Driver 12 Fault Fault Type(s): 1 = Driver Open 2 = Driver Short 3 = Driver Overcurrent	**Set:** Driver 12 is either open or shorted. Or Driver 12 current exceeded 3.5A **Clear:** Correct open or short, and then Reset Controller.	Shutdown driver 12.
0x31	Digital Out 1 Fault Fault Type(s): (1)	**Set:** Digital Out1 is current exceed 4.1A. **Clear:** Correct open or short, and then Reset Controller.	Shutdown digital output1.
0x32	Digital Out 2 Fault Fault Type(s): (1)	**Set:** Digital Out2 is current exceed 4.1A. **Clear:** Correct open or short, and then Reset Controller.	Shutdown digital output2.
0x33	Digital Out 3 Fault Fault Type(s): (1)	**Set:** Digital Out3 is current exceed 4.1A. **Clear:** Correct open or short, and then Reset Controller.	Shutdown digital output3.
0x34	Safety Output Fault Fault Type(s): (1)	**Set:** Safety output current exceeded max limit. **Clear:** Reset the controller	Shutdown safety output
0x35	NV Failure Fault Type(s): 1 = Invalid checksum. 2 = NV write failed. 3 = NV read failed. 4 = NV write did not complete during power down.	**Set:** Controller operating system tried to read or write to EEPROM memory and failed. **Clear:** Download the correct software and matching parameter default settings into the controller and Reset Controller.	Shutdown all driver outputs/ digital outputs.
0x38	Parameter Out Of Range Fault Type(s): Reports the CAN Object ID of parameter.	**Set:** Parameter detected outside of limits **Clear:** Bring parameter within its limits.	Shutdown all driver outputs/ digital outputs.
0x3B (3 x 11)	External Current Out Of Range Fault Type(s): 1	**Set:** The total current of external 5V and 12V exceeded 340mA **Clear:** Reset the Controller	Shutdown external 5V and 12V.
0x3E (3 x 14)	PDO Mapping Error Fault Type(s): 1	**Set:** Incorrect PDO map detected. **Clear:** Reset Controller.	PDO message disabled
0x3F (3 x 15)	PDO Timeout Fault Type(s): 1	**Set:** Time between CAN PDO messages received exceeded the PDO Timeout Period. **Clear:** Receive CAN NMT message, or Reset Controller.	NO_ACTION in software.

Table 24 Fault Table: The Set/Clear Conditions & Fault Actions, cont'd

Flash Code	Fault Name	Set/Clear Conditions	Fault Actions
0x41	VCL Run Time Error Fault Type(s): 1	**Set:** VCL Run Time Error detected **Clear:** Edit VCL application software to fix this error condition; flash the new compiled software and matching parameter settings; Reset Controller.	Shutdown all driver output and digital output
0x42	User 1 Fault Fault Type(s): OEM Definable	These user faults can be set/clear by the User/OEM in the application-specific VCL software	These fault actions can be defined by the User/OEM and are implemented in the application-specific VCL software
0x43	User 2 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x44	User 3 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x45	User 4 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x46	User 5 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x47	User 6 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x48	User 7 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x49	User 8 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x4A **(4 x 10)**	User 9 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x4B **(4 x 11)**	User 10 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x4C **(4 x 12)**	User 11 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x4D **(4 x 13)**	User 12 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x4E **(4 x 14)**	User 13 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x4F **(4 x 15)**	User 14 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x51	User 15 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)

Table 24 Fault Table: The Set/Clear Conditions & Fault Actions, cont'd

Flash Code	Fault Name	Set/Clear Conditions	Fault Actions
0x52	User 16 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x53	User 17 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x54	User 18 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x55	User 19 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x56	User 20 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x57	User 21 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x58	User 22 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x59	User 23 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x5A (5 x 10)	User 24 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x5B (5 x 11)	User 25 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x5C (5 x 12)	User 26 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x5D (5 x 13)	User 27 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x5E (5 x 14)	User 28 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x5F (5 x 15)	User 29 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x61	User 30 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x62	User 31 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)
0x63	User 32 Fault Fault Type(s): See User 1 fault (above)	See User 1 fault (above)	See User 1 fault (above)

Table 24 Fault Table: The Set/Clear Conditions & Fault Actions, cont'd

Flash Code	Fault Name	Set/Clear Conditions	Fault Actions
0x71	Supervision Fault Type(s): Curtis Supervision Code	**Set:** Internal controller failure **Clear:** Reset Controller.	Shutdown all driver output and digital output
0x72	Supervision Input Check Fault Type(s): 1	**Set:** Damaged Controller. **Clear:** Reset Controller.	Shutdown all driver output and digital output
0x73	Encoder 1 Fault Fault Type(s): 1 = sine_cosine encoder input voltage is out of range 2 = sawtooth encoder input voltage is out of range	**Set:** For sine_cosine encoder, the input voltage exceed the **Sin Max Voltage(Cos Max Voltage)** limit or dropped below **Sin Min Voltage(Cos Min Voltage)** limit. For sawtooth encoder, the input voltage exceeded **Sawtooth Encoder Max Voltage** or dropped below **Sawtooth Encoder Min Voltage.** **Clear:** Bring encoder voltage input within limit and reset the controller.	Encoder stops work.
0x74	Encoder 2 Fault Fault Type(s): See ENCODER_1_FAULT above.	See ENCODER_1_FAULT above. Check Encoder 1 & 2 settings: selections must match	Encoder stops work.
0x75	Pot Fault Fault Type(s): 1 = POT resistance measurement exceeded the tolerance limit 2 = POT position measurement exceeded the tolerance limit	**Set:** POT resistance or position measurement exceeded the tolerance limit(**Pot_Tolerance** setting) **Clear:** The POT resistance or position measurement is within the tolerance limit (10%).	No special action in software.
0x91	VCL Watchdog 1 Fault Fault Type(s): 1	**Set:** The time interval of feed VCL watchdog exceeded the timeout value. **Clear:** Reset controller	The fault actions can be defined by the User/OEM in the application-specific VCL software
0x92	VCL Watchdog 2 Fault Fault Type(s): 1	See VCL_WDT_1_FAULT above.	See VCL_WDT_1_FAULT above.
0x93	VCL Watchdog 3 Fault Fault Type(s): 1	See VCL_WDT_1_FAULT above.	See VCL_WDT_1_FAULT above.
0x94	VCL Watchdog 4 Fault Fault Type(s): 1	See VCL_WDT_1_FAULT above.	See VCL_WDT_1_FAULT above.
0x95	VCL Watchdog 5 Fault Fault Type(s): 1	See VCL_WDT_1_FAULT above.	See VCL_WDT_1_FAULT above.

8 — MAINTENANCE

There are no user serviceable parts in Curtis 1351 system controller. No attempt should be made to open, repair, or otherwise modify the controller. Doing so may damage the controller and will void the warranty.

It is recommended that the controller and connections be kept clean and dry and that the controller's fault history file be checked and cleared periodically.

CLEANING

Periodically cleaning the controller exterior will help protect it against corrosion and possible electrical control problems created by dirt, grime, and chemicals that are part of the operating environment and that normally exist in battery powered systems.

⚠ CAUTION

When working around any battery powered system, proper safety precautions should be taken. These include, but are not limited to: proper training, wearing eye protection, and avoiding loose clothing and jewelry.

