DVC700 Family

System and Programming User Guide

The information in this publication is intended as a guide only, and HCT assumes NO responsibility for usage and implementation in any user written application code structure.
HCT strongly suggests that the user attends one of the product training courses to ensure correct and full understanding of this information and to learn further optimized methods of control techniques.
Please contact HCT customer service to book one of the scheduled training dates or to discuss arranging a course specific to your company needs.

Thank you for using High Country Tek Inc. Products.

Application and BIOS Compatibility with previous versions of DVC products

DVC7/10 applications compiled using the previous (4.x) or older versions of the Programming Tool will not work on a DVC707/710 without first compiling the application in the new 5.4 Programming Tool as a DVC707/710 project. This will bring the memory maps and/or internal variables into alignment with the DVC710's 5.4x BIOS and expansion modules. (This may be accomplished by manually editing the .dvc file using any text editor.)   Programming Tool 5.4 generated applications will not run on any DVC10 using the 4.x BIOS.

The Intella® Programming Tool version 5.4(x) may be used to program both DVC707 and DVC710 projects. When converting a project from one master module to another, it is best to only convert from the DVC707 to the DVC710.   This will help avoid truncating features that are not available in the DVC707.

If converting from a DVC707 to DVC710 project or from a DVC710 to DVC707 project, any written code that uses different internal variable headers such as "DVC80" for the DVC707 and "J1939" for the DVC710 must be manually changed to match the format of the module in the current project. (The Edit » Replace All feature is a good tool to do this) for Example;

"DVC80.messagename.NoRsp" would need to be changed to "J1939.messagename.NoRsp" when converting from a DVC707 to DVC710 project.

DVC710 Variable Quick List

Process PI	Uni and Ana Inputs	Outputs SCLS	Outputs DCHS	Outputs SCHS
Name.Setpoint	InputName	PWM Name	PWM Name	PWM Name
Name.Feedback	NameLo	Name.Enable	Name.Dir	Name.Enable
Name.ProErr	NameHi	Name.Short	Name.Enable	Name.Short
Name.ProSumErr	Name.Dir	Name.Open	Name.Short	Name.Open
Name.ProP	Name.RawVolts	Name.Rampup	Name.Open	Name.Rampup
Name.ProI	Name.RefVolts	Name.Rampdown	HSName.Rampup	Name.Rampdown
Name.ProItime	Name.MinVolts	Name.Frequency	HSName.Rampdown	HSName
Name.Cur	Name.MaxVolts	Name.Dutycycle	HSName.Short	HSName.Short
Name.RampCur	Name.MinLimit	Name.Freqerror	HSName.Open	HSName.Open
Name.CurErr	Name.MaxLimit	HSName	Name.Cur	HSName.OpenDisable
Name.CurSumErr	Name.RefMinLimit	HSName.Short	Name.RampCur	Name.Cur
Name.CurP	Name.RefMaxLimit	HSName.Open	Name.CurErr	Name.RampCur
Name.CurI	Name.CenterVolts	HSName.OpenDisable	Name.CurSumErr	Name.CurErr
Name.MinCurA	Name.Deadbandv	Name.Cur	Name.CurP	Name.CurSumErr
Name.MaxCurA	NameX	Name.RampCur	Name.CurI	Name.CurP
Name.MinCurB	Name.MinF	Name.CurErr	Name.MinCurA	Name.CurI
Name.MaxCurB	Name.MaxF	Name.CurSumErr	Name.MaxCurA	Name.MinCurA
Name.Config	Name.LOS	Name.CurP	Name.MinCurB	Name.MaxCurA
	Name.RealRPM	Name.CurI	Name.MaxCurB	Name.Config

I / O Fx	Name.Counter	Name.MinCurA	Name.Config
Name.In	Name.PulsesPerRev	Name.MaxCurA	
Name.Out		Name.Config	
Name.X0-X7			
Name.Y0-Y7			

Digital Inputs	Miscellaneous	High Side Only
Name	Supply	Name
Name.RealRPM	DVC_Temperature	Name.OpenDisable
Name.PulseTimeout	FreeRunningTimer	Name.Short
Name.PulsesPerRev	MACID	Name.Open
Name.Counter	HC_Coil_Gain_Og*n*	
Name.LOS	LC_Coil_Gain_OG*n*	
	BlinkCode	

About Us

HCT was founded in 1983 as an electronics contract manufacturing company, based upon the founder's extensive background in high-tech manufacturing, including senior manufacturing positions at several Silicon Valley companies. Early in its history, HCT began a systematic migration towards proprietary products. The Company anticipated the need for Electronic Controls for Mobile Equipment and invested more than $2M to develop the DVC™ family of products, which began shipments in 2001.

HCT's proprietary products have been designed specifically for the Mobile Equipment Market, taking into consideration customers' broad functionality needs, demands of the severe operating environments, and other customer requirements unique to this market. The groundbreaking modular architecture of the DVC system allows customers to add functionality, reliability and precision where and when they need it, while preserving all of their investment in prior application development. Industry-standard communication protocols (e.g. CAN bus) and HCT's unique graphical, PC based, programming tool allows our customers to easily implement and maintain Electronic Control Systems via this fully configurable, modular solution.

The patented Intella™ Software System is the heart of the company's DVC product line. It was designed to enable customers and channel partners who are relatively unfamiliar with sophisticated electronic control systems to customize our products in the field with minimal training and/or support. Intella™ Software is a comprehensive development environment for the design, development, testing, modification and support of DVC system applications. Intella™ Application Libraries can be used as templates for application development. All of this functionality is in a system that allows the application designer, application programmer and maintenance programmer to operate within a single application development environment.

The DVC master controller module, with its flexible hardware and software, can run many applications as a single stand-alone module. Combinations of DVC™ modules (10 modules to date, with different functionalities) enable HCT to support a wide range of machine control applications. All DVC™ modules are packaged in small, ruggedized enclosures. Each module is encapsulated to withstand extreme conditions in harsh operating environments. The "hardened" enclosure allows customers to locate the DVC™ modules near the sensors, valves, etc. they will control. This can significantly reduce the amount of cabling in a system and, correspondingly, the cost.

We pride ourselves on producing cost effective devices that are rugged, abuse resistant, and easy to setup and diagnose. We are able to respond quickly to customer requirements due to our in-house engineering and assembly departments. We also provide turnkey manufacturing services for customers that do not require engineering services. Our standard product line includes environmentally hardened hydraulic proportional valve drivers, digital closed and open loop controllers, and user programmable CAN bus controller systems for mobile off-highway applications.

In addition to our standard products, we develop custom products per customer requirements. Full product specification from the customer is welcome, but not required. We often work with the customer to determine the specifications for the product required to solve their problem. Our experience in the industrial and mobile control markets speeds up the product design time and greatly reduces the occurrence of unanticipated problems. Customer support after the sale is one of our strong points.

When you need SOLUTIONS for electronic control of mobile and industrial devices think………. *HIGH COUNTRY TEK*

Limitations on Use of DVC Products

DVC products may not be suitable for use in any of the following applications;

a) Any product which comes under the Federal Highway Safety Act, namely steering or breaking systems for passenger-carrying vehicles or on-highway trucks.
b) Aircraft or space vehicles.
c) Ordnance equipment.
d) Life support equipment.
e) Any end product which, when sold, comes under U.S. Nuclear Regulatory Commission rules and regulations.

With the above in mind XXX needs meet the following criteria:

a) Abide by warranty rules stated above regarding installation and product usage best practices.
b) Provide a liability waiver regarding the use of the parts within a steering system for on-road vehicles

High Country Tek (HCT) does not promote use of any HCT product in any of the above listed applications. HCT does not have any performance assurance programs for testing products in the above listed applications. HCT's products are not designed for these applications and HCT does not warrant, recommend, or specifically approve of its products for these applications.