1. Use the following cleaning procedure for routine maintenance. Never use a high-pressure washer to clean the system controller.
2. Remove power by disconnecting the battery. B+ first (at the battery, not the controller).
3. Remove any dirt or corrosion from the power and signal connector areas. The controller should be wiped clean with a moist rag. Dry it before reconnecting the battery. Connect the B+ last.
4. Make sure the connections are tight. Refer to Chapter 2 for the maximum tightening torque specifications for the battery connections.

FAULT HISTORY

The Curtis Integrated Toolkit™ Programmer (application tool) can be used to access the 1351 system controller's fault history file. The Programmer will read out all the faults the controller has experienced since the last time the fault history file was cleared.

- Faults such as driver, switch, and RTD inputs faults may be the result of loose wires; the system wiring should be carefully checked.

After a problem has been diagnosed and corrected, it is a good idea to clear the fault history file. This allows the system controller to accumulate a new file of faults. By checking the new fault history file at a later date, the technician can readily determine whether the problem was indeed fixed.

APPENDIX A

1351 CANBUS PDO MAP SETUP

This appendix provides guidance on how to set up PDO maps on the 1351 controller. It is similar to the setup for the F-series controllers, whose manual also provide a guide for setting up their PDO maps. Two methods for setting up the PDO maps include:

- Using the *Curtis Integrated Toolkit*™ (CIT).
- Using SDO write messages.

When selecting variables for the PDO maps, all parameters with the OEM Factory access-level (or below) are accessible by CANopen PDO or SDO.

Nomenclature note:

- A hexadecimal value written as 80000211h = 0x80000211.
- A CAN Object Index value written as XXXX.XX = 0xXXXX 0xXX, where the text ".XX" and 0xXX are the CAN Object's Sub Index value.

CANopen PDO Mapping Object description

CANopen CiA 301 specifies that 5 steps must be taken to re-map a PDO. This must occur while the NMT state is pre-operational.

1. Disable PDO
2. Set mapping by setting map-length to 0
3. Modify mapping
4. Enable mapping by setting map-length to correct value
5. Enable PDO

Structure of RPDO communication parameter object (4 bytes), the RPDO COB IDs.

Index	Sub-Index	Bit 31	Bit 30	Bit 29	Bits 11–28	Bits 7–10	Bits 0–6
		Disabled	Reserved	Frame	Reserved	Standard Message Type	Node-ID
CAN1: RPDO1: 1400h RPDO2: 1401h RPDO3: 1402h RPDO4: 1403h	01h	Enabled = 0 Disabled = 1	0	0 (11 bit CAN base frame)	00000h (Used for 29-bit extended frame)	RPDO1: 0200h + Node ID RPDO2: 0300h + Node ID RPDO3: 0400h + Node ID RPDO4: 0500h + Node ID	01h – 7Eh

Structure of TPDO communication parameter object (4 bytes), the TPDO COB IDs.

Index	Sub-Index	Bit 31	Bit 30	Bit 29	Bits 11–28	Bits 7–10	Bits 0–6
		Disabled	RTR not allowed	Frame	Reserved	Standard Message Type	Node-ID
CAN1: TPDO1: 1800h TPDO2: 1801h TPDO3: 1802h TPDO4: 1803h	01h	Enabled = 0 Disabled = 1	Must be 1	0 (11 bit CAN base frame)	00000h (Used for 29-bit extended frame)	TPDO1: 0180h + Node ID TPDO2: 0280h + Node ID TPDO3: 0380h + Node ID TPDO4: 0480h + Node ID	01h – 7Eh

Structure of PDO event timer (timeout) parameter object (2 bytes)

Index	Sub-Index	Bits 0–15
		Event Timer
CAN1: RPDO1: 1400h RPDO2: 1401h RPDO3: 1402h RPDO4: 1403h TPDO1: 1800h TPDO2: 1801h TPDO3: 1802h TPDO4: 1803h	05h	RPDO: PDO Time-out period. 0 if timeout check is disabled TPDO: PDO Time-out period. 0 if timeout check is disabled

Structure of PDO length object (1 byte)

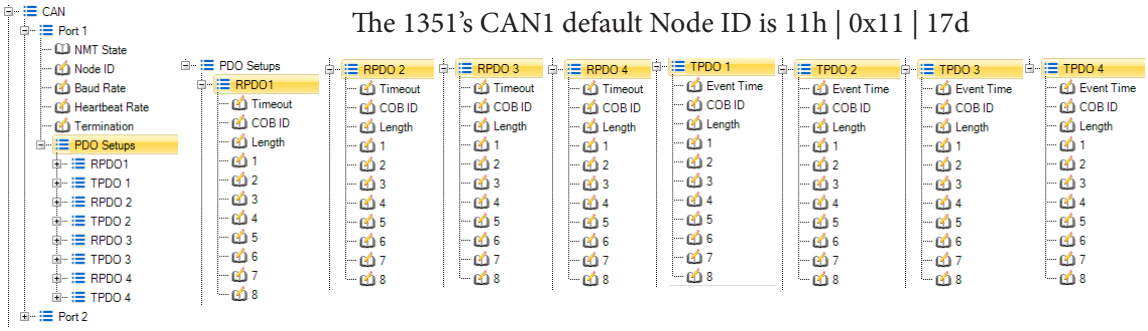
Index	Sub-Index	Bits 0–7
		Bit Length
CAN1: RPDO1: 1600h RPDO2: 1601h RPDO3: 1602h RPDO4: 1603h TPDO1: 1A00h TPDO2: 1A01h TPDO3: 1A02h TPDO4: 1A03h	00h	Number of objects is map (not the number of bits or bytes)

Structure of PDO mapping object (4 bytes)

Index	Sub-Index	Bits 31–16	Bits 8–15	Bits 0–7
		PDO Mapping Index	**Sub Index**	**Bit Length**
CAN1: RPDO1: 1600h RPDO2: 1601h RPDO3: 1602h RPDO4: 1603h TPDO1: 1A00h TPDO2: 1A01h TPDO3: 1A02h TPDO4: 1A03h	01h . . . 08h	PDO Mapping Index	Sub-Index	8, 16, or 32 Curtis does not allow mapping of individual bits. 8d = 8h 16d = 10h 24d = 18h 32d = 20h

CIT method: Programmer » Configuration » CAN » Port 1 » PDO Setups

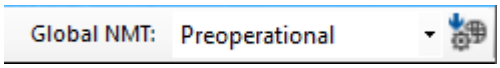
The 1351's CAN1 default Node ID is 11h | 0x11 | 17d



Example for RPDO Mapping with the Programmer

This example, we will be setting up RPDO1 with a Node ID 0x11, and map *Driver_1_Command* and *User1*.

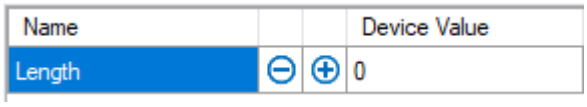
Make sure that the node is pre-operational node. If not, send an NMT message using any of these methods:

- CIT Package and Flash app. 

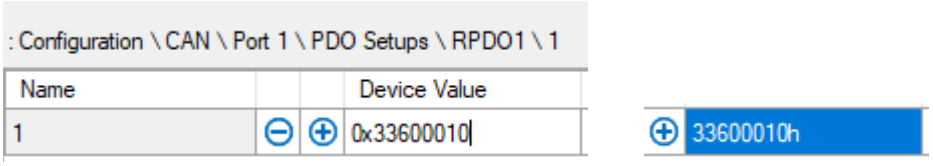
- PCAN-View (or similar).

The first step is to disable RPDO1. The 1351's default Factory Menu has the PDO's disabled. However, if the project does not, it disable it by setting the most significant bit of `can_rpdo_1_cob_id` to true (1). Navigate in the CIT Programmer to the following location Programmer » Configuration » CAN » Port 1 » PDO Setups. Set the `can_rpdo_1_cob_id` to 80000211. (Note, the most significant byte is 1000b = 8h, as the 31st bit = 1 disables the RPDO. See RPDO COB ID, above).

Next, disable the mapping of RPDO1 by setting `can_rpdo_1_length` to 0 (i.e., the Factory menu default).

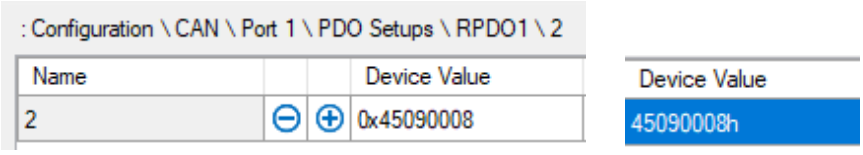


Map the 16-bit *Driver_1_Command* variable with CAN-object 0x3360.00, by setting `can_rpdo_1_map_1` to a value of 0x33600010. Note that when setting up a PDO that writes to an Operating System variable, the complete word must be written at once (32-bit write to 32-bit variable , 16-bit write to 16-bit variable). Input all values in hex. In this example, the 16-bit *Driver_1_Command* variable's length is 10h (i.e., the last 2 bytes).



Note: type-in the value as described (i.e., 0x33600010), which reverts to the 33600010h format after the enter-key (⏎).

Map 8 bits of the 32-bit *User10* variable with CAN-object 0x4509.00, by setting `can_rpdo_1_map_2` to a value of 0x45090008. In this example, the 8-bits of *User10* variable's length is 8h (i.e., the last byte).



Set `can_rpdo_1_event_timer` to a value in milliseconds, if a timeout check is required on Receive PDO messages. Here, the value is set to 200 milliseconds.

: Configuration \ CAN \ Port 1 \ PDO Setups \ RPDO1 \ Timeout

Name			Device Value
Timeout	⊖	⊕	200

Set `can_rpdo_1_length` to the number of variables (not bytes) that are mapped. That is 2 in this example.

: Configuration \ CAN \ Port 1 \ PDO Setups \ RPDO1 \ Length

Name			Device Value
Length	⊖	⊕	2

Now, the PDO can be re-enabled by setting `can_rpdo_1_cob_id` to the value 0x00000211. (i.e., the 31st bit is changed from 1 to 0 for Enabled, see the RPDO COB ID table, above).

: Configuration \ CAN \ Port 1 \ PDO Setups \ RPDO1 \ COB ID

Name			Device Value
COB ID	⊖	⊕	0x00000211

The PDO will become active when changing the NMT State to Operational.

Follow this format for mapping RPDO2 – 4, matching the message type (3rd byte) number, while retaining the same Node ID. For example, for Node ID = 0x11:

RPDO1 ... 80000211 = disabled, 00000211 = enabled
RPDO2 ... 80000311 = disabled, 00000311 = enabled
RPDO3 ... 80000411 = disabled, 00000411 = enabled
RPDO4 ... 80000511 = disabled, 00000511 = enabled

Example for TPDO Mapping with the Programmer

For this example, we will be setting up TPDO1 with a Node ID 0x11, and map *Keyswitch_Voltage* and *User20*.

Make sure that the node is pre-operational node (see the above RPDO example).

The first step is to disable TPDO1. This is similar the above RPDO, by setting the most significant bit of COB ID `can_tpdo_1_cob_id` to true (1). Navigate in the CIT Programmer to the following location Programmer » Configuration » CAN » Port 1 » PDO Setups. Set the `can_tpdo_1_cob_id` to C0000191. (Note, for the TPDO, the MSB is 1100 = Ch. See the TPDO COB ID table, above).

: Configuration \ CAN \ Port 1 \ PDO Setups \ TPDO 1 \ COB ID

Name			Device Value
COB ID	⊖	⊕	C0000191h

Next, disable the mapping of TPDO1 by setting `can_tpdo_1_length` to 0.

: Configuration \ CAN \ Port 1 \ PDO Setups \ TPDO 1 \ Length

Name			Device Value
Length	⊖	⊕	0

Map the 16-bit Keyswitch_Voltage variable with CAN-object 0x331C.00, by setting *can_tpdo_1_map_1* to a value of 0x331C0010. Note that when setting up a PDO that writes to an Operating System variable, the complete word must be written at once (32-bit write to 32-bit variable, 16-bit write to 16-bit variable). Input all values in hex. In this example, the 16-bit Keyswitch_Voltage variable's length is 10h.

: Configuration \ CAN \ Port 1 \ PDO Setups \ TPDO 1 \ 1

Name			Device Value
1	⊖	⊕	0x331C0010

Device Value
331C0010h

Note: type-in the value as described (i.e., 0x331C0010), which reverts to the 331C0010h format after the enter-key (↵).

Map 16 bits of the 32-bit User22 variable with CAN-object 0x4513.00, by setting *can_tpdo_1_map_2* to a value of 0x45130010. In this example, the 16-bits of User2 variable's length is 10h.

: Configuration \ CAN \ Port 1 \ PDO Setups \ TPDO 1 \ 2

Name			Device Value
2	⊖	⊕	0x45130010

Device Value
45130010h

Set *can_tpdo_1_event_timer* to a value in milliseconds, which sets the transmit period.

: Configuration \ CAN \ Port 1 \ PDO Setups \ TPDO 1 \ Event Time

Name			Device Value
Event Time	⊖	⊕	40

Set *can_tpdo_1_length* to the number of variables (not bytes) that are mapped. That is 2 in this example.

: Configuration \ CAN \ Port 1 \ PDO Setups \ TPDO 1 \ Length

Name			Device Value
Length	⊖	⊕	2

Now, the PDO can be re-enabled by setting *can_tpdo_1_cob_id* to value 0x40000191. (i.e., the 31st bit is changed from 1 to 0 for Enabled, setting the MSB to 0100b = 4h. See TPDO COB ID table, above).

: Configuration \ CAN \ Port 1 \ PDO Setups \ TPDO 1 \ COB ID

Name			Device Value
COB ID	⊖	⊕	0x400000191

Device Value
40000191h

The PDO will become active when changing the NMT State to Operational.

Follow this format for mapping TPDO2 - 4, matching the message type (3rd byte) number, while retaining the same Node ID. For example, for Node ID = 0x11:

TPDO1 ... C0000191 = disabled, 40000191 = enabled
TPDO2 ... C0000291 = disabled, 40000291 = enabled
TPDO3 ... C0000391 = disabled, 40000391 = enabled
TPDO4 ... C0000491 = disabled, 40000491 = enabled

Example for RPDO Mapping with SDO Writes

For this example, to set up RPDO1, and map *Driver_1_Command* (0x3360 0x00) and *User10* (0x4509 0x00).

Send CAN messages in the following table's order to set up RPDO1 mapping on a device with Node ID 0x28.

An example using PCAN-View messages is shown, Steps 1 - 9.