HCT cannot and does not accept responsibility for or warrant any of its products that have been subjected to improper installation, improper application, negligence, tampering, misuse, abuse, damage, or which have been repaired, altered, or modified outside of HCT's factory delivered condition.

In no event shall HCT be liable to any buyer or end user of any of its products for any incidental, consequential (including, but not limited to, lost profits, business, or other pecuniary loss), indirect, punitive, or special damages arising out of or in any manner incident, relating, pertaining, or attributable to its products, HCT's liability being limited to the value of the product sold or an obligation to repair or replace a defective product.

HCT's liability under this warranty for package/sub-systems/systems hydraulic products shall extend only to repair or replacement, f.o.b. to a location of HCT's determination, of any product determined by HCT's inspection to be defective and warranted and will be at HCT's discretion and is the exclusive remedy of Buyer.

Version 5.4 Features and Enhancements Summary

New Features

o Introduction of the DVC707, DVC722 and DVC741 into the programming environment
o Miscellaneous Programming Tool bug fixes
o Introduction of the DVC725 and DVC745 expansion modules

Manual Index:

DVC System and Software

1.1 Introduction

The DVC710 module is user programmable and able to support a wide range of control applications. The DVC710 is a master controller and its flexible hardware and software allow it to run many hydraulic control applications as a single module. This user guide illustrates the techniques to create and maintain user applications that run on the DVC710 and compatible expansion modules. Instructions on how to use the DVC programming tools are provided along with definitions, programming steps and examples. The DVC710 is programmable using the Programming Tools described in this manual. You simply select the project type you are designing for via a main menu item.

The DVC707 module is like the DVC710 with a reduced I/O and feature set. For example, the DVC707 has only one CAN bus and can do either DVC Devicenet or J1939 but not both at the same time.

NOTE: Programming and operation of the DVC707 is done in the same manner as the DVC710; therefore this manual will discuss programming and operating the DVC710 which includes the full feature set available. When programming or operating a DVC707, not all features discussed may be available.

1.2 The DVC System Overview

The DVC707/DVC710 BIOS software provides high-level data processing to your program regarding the state of each of your system inputs and the actual input/output electrical interfacing to sensors, joysticks, potentiometers and valves is automatically handled for you through relationships defined within the application code.

All DVC Modules are packaged in small rugged enclosures. All connectors are sealed and with the exception of the DVC61, each module is encapsulated to withstand extreme conditions in harsh operating environments. The hardened enclosures allow you to locate the module near the sensors, valves, etc. they will control. This can greatly minimize the amount of cabling in your system and significantly lower your costs.

1.3 DVC700 Family Introduction

The DVC707/710 have enough processing power and input output functionality to support a wide range of hydraulic applications. Should more capability be needed than provided by a single DVC707 / DVC710, multiple interconnected expansion modules may be used and share information via the CAN bus.

Both master controllers support the DVC61 text display or J1939 Graphical displays such as the PV780, PV450 and PV380. The RS232 port on the DVC707 / DVC710 is typically used for loading and monitoring your application program. Four light emitting diodes (LEDs) mounted on top of the DVC707 / DVC710 modules also are useful for monitoring your system's operational status.

The DVC710 has three Universal inputs (programmable to accept the most common sensor inputs), three 0 – 5 Volt analog inputs (programmable for joysticks and potentiometers) and eight digital inputs with Dig_1 able to be configured as a SYSTEM DISABLE input.

The DVC710 has a single +5 volt regulated reference output. This reference can supply up to a total of 250ma of current.

In addition, the DVC710 has 6 High-Side (voltage and current sourcing) outputs and 3 Low Side PWM (pulse width modulation) outputs for bang-bang and proportional valve control. The High-Side outputs provide +POWER (system power typically 12-24 volts) when enabled by your program to the coils used to open or close your valves. The PWM outputs serve two functions for current regulated proportional valve closed loop control. First, they provide the current return path from the negative side of the coil. This current is measured and compared to the desired coil current. Given the difference between the desired and actual current the PWM pulse output duty cycle (i.e. the percent of time current is allowed to flow through the coil) is adjusted to eliminate this error or difference. The internal DVC710 circuitry and BIOS automatically adjust this PWM duty cycle and therefore the effective voltage (and current) seen by the coil. This regulated valve coil current provides a constant valve output (i.e. spool position), which is unchanged by coil resistance, connection length or power supply fluctuations. The High-Side and PWM outputs can be used stand-alone or in conjunction with one another to support the wide combination of valve types you may have in your system. From 3 to 9 valves depending on the valve types can be controlled by a single DVC710. The DVC710 can connect to the CAN Bus via DVC DeviceNet and/or J1939.

The DVC707 has the same basic functionality as the DVC710 but with fewer I/O's. The DVC707 contains three digital/pulse (RPM) Inputs, two standard 0 – 5 Volt analog inputs, three Universal analog inputs programmable for analog or pulse operation and 0 – 5 Volts or 4 – 20 mA. It has two full output groups and two spare High Side Outputs capable of driving two dual coil valves/pumps or up to eight single coil valves etc. The DVC707 has a single CAN bus capable of either DVC DeviceNet or J1939 communications as well as a single 5 Volt reference output that can source up to 250mA to sensors.

1.4   System Configurations

The DVC700 family is designed to control the simplest to most complex machines. Many different configurations are supported including;
DVC707 / DVC710 standalone
DVC707 / DVC710 standalone with optional J1939 Graphical Display or DVC61 Display
DVC707 / DVC710 as an active node on the J1939 CAN Bus
DVC707 / DVC710 with multiple CAN Bus connected DVC Input / Output expansion and Display modules

1.5   CAN Bus

The DVC 710 supports two separate CAN Bus networks. CAN Bus 1 will simultaneously communicate on the J1939 bus and the DVC CAN Bus. This means that the user may use the DVC710 to communicate with both J1939 bus nodes and other DVC Modules simultaneously using the same physical wires. Slave DVC Modules use an 11 bit Identifier rather than the 29 bit identifier that J1939 uses; therefore systems that do not employ a high network load may run both at the same time on the same physical bus without any message degradation saving cabling and labor costs.
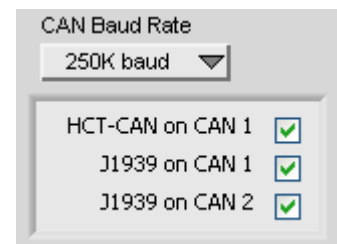
CAN Bus 2 will support J1939 messages only and may be used in tandem or separately from CAN Bus 1 depending on the needs of the application and system. It is suggested that CAN Bus 2 be used for lower priority higher flow traffic such as a J1939 Display or Data Logger. This would allow the user to separate these less critical messages from the systems main ECU system that will be running many high priority messages such as TSC1 etc.

The DVC710's CAN Bus configuration and Baud Rate is selected through the factory Information screen where each feature may be either enabled (Checked) or disabled (Unchecked). Both CAN Bus systems run at the same Baud Rate. Factory Default Baud Rate is 250K baud. The DVC707's CAN Bus can either communicate with other DVC Modules or as a node on the J1939 bus but not both simultaneously.

1.6　How the System Works

This is an overview of the operation of the DVC707 / DVC710 with available expansion modules they can control. While not being an exhaustive discussion of the DVC design and operation, hopefully, this overview will allow you to better understand the Programming Tool and Program Loader Monitor and to see how your system may be programmed and controlled. For more information about expansion modules, please see Chapter 6, Expansion Modules

There are four fundamental concepts that the user needs to understand.
1. The BIOS application is the interface between the user's program application and the modules inputs and outputs.
2. All DVC modules are loaded with a BIOS program that allows it to communicate with ether the application program directly or with the master module that is running the application.
3. Expansion modules must be in communication with the DVC707 / DVC710 master controller in order to perform work.
4. Within the Master Module, the application program is executed by the BIOS using a 10ms cycle (Logic Cycle); I/O Updates (including expansion I/O) are maintained automatically by the BIOS and in parallel with the application.