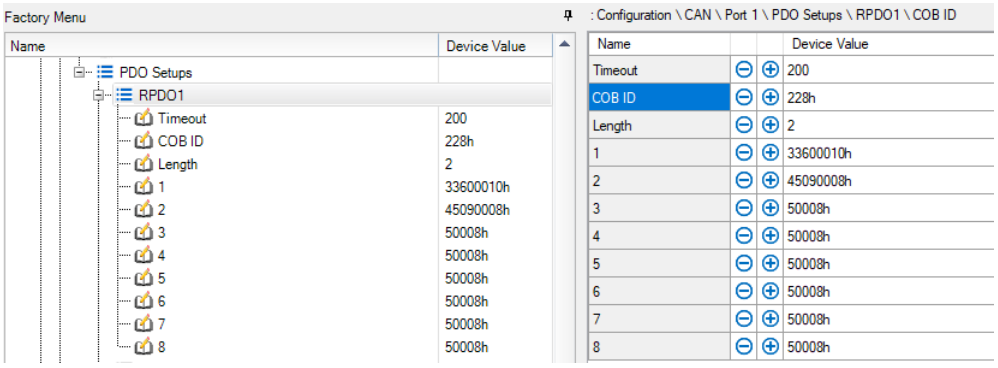
Note that the Data fields are Little Endian format (e.g., Object Index 0x3360.00 is input as 00 60 33), and this example uses Node ID = 0x28

Header	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Description
000	80	28							Send NMT Pre-Operational node 0x28
628	23	00	14	01	28	02	00	80	4 Byte SDO Write: Disable RPDO1
628	2F	00	16	00	00				1 Byte SDO Write: Disable Map/ Set length to 0
628	23	00	16	01	10	00	60	33	4 Byte SDO Write: Map 1st Object as *Driver_1_Command* (0x3366 0x00)
628	23	00	16	02	08	00	09	45	4 Byte SDO Write: Map 2nd Object as *User10* (0x4509 0x00)
628	2F	00	16	00	02				1 Byte SDO Write: Enable Map, Set the length to match the variables, here = 2
628	2B	00	14	05	C8	00			1 Byte SDO Write: Set PDO Timeout to 200 ms (200d = C8h)
628	23	00	14	01	28	02	00	00	4 Byte SDO Write: Enable PDO
000	01	28							Send NMT Go Operational to Node ID 0x28

The resulting PDO Mapping record

PDO Record	Mapped Object Data	Description
1400.01	00 00 02 28h	RPDO1 communication object enabled for node 0x28
1400.05	00 C8	RPDO1 Timeout set to 200 ms
1600.00	02h	RPDO1 Map Length of 2 (variables)
1600.01	33 60 00 10h	RPDO1 first mapped object 3360.00 (*Driver_1_Command*) with 16 bit length
1600.02	45 09 00 08h	RPDO1 second mapped object 4509 (User10) with 8 bit length

The resulting RPDO mapping record is reviewable in Programmer » Configuration » CAN » Port 1 » PDO Setups



VCL FUNCTIONS

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Map_Two_Points	5	2 Point Interpolation Routine	This routine interpolates between two points Y1 and Y2 based upon an input x. Interpolating based on X1 and X2
Automate_ABS	2	Run the absolute value function automatically	This function returns the absolute value of its input
Get_ABS	2	Get the Absolute Value of the Input	This function returns the absolute value of the input variable.
Send_Mailbox	1	Request that a Message Buffer be sent	This function forces the mailbox contents to be sent out. This is necessary for any mailboxes that have been setup with the C_EVENT constant. The mailbox must have been setup and its data pointers defined (setup_mailbox and setup_ mailbox_data functions called) before using this function!
Setup_CAN_Transmit_ Mailbox	6	Setup the General Parameters for a transmit mailbox.	Setup the General Parameters for a Transmit Mailbox. This function sets up the initial parameters for a mailbox. This function MUST BE called before defining the data for the mailbox (see setup_mailbox_data) or trying to use the mailbox.
Setup_CAN_Receive_ Mailbox	10	Setup the General Parameters for a receive mailbox	This function sets up the initial parameters for a receive mailbox.
Setup_CAN_Receive_Data	4	Maps variables for a receive mailbox	Define the pointers used with a Message Buffer. This function is used to define the 'data pointers' for a given mailbox. The setup_mailbox function MUST HAVE BEEN called before defining the data with this function.
Setup_CAN_Transmit_Data	4	Maps variables for a transmit mailbox	This function is used to define the 'data pointers' for a given mailbox.
Enable_Receive_Mailbox	1	Enable VCL Receive Mailbox	Enable all the CANopen standard processing VCL_FUNCTION Enable_receive_mailbox; Enable VCL Receive Mailbox.
Disable_Receive_Mailbox	1	Disable VCL Receive Mailbox	VCL_FUNCTION Disable_receive_mailbox; Disable VCL Receive Mailbox.
Enable_Transmit_Mailbox	1	Enable VCL Transmit Mailbox	Enable all the CANopen standard processing VCL_ FUNCTION enable_transmit_mailbox ; Enable VCL Transmit Mailbox
Disable_Transmit_Mailbox	1	Disable VCL Transmit Mailbox	disable_transmit_mailbox :: VCL - Disable CANopen Processing VCL_FUNCTION disable_ transmit_mailbox ; Disable VCL Transmit Mailbox
Get_Received_Counter	1	This function retrieves the Receive Data Counter	VCL_FUNCTION get_received_counter
Get_Transmit_Counter	1	This function retrieves the Transmit Data Counter	VCL_FUNCTION get_transmit_counter
Get_Received_Status	1	This function retrieves the Received Status	VCL_FUNCTION get_received_status
Clear_Received_Status	1	This function Clears the Received Status	VCL_FUNCTION clear_received_status

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Get_Transmit_Status	1	This function retrieves the Transmit Status	VCL_FUNCTION get_transmit_status
Clear_Transmit_Status	1	This function Clears the Received Transmit Status	VCL_FUNCTION clear_received_transmit_status
Get_Receive_Timeout	1	This function retrieves the Receive Timeout Status	VCL_FUNCTION get_receive_timeout
Get_Receive_ID	1	This retrieves the arbitation ID for this receive mailbox	VCL_FUNCTION Get the received arbitration ID
Clear_Receive_Timeout	1	This function Clears the Receive Timeout Status	VCL_FUNCTION clear_receive_timeout
Assign_CAN_Mailbox	2	This function assigns a VCL CAN receive or transmit mailbox	Use this function to assign a receive or transmit mailbox and store the mailbox handle in a VCL variable. This way, the correct mailbox can be referenced when checking receive status or other mailbox properties.
Automate_Copy	3	Copy a Variable to another location automatically	This function copies a variable from source to destination
Automate_Block_Copy	4	Copy a block of Variables to another location automatically	This function allows you to copy a sequential block of variables from one location to another location. The source and destination blocks cannot overlap. The source and destination blocks full range must be within the variable table, otherwise a PT_RANGE error will be issued. If you want to disable automatic copying, simply set all parameters, except the ID, to zero (e.g., automate_block_copy(CPY3,0,0,0). If you do not set the length to zero, then it is likely that a PT_RANGE error will occur.
Automate_Limited_Block_Copy	9	Copy a block of Variables to another location automatically	This function allows you to copy four variables from one location to another location. The source and destination address for each variable to transfer must be specified. A source/destination pair will be ignored (i.e., not transferred) if either the source or the destination value is set to zero.
Setup_Delay	2	Setup a Time Delay	This function installs a new time delay
Setup_Delay_PreScale	2	Setup a Delay Timer's pre-scale	This routine setups up a pre-scale for time delay block. The default pre-scale is 1 (i.e., the timer counts down at a 1ms rate).
Automate_Filter	5	Filter a value	This function initiates filtering on the selected variable
Setup_Limit	3	Setup a Limit Function	This function installs new value for a limit block
Automate_Limit	3	VCL - Automate a Limit Function	This function sets one or more of the values in a limit block to be automatically updated
Set_Limit	2	Set the Limit Value	This function installs new value for a limit
Get_Limit	2	Get the Limited Value	This function returns the input 'clamped' by the limit
Setup_Map	16	Setup a map	This function specifies a map.
Automate_Map	2	Set the mapping up to be done automatically	This function allows you to set up a map that runs automatically. The 'source' is the variable that will be mapped.