Each DVC module has an internal program or BIOS to control the module's operation and its communications over the CAN Bus. All modules in the application operate asynchronously with their own internal clock. During boot up, the BIOS configures each modules internal circuits to correspond to the input/output configurations specified by the programmer using the Programming Tool. The BIOS of a DVC expansion module gets the input/output configurations that the user configured using the Programming Tool from the master DVC707 / DVC710 through a series of CAN Bus messages. Once this profile is loaded into the expansion module's memory, the module will configure its I/O then begin reading and writing to the inputs and outputs as commanded by the master modules application program.

During operation, the DVC707 / DVC710 and each of the DVC expansion modules continuously exchange messages between each other over the CAN Bus. Each expansion module sends messages detailing the state of all of its I/O. These messages are received by the DVC707 / DVC710 and stored into its memory. After receipt, the DVC707 / DVC710 controller has a complete status of each of the expansion module's input and output states. Similarly, the DVC707 / DVC710 tests the state of its own inputs and outputs before running the next sequence of logic instructions from the application program. The BIOS will then update all I/O as needed by both sending out messages to expansion modules and controlling its own hardware. Reading I/O, processing application code and setting I/O as needed constitutes one Logic Cycle. This cycle will continue as long as the system is running.

1.7　Closed Loop Control Principles

Closed loop control is simply a process where the DVC will adjust a proportional output in attempt to drive the feedback input to a point where it equals the command (Set Point) input. In a PI system like that used by the DVC family the adjustment amount is a function of the error (set point - feedback) and P and I terms. The P term scales the current adjustment proportional to the error and the I-term scales the correction as a function of the error over time. These corrections are summed. Generally the higher the values of the P and I terms the faster the error will be corrected. Correcting too fast can cause over correction (i.e. overshoot and ringing).   Closed loop applications must be aligned or tuned during actual system operation.

1.8   Programming and Debugging the DVC System

The Windows PC based DVC707 / DVC710 Programming Tool gives you the ability to program the DVC modules to work in a variety of applications without large development costs. Some knowledge of Windows, computer programming and electro-hydraulics is beneficial.

The Programming Tool's main screen shown here is called the Project screen. Every project consists of components. A component can be a, DVC Master Module, an expansion module or a Logic Sequence. At a minimum and by default, a DVC710 (Master) module and an "Always" bubble icon must be defined. To change the project to a DVC707, simply open the Project menu and select the DVC707. As needed, the programmer can add additional physical and programming components by right clicking the mouse on the Project screen and selecting the component type you wish to add. Once selected from the pop up menu, the component will be added to the project as another icon. Simply click on the icon and drag it to the desired location. Double clicking a component icon will open it for programming or configuration.
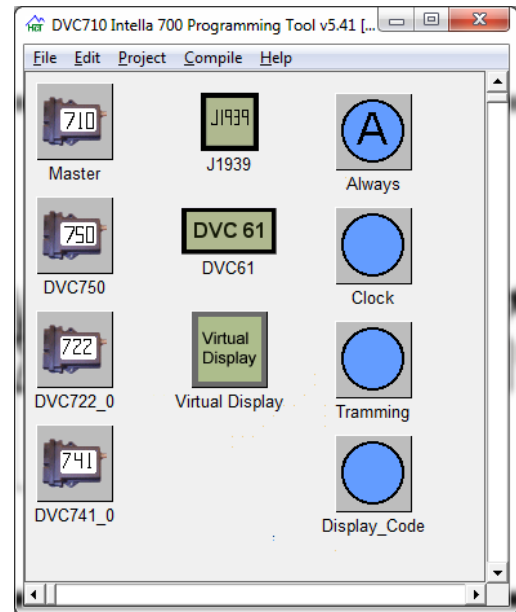
NOTE:
If using more than one master controller on a system, the programmer may initiate communication between the master controllers by either using the DVC Master to DVC Master feature in the programming tool or simply configuring J1939 messages to broadcast their state to the other Master Modules.