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Get_Map_Output	2	Interpolate within a 2D table of N entries	This function maps the input variable to a new value using the specified map.
Get_Map_Segment	2	Return the Segment Containing X	This function returns the segment (0 to 7) of the input variable using the specified map. If the X input is less than the lowest X value in the table, then the segment number will be 0. If the X input is greater than the highest X value, then the segment number will be equal to the number of XY pairs in the table plus one. For example, if you have a table with three X/Y pairs (44,123, 100,230, 300,5000) then: setting FP2=33 would return 0 " FP2=88 " " 1 " FP2=150 " " 2 " FP2=400 " " 3
MulDiv	3	Multiply the Input by Multiple, then divide by Divisor	This function returns the value of its input multiplied by Multiplier parameter and then divided by Divisor parameter. If the divisor is equal to 0, and both multiplier and value are greater than zero or, if both multiplier and value are less than zero, then the output will be MAXIMUM = 32767; otherwise, the output will be –32768
Automate_MulDiv	4	Run the Multiply/Divide function automatically	This function returns the value of its input multiplied by Multiplier parameter and then divided by Divisor parameter.
Get_MulDiv	4	Multiply the Input by Multiple, then divide by Divisor	This function returns the value of its input multiplied by Multiplier parameter and then divided by Divisor parameter. If the divisor is equal to 0, and both multiplier and value are greater than zero or, if both multiplier and value are less than zero, then the output will be MAXIMUM = 32767; otherwise, the output will be – 32768
NVM_Write_Parameter	1	Write a parameter to non-volatile memory	This function writes the value of the parameter to non-volatile memory. Be aware, writes typically take 6ms to complete. You should monitor NVM_Status to determine when the operation finishes and/or check the function return value to make sure that the write was accepted. The parameter's value is saved in this routine (although you should not write to it again until the write operation has completed).
Automate_PID	8	Set the PID Loop up to run automatically	This function allows you to set up a PID that runs automatically on each clock tick. You can specify any Variable for the parameters: Setpoint, Feedback, Kp, Ki and Kd. The value of Kp is normalized to a value of 16384. What this means is that, when both the Setpoint and the Feeback variables range from 0 to 32767, a value of 16384 for Kp will cause the loop to track the setpoint. A lower value for Kp will result in the output being less than the setpoint, while a higher value of Kp will cause the output to exceed the setpoint. The MulDiv and/or Scale functions may be needed to adjust the values of Setpoint and Feedback.

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Reset_PID	1	Reset the PID Loop Variables	This function resets all of the PID loop variables
Automate_Driver	3		VCL_FUNCTION automate_driver; Make a pwm output update on automatically every 4ms
Setup_Ramp	4	Setup a ramp	This function allows you to initialize the following ramp variables. Warning: before you use a ramp you MUST set it up.
Automate_Ramp	4	Set the ramping up to be done automatically	This function allows you to set up a ramp that runs automatically on each clock tick. You can specify a Variable that will be the "target" of the ramp as well as a Variable that can control the ramp hold function. A value of 0 for Target, Hold or Rate leaves the value unchanged. This allows you to set some values (as constants) using the setup routine while allowing run-time control over others. For example, to set the target but have automatic control over the ramp hold (using whatever values is written to user1), you'd set FP2 (Target) to 0 and FP3 (Hold) to @user1.
Set_Ramp_Target	2	Set the Target Ramp Value	This function installs a new target value.
Set_Ramp_Hold	2	Set the Ramp Hold value	This function installs a new value for the hold-control. A value of 0 enables the ramps. Any non-zero value will hold the ramp at its present value.
Set_Ramp_Rate	2	Set the Ramp Rate Value	This function installs a new value for the rate. On each clock tick (1 millisecond) this value will be added or subtracted from the current ramp value if it is not already equal to the target value.
Random_Seed	1	'Seed' the random number generator	The random number generator is automatically seeded with the value of KSI * 2 when the system starts up. If you want, you can use some other pseudo-random number as the seed; or, write a particular number as the seed so that you always get the same (pseudo-random) sequence
Random	0	Return a (pseudo) random number	This routine returns the next (pseudo) random number.
Automate_Scale	3	Set the Scaling up to be done automatically	This function automates the scaling process. If the second (Input) or third (scale factor) parameters are set to a non-zero value, then that is assumed to be an index into the variable table where the actual value is stored. If the index to the scaling factor is not set (i.e. Scale = 0) then its value will not be altered by this routine and you can set it using a call to setup_scale_factor().
Setup_Scale_Factor **Setup_Scaling** (see Sys Info)	2	Setup a Scaling Factor	This function installs a scaling factor in the designated scaling block. The scaling factor is a signed, 16-bit, integer that will be interpreted as a percentage (i.e., 32767 = 1, 16383 = 0.5, and so on).
Scale_Value	2	Return the value of the input scaled by the Scale Factor	This function returns the value scaled by the block's scaling factor. The scale factor is interpreted as a 16-bit, signed, integer. For example, if the scale factor were 16,383 (equal to 50%), and the input value was 400, then the output would be 200.

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Setup_Select	3	Setup a Selector Switch Block	This function installs new values for a 2 position selector switch. If you set parameter 2 to 0 the selector block will be disabled.
Automate_Select	2	Automate the select variable	This function causes the input selection to be performed automatically. The second parameter identifies a variable that will be used to select the output of the specified selector block. If you set parameter 2 to 0 the selector block will be disabled.
Set_Select	2	Enter a new value for the select variable	This function set the value of the select parameter that is used to choose which input appears at the output (SELn_Output)
Setup_4p_Select	9	Setup a 4-Pole Selector Switch Block	This function installs new values for a 4-Pole, Double-Throw selector switch. You can set any source parameter (P#_Zero or P#_NonZero) to 0 to disable its transfer. In this case, the value of the output will remain unchanged. For example, if you set up SEL4 with both P2_Zero and P2_NonZero set to zero, the value of sel4_4p_2_output will not be changed. If you set P2_Zero to 0 (but have defined P2_NonZero), the value of sel4_4p_2_ output will only be changed when the selector is non-zero.
Automate_4p_Select	2	Automate a 4-Pole Select Variable	This function causes the input selection to be performed automatically. The second parameter identifies a Variable that will be used to select the output of the specified selector block. If you set parameter 2 to 0 the selector block will be disabled.
Set_4p_Select	2	Enter a new value for a 4-pole select variable	This function set the value of a 4-pole select parameter that is used to choose which input set appears at the output: the ‘zero’ set or the 'non-zero' set.
put_message_to_string	5	Place a message into an array of ASCII numbers	Construct a message and parse it into a list of ASCII character equivalents. This function constructs a 12-character ASCII message with pre-text on the left, a variable’s value in the middle, and post-text on the right. The variable's value can have a decimal place added using the NFormat specifier If the resulting message is longer than 12 characters, truncation occurs. -- The variable’s value is truncated. -- The post-text is truncated second. -- The pre-text is truncated last. -- All truncation occurs on the right of a section. --- All unused array positions will be filled with a space (ASCII 32)
		EXAMPLE: If the variable = 12345, the pre-text is “ML”, the post-text is “RPM”, the decimal format is PSM_DECIMAL_1 and the ArrayStart is User101 Then: User101 = 77 “M” User107 = 52 “.” User102 = 76 “L” User108 = 46 “4” User103 = 32 “ ” User109 = 53 “5” User104 = 49 “1” User110 = 82 “R” User105 = 50 “2” User111 = 80 “P” User106 = 51 “3” User112 = 77 “M”	

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Reset_Controller	0	Hardware reset	Perform a software reset
Setup_Timer	4	Setup a Timer	This function initializes a timer to a particular value
Enable_Timer	1	Enable a Timer	This function Enables a timer (starts it timing)
Disable_Timer	1	Disable a Timer	This function Disables a timer (stops it from timing)
Reset_Timer	2	Reset a timer to Zero	This function resets all of the timer values (ms, sec, min, hr and day) to zero and then sets the Tmr_Enable flag equal to the second parameter
vcl_get_size	1	Return the size (in bytes) of a variable	Return the size (in bytes) of a variable
Clear_Diagnostic_History	0	Clear the fault history log	This routine clears the Fault History log
Disable_Fault_Clear	1	Get Fault Code	The prescribed fault is cannot be cleared after calling this function
Enable_Fault_Clear	1	Get Fault Code	The prescribed fault is can be cleared after calling this function
Get_Fault_Code	1	Get Fault Code	The fault stack ranges from 0 to the number of faults current active. This routine allows you to recover fault codes from the fault stack by passing the index to the fault stack and getting back the fault code. To use this routine, start with an index of zero. Call this routine. If you get back a value of zero, there are NO faults. If you get a value other than zero, report the fault and increment the index. Then, repeat the process until you do get a value of zero.
Set_VCL_Fault	2		VCL_FUNCTION set_vcl_fault
Clear_VCL_Fault	1		VCL_FUNCTION clear_vcl_fault
Set_VCL_Fault_Action	2		VCL_FUNCTION set_vcl_fault action
Get_CRC	2		VCL_FUNCTION Get CRC value of a byte
Set_Receive_Timeout	2	Set the receive timeout in ms for an assigned receive mailbox.	This function is used to update the receive timeout of an assigned receive mailbox.
Enter_Start_Manager	1	Vector to the start manager	This routine vectors to the start manager and stays there.
Sin	1	returns the sine of a number	VCL_FUNCTION sine()
Cos	1	returns the cosine of a number	VCL_FUNCTION cosine()
Tan	1	returns the tangent of a number	VCL_FUNCTION tangent()
ASin	1	returns the arcsin of a number	VCL_FUNCTION arcsin()
ACos	1	returns the arccos of a number	VCL_FUNCTION arccos()
ATan	2	returns the arctan of two lengths (y,x)	VCL_FUNCTION arctangent()
Sqrt	1	returns the square root of a number	VCL_FUNCTION sqrt()
Setup_NMT_State	2	specific to each CAN port	This function is used to control the NMT state of two CAN ports.

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Setup_3140T	1	Setup a 3140 gauge	Setup a 3140 gauge and returns handle for a 3140 gauge. The OS will automatically send an NMT message to the gauge to make it go operational. It will be resent if at any time the gauge inadvertently goes out of operational mode. Once the gauge is operational, periodic PDO communication will be started.
Put_3140T_Icons	2	Changes the Icons for the 3140 gauge	This function takes a 16-bit VCL-constructed command word for the 3140 gauge icons. Once the command word is supplied to this function, it is periodically sent to the 3140 gauge in the first two bytes of RPDO1. For the exact order of command icons, please refer to the 3140 documentation.
Put_3140T_Backlight	2	Changes the backlight for the 3140 Gauge	Changes the backlight for the 3140 Gauge
Put_3140T_Message	6	Put a message on the 3140 gauge	Places a message on either the large 3 character upper display or the small 6 character lower display. This function installs a new message for the gauge with the pre-text on the left, the number in the middle (right aligned), and the post-text on the right. The number and post-text will be printed only if adequate space remains after the pre-text has been printed. The number will be printed only if there is adequate space remaining after the post-text has been printed. PreText and PostText fields are left justified (i.e., all truncation occurs on the furthest right characters). Number field is right justified (i.e., all truncation occurs on the furthest left characters)
Put_3140T_Timer	2	Puts an hourmeter timer value on the 3140 Gauge in tenths of hours	This command does in a single command what can be done using Put_3140T_Icons and Put_3140T_Message in multiple steps. This command will temporarily override the input that was given to Put_3140T_Icons, such that: The lower text is on (solid), the hourmeter icon is on (solid), and the decimal point is on (solid). It will override these relevant command-word bits, even if a new Put_3140T_Icons command is received. It will release overriding these bits when a Put_3140_Message is issued to the SMALL_TEXT. Once released, the command-word of the last Put_3140T_Icons must be used.
Set_3140T_Hourmeter_Enable	2	Turns the internal gauge hourmeter counter ON or OFF	Turns the internal gauge hourmeter counter ON or OFF
Setup_840C	1	Setup an 840C gauge.	Returns handle for an 840C gauge.
Put_840C_LED	2	Changes the LEDs on a 840C gauge	Changes the LEDs on a 840C gauge
Put_840C_Message	5	Put a message on the spyglass	Put a message on the spyglass
Put_840C_Timer	2	Puts an hourmeter timer value on the spyglass in tenths of hours	Puts an hourmeter timer value on the spyglass in tenths of hours
Get_Fault_Last_Time	1	See Chapter 7.	
Get_Fault_First_Time	1	See Chapter 7.	