Debugging your application is generally done with the assistance of the PC based DVC Program Loader Monitor. This software supports a Virtual Display allowing your application to display variable values and where the code is executing as well as showing you the status of the various inputs and outputs of your system.
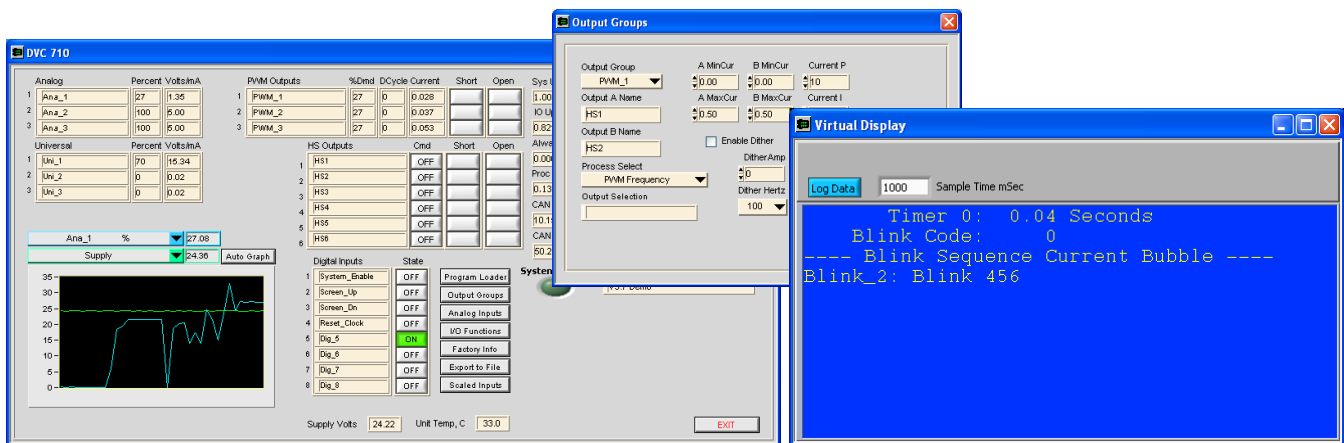
PLM Screens commonly used for debugging application code

1.9 Program Components

The following components are available for use in the version 5.4x Programming Tool. Future additions to the DVC700 family of components will be added as they are released.

All of the following icons are accessible by right clicking on the Project screen. Double click the icons on the Project screen to open up the component specific programming menus or environments. Right mouse click an icon to delete it.

Master Module – Depending on the project type, either the DVC710 or DVC707 Master Icon will be in the project. Open this icon to configure the master module for the project.

Always Bubble – The Always Bubble is automatically added to each new project. This is the programming bubble that contains code to be executed during every "Logic Cycle".

Logic Sequence – Where system operation code is created using state machine like Bubbles and Transitions.

Virtual Display - Where the Program Loader Monitor Virtual Display screens are defined.

Application Simulator – With this Icon is added to the project, when the application is loaded into a Master Module the inputs and outputs of the module are disabled and can be excersized from the Programming Loader Monitor (PLM) on the spplication simulator screens. This allows the programmer do do some bench testing of the application before installing it on a "live" machine. When finished using this feature, simply delete it from the project and recompile the application. When it is reloaded into the DVC707 / DVC710, all I/O will be re-enabled.
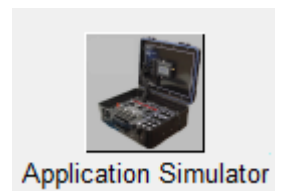
DVC to DVC – Adds four send and four receive messages that may be configured and used to communicate with another DVC707 / DVC710 Master module.