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Get_Fault_Count	1	See Chapter 7.	
Get_Fault_Type	1	See Chapter 7.	
Get_Fault_CAN_Id	1	See Chapter 7.	
Get_Flash_Code	1	Error Codes. See Chapter 7.	
Get_Parameter_Min_Raw	1	Gets the minimum raw value of an input parameter.	Gets the minimum raw value of an input parameter.
Get_Parameter_Max_Raw	1	Gets the maximum raw value of an input parameter.	Gets the maximum raw value of an input parameter.
Get_Parameter_Min_ Display	1	Gets the minimum display value of an input parameter.	Gets the minimum display value of an input parameter.
Get_Parameter_Max_ Display	1	Gets the maximum display value of an input parameter.	Gets the maximum display value of an input parameter.
Get_Parameter_Exponent	1	Gets the exponent display value of an input parameter.	Gets the exponent display value of an input parameter.
Get_Parameter_Default	1	Gets the raw default of the parameter	Gets the raw default of the parameter
Get_Floating_Point_Scaled	2	Sets a floating point parameter, multiplied by a scale factor	Gets a floating point parameter, multiplied by a scale factor
Set_Floating_Point_Scaled	3	Sets a floating point parameter, multiplied by a scale factor	Sets a floating point parameter, multiplied by a scale factor
Get_Diagnostic_Timer	0	Returns the value of the system timer used for fault first time and fault last time.	
Get_Master_Timer	0	Returns the value of the controller's total key on (power) time. This cannot be reset.	
Clear_Diagnostic_Timer	0	Clear the diagnostic history Timer	
Fault_Active	1	Check if a fault is currently active in the controller	This function checks if a fault is currently active in the controller
Put_Driver (✔)	2	This function stores a new value into the PWM output and digital output.	Modified function to set a driver command. 0-1000 = 0 to 100.0%
reset_pulse_counter (✔)	1	High Speed Inputs (1 or 2) Count value	Rest the high speed digital input counter
Control_External_Power (✔)	2	Control the output of external supplies	This function controls the external 5V and 12V output.
setup_canopen_transmit_ mailbox (✔)	4	Sets up the initial parameters for a CANopen transmit mailbox.	This CANopen specific function sets up the initial parameters for a CANopen mailbox. It serves the same purpose as the generic setup function, but simplifies the setup for CANopen users.
define_canopen_transmit_ data (✔)	4	Used to define the data sent from the mailbox.	This function is used to define the data sent from the mailbox.
setup_canopen_receive_ mailbox (✔)	5	Sets up the initial parameters for a CANopen receive mailbox	This CANopen specific function sets up the initial parameters for a CANopen mailbox. It serves the same purpose as the generic setup function, but simplifies the setup for CANopen users
setup_receive_mailbox_ auto_reply (✔)	2	Configure the mailbox to be sent upon receipt of data.	Configure a transmit mailbox to be sent upon receipt of data in this RX mailbox.

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
setup_receive_mailbox_ timeout (✔)	3	Set up the timeout period for a timeout fault.	Setup a timeout fault that is active if this mailbox does not receive new data within the Timeout period.
define_canopen_receive_ data (✔)	4	Used to define the data received in a mailbox is arranged	This function is used to define how the data received in a mailbox is arranged.
send_NMT (✔)	3	Control the state of the CANopen ancillary(s).	When a port is enabled to be a manger, this function is used to control the state of ancillary(s).
get_nmt_state (✔)	2	Get the state of the device requested.	Returns the state of the device requested by monitoring the Heartbeat messages
enable_node_guarding (✔)	1	Turns on heartbeat monitoring.	Turns on heartbeat monitoring on specified CAN port.
disable_node_guarding (✔)	1	Turns off heartbeat monitoring.	Turns off heartbeat monitoring on specified CAN port
setup_heartbeat (✔)	3	Sets the expected rate for a required Node ID's heartbeat	This function adds a can device into node guarding slots.
get_heartbeat_time (✔)	2	Get the time between last two heartbeats	If the heartbeat has not been lost, this function will return the time between the last two heartbeats.
check_heartbeat_status (✔)	2	Returns the status of the heartbeat timeout check	Returns the status of the heartbeat timeout check on the specified Node ID
enable_emergency_ message_monitor (✔)	1	Starts Emergency Message monitoring	Enable the system to monitor emergency messages on the bus.
disable_emergency_ message_monitor (✔)	1	Disable Emergency Message monitoring	Disable the monitor function
setup_emergency_ message_monitor (✔)	2	Monitor a Node for Emergency messages	Setup a specific node to be monitored for emergency messages
check_canopen_ emergency (✔)	2	Checks for message on port & device	Check if there was an emergency message active on this port and ancillary device.
get_canopen_emergency_ message (✔)	1	Returns the first the full 8 bytes of emergency message and node_Id and places them into the pre-defined VCL variables	This function is used to get the emergency message data by index from message buffer.The depth of the message buffer is 64.
clear_emergency_ message_buffer (✔)	3	Remove messages from the emergency message buffers.	This function can be used to remove messages from the emergency message buffers. This might be needed if messages have not been removed in a long time and the buffer is full.
setup_transmit_SRDO (✔)	4	Links two mailboxes to be transmitted in a linked fashion.	Setup the necessary CANopen compliant buffers, timers and mailboxes to send a SRDO.
setup_receive_SRDO (✔)	6	sets up the timing detection and fault functions for a receive SRDO	Setup the necessary CANopen compliant buffers, timers and mailboxes to Receive a SRDO.
enable_transmit_SRDO (✔)	1	start the transmit SRDO	Start the transmit SRDO. If set to a cyclic rate, this will start transmissions.
enable_receive_SRDO (✔)	1	start the receive SRDO	Enable the SRDO receive functionality
send_SRDO (✔)	1	Initiates TX of the two SRDO messages	Initiates TX of the two messages associated with this SRDO with an interspace timing.
check_SRDO_receive (✔)	1	Check the state of SRDO receive	Check the data and timing integrity of the RSDO reception
disable_transmit_SRDO (✔)	1	stop the transmit SRDO	Disable the SRDO transmit functionality
disable_receive_SRDO (✔)	1	stop the receive SRDO	Disable the SRDO receive functionality

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
request_SDO_read (✔)	5	Send an SDO read request to a specific node	This function sends an expedited SDO request to read a value from a specific device identified by its Node ID. The function will return a handle which is used to check the status of the particular request and pulls the data from the input reception buffers.
check_SDO_read (✔)	1	Check if a specific request for SDO data has been fulfilled	Check the SDO receive buffer for a specific message response
get_SDO_length (✔)	1	Return the number of valid bytes in the SDO reception	Find out how many active data bytes are in the received message
get_SDO_data (✔)	1	Pull the data from the request receive buffer	Get the received data
remove_SDO_read (✔)	1	Remove a SDO request from SDO read buffer	Remove a message for the receive buffer
write_SDO (✔)	7	Send an SDO write request to a specific node	This function sends an expedited SDO request to write a change to a parameter in a specific device identified by its Node ID. The function will return a handle used to check the status of the SDO write.
check_sdo_write (✔)	1	Check if a specific write for SDO data has been fulfilled.	Check of the response message
remove_SDO_write (✔)	1	Remove a SDO request from SDO write buffer.	This function allows the VCL program to remove a single Write SDO from the appropriate buffer.
clear_SDO_read_buffer (✔)	0	Clears the SDO read buffer	This function can be used to remove all messages from the respective read buffers. This might be needed if messages have not been removed in a long time and the buffer is full.
clear_SDO_write_buffer (✔)	0	Clears the SDO write buffer	This function can be used to remove all messages from the respective write buffers. This might be needed if messages have not been removed in a long time and the buffer is full.
set_watchdog_timeout (✔)	2	Sets the time-out in milliseconds for any one of five watchdog timers	This function will set the time-out in milliseconds for any one of five watchdog timers available. When the function is called, the specific watchdog timer is cleared and the time-out threshold is updated.
kick_watchdog (✔)	1	Starts and reset the specified watchdog timer	The first call of this starts the watchdog timer (required). Succeeding calls will reset (re-starts) the watchdog timer timing.
set_watchdog_fault_action (✔)	2	Set user defined actions when the watchdog timer times-out.	When a watchdog timer times-out, the user-defined actions are launched by the OS. These watchdog fault actions are setup in identical fashion to the User Fault Action in VCL.
Setup_Driver_Regulator (✔)	9	Set up the parameters for the driver current regulator	This routine allows VCL to set up the parameters for the driver current regulator.
Setup_Driver_Pull_In_And_Hold (✔)	3	Set up the parameters for the driver initial level and time	This routine allows VCL to set up the parameters for the driver initial level and time.
get_canopen_device_emergency_message (✔)	3	Returns the first the full 8 bytes of emergency message of specified device and places them into the pre-defined VCL variables	This function is used to get the emergency message data of the specified device by index from message buffer. The depth of the message buffer is 64.

Function Name new/unique to 1351 = (✔)	Arguments	Short Comment	Description
Enter_Sleep_Mode (✔)	3	Puts the controller into low power mode, 10mA max.	Specifies an Exit Type, a Switch Mask, and the Timer is seconds.
Reset_Position (✔)	1	Reset_Position (ENC1 or ENC2)	Encoder Position monitor variable
automate_frequency_ output (✔)	5	Set up a frequency output on driver10.	This function sets up PWM Driver10 (pin 33) output to yield a frequency proportional to the input variable at an execution rate of 16 ms. This output can be used to drive an electronic speedometer or tachometer.
disable_frequency_output (✔)	0	No argument(s) Turns off the frequency output	This routine is used to turn off the Driver 10 frequency output function
Set_Accelerometer_Range (✔)	2	Set the range of the accelerometer	This routine is used to set the scale (from 2g to 16g) of the accelerometer.

APPENDIX B — VEHICLE SYSTEM DESIGN CONSIDERATIONS-&-RECYCLING

VEHICLE DESIGN CONSIDERATIONS

REGARDING ELECTROMAGNETIC COMPATIBILITY (EMC)

AND ELECTROSTATIC DISCHARGE (ESD)

ELECTROMAGNETIC COMPATIBILITY (EMC)

Electromagnetic Compatibility (EMC) encompasses two areas: emissions and immunity. Emissions are radio frequency (RF) energy generated by a product. This energy has the potential to interfere with communications systems such as radio, television, cellular phones, dispatching, aircraft, etc.

Immunity is the ability of a product to operate as intended in the presence of RF energy generated by other sources as well as itself. EN12895 is the relevant EMC standard for the CE marking of industrial trucks intended for sale in Europe and some other countries.

EMC Compliance is ultimately a system requirement. Part of the EMC performance is designed into or inherent in each component of a system; another part is designed into or inherent in end product/system characteristics such as shielding, wire routing, individual component layout and a portion is a function of the interactions between all these parts. The techniques presented below can help reduce the risk of EMC problems in products that incorporate Curtis system and motor controllers.

Emissions

High frequency signals can produce RF emissions that are measurable during Radiated Emissions testing. Long cable and wire harness runs essentially become antennas for the emissions to travel (beyond the source). Therefore, emission reduction techniques include making the battery and motor cables as short as possible. Minimize the lengths of the AMPseal connector's wire harness runs and the formation of wire loops. Further emission decreases may include using shielded cables or ferrites on the control wires and twisting the motor and battery cables. Route the battery and AC motor cables separate from the control wires. When separating control wires and the battery/motor cable routing is not possible, cross them at right angles.

RF Immunity

Radiated immunity problems may occur when the controller is located close to other devices generating high RF energy. Possible ways to help prevent other devices from interfering with a Curtis controller include:

- Placing the controller as far as possible from such noise sources.
- Shield the controller from the noise
- Enclose the controller in a metal box and add proper ferrites to all cabling entering and leaving it.
- Other possible solutions include the use of ferrite beads at the RF noise source(s) to prevent the noise from traveling along the wiring harness and cross-conducting onto sensitive wires and common connections.

Curtis controllers contain ESD-sensitive components. It is therefore necessary to protect them from ESD damage. See Table 3 (page 12) for the controller ESD ratings.

ESD immunity is improved by providing sufficient distance or isolation between conductors and the ESD source so that a discharge will not occur.

DECOMMISSIONING AND RECYCLING THE CONTROLLER

The controller is for installation into an Original Equipment Manufacturer (OEM) vehicle. Ideally, as the system manger installed as part of the specific vehicles' electrical or electro-hydraulic control system.

For controller decommissioning and recycling:

1. Follow the OEM's vehicle decommissioning instructions.
2. Follow all applicable landfill directives or regulations for Electrical and Electronic Equipment (EEE) waste.

APPENDIX C — PROGRAMMING DEVICES FOR THE 1351

This manual was written using Curtis software and hardware "tools" to access the parameters and monitor items. These tools are required to setup and fully utilize the 1351 System Controller. They also access the faults and offer diagnostic routines.

Curtis Integrated Toolkit™

The PC based Curtis Integrated Toolkit™ (CIT) program communicates with the 1351 System Controller over the CANbus. An interface device to connect the PC to the CANbus is required. CIT is compatible with many leading USB ↔CAN interface dongles from Peak, Kvaser, iFAC, Sondheim, etc. The Curtis Integrated Toolkit™ program has embedded help instructions. A "license-key" is required for the CIT program (available from Curtis*).

The Curtis Integrated Toolkit™ main user-interface, known as the Launchpad, controls access to the application tools used to:

- Adjust/set parameters (i.e., "program the controller"), modify* their menus, or add or group new parameters.
- Write, edit, compile, and save VCL code.
- Troubleshoot and analyze a controller (device) incorporated into a vehicle system.
- Save an application as a project, and apply the project settings to multiple vehicle system controllers.
- Update (upload) the 1351 firmware/software and VCL programs.

Curtis 1313 handheld programmer

The Curtis 1313 handheld programmer (1313 HHP) is a self-contained parameter programmer and diagnostic tool for the 1351 system controller. The model required uses the CANbus. The 1351 system controller does not support the serial-based 1313 HHP. The generic CANbus 1313 HHP is model 1313-xx31. It has the blue band. Its operation is fully explained in the 1313 HHP user's manual, *CANbus 1313 HHP, 53225 Rev A 3/18*, downloadable from the Curtis website*.

The 1313 HHP can upload new software to the 1351, yet it does not upload VCL programs.

1313 HHP

** Contact your Curtis distributor or the regional Curtis sales office to obtain the Curtis Integrated Toolkit ™ and the 1313 HHP. Consult with your Curtis support engineer for help or training with the setup and using these tools and the 1351 System Controller.*

APPENDIX D — 1351 MODELS AND SPECIFICATIONS

Table D Model Chart and Specifications

Item	1351-5001	1351-7001
Voltage	12-48V	36-96V
CAN ports	2	
Analog Inputs	11	
Dynamic Pot Input	1	
RTD Inputs	4	
Switch Inputs	14	
High Speed Inputs	2	
Encoder Inputs	2	
PWM Outputs	10	
Digital Outputs	3	
Half Bridge Outputs	2	
Safety Output	1	
Analog output	1	
External Supply	(1 × 5V and 1 × 12V)	
Nominal Input Voltage	12 – 48 Volts	36 – 96 Volts
Minimum Operating Voltage (after startup)	6 Volts	15 Volts
Maximum Operating Voltage	60 Volts	120 Volts
Electrical Isolation to Heatsink	500 Vac	1200 Vac
Storage Ambient Temperature	–40 – 85° C	
Operating Ambient Temperature	–40 – 50° C	
Package Environmental Rating	IP65 as per IEC60529	
Weight	0.60 Kg	
Dimensions WxLxH	160mm × 125mm × 38.5mm	
EMC	Designed to the requirements of EN 12895:2015	
Safety	Designed to the requirements of EN ISO 13849-1:2008	
UL	UL recognized component per UL583	