DVC722 – The programmer will add one of these for each DVC722 used in the project. The programmer will open this icon to configure the DVC722 module.
NOTE: Each DVC722 added to the project must have a unique name. i.e. DVC722_0, DVC722_1 or CAB_INPUTS, REMOTE_INPUTS etc.

DVC741 – The programmer will add one of these for each DVC741 used in the project. The programmer will open this icon to configure the DVC741 module.
NOTE: Each DVC741 added to the project must have a unique name. i.e. DVC741_0, DVC741_1 or STARBOARD_OUTPUTS, PORT_OUTPUTS etc.

DVC750 – The programmer will add one of these for each DVC750 used in the project. The programmer will open this icon to configure the DVC750 module.
NOTE: Each DVC750 added to the project must have a unique name. i.e. DVC750_0, DVC750_1 or BOOM_OPERATIONS, MACH_OPERATIONS etc.

J1939 – The J1939 icon is used to add J1939 messages to a project. Open this icon to add and configure individual messages to be used in an application.
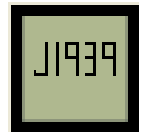
DVC61 – The programmer will add one of these for each DVC61 used in the prohect. Open this icon to define the display screens.
NOTE: Each DVC61 added to the project must have a unique name. i.e. DVC61_0, DVC61_1 or FORWARD_DISPLAY, AFT_DISPLAY etc. On systems with multipal DVC61 displays, all screens may be defined inside one DVC61's icon. Because all variables are global, the other DVC61s in the project can use screens defined inside any DVC61's icon. This is a convient way to keep track of screens and help prevent multapal instanced of the same information which can help save both Program Memory and System Configuration Memory.

1.10 Menus

The Programming tool features a menu across the upper portion of the project screen. The following is a description of the items that compose the menu:

Menu Item	Sub Menu Item	Description
File	New	Create a New Project
	Open	Open an Existing Project (.DVC File)
	Restore BAK File	Restore the last Saved / Compiled Project
	Save	Save Changes to the DVC Project
	Save As	Save DVC Project to a new file
	Load MEM File	Load Program Loader Monitor Modified .MEM File into Project
	Print	Select and Print the DVC Project Code, Bubble Logic diagrams, Module diagrams and Module Data
	Previous File List	Displays the 4 previous files worked on. Clicking on one of these will open the project
	Exit	Shut Down the Programming Tool

Edit	Find	Search the application code for specific text
	Replace All	Search the application code for specific text and replace with new text

Project	DVC710	Set the Application / Project Type to DVC710
	DVC707	Set the Application / Project Type to DVC707

Compile	Make	Save the Project File and Create the DVC Application Output Files

Help	About	Shows the Programming Tool's Version and Part Number

1.11 Projects

A Project contains all of the information about your system in a form that allows you to specify and change your system's hardware and software components over time. The Programming Tool File menu allows you to save your project ("projectname.DVC") into a directory of your choosing and to reopen a previously saved project. When your project is compiled (by selecting "Make") four files are created for the project. These files have the .DVC, .BAK, .MEM and .PGM file extensions.

.DVC – A text file that contains the entire application including all I/O configuration and programming data for the entire project.
.BAK – A text file that contains a backup of the .DVC file. This file is maintained one compiled revision back.
.MEM – A binary that file contains the memory image which describes the applications I/O configuration. The MEM file also includes all defined variables.
.PGM – A binary file that contains the compiled program application.

NOTE: The .PGM and .MEM files are loaded into the DVC707 / DVC710s Flash memory when loading an application into the module. Both files must be located in the same folder/directory in order to successfully load.

1.12 Configuring Inputs and Outputs

The DVC707 / DVC710 and the DVC expansion modules are all configurable using the Programming Tool's graphical interface. Each module's input and output characteristics (voltage range, valve coil currents, etc.) as well as display configuration can be specified. Once the values are specified, the DVC707 / DVC710 BIOS will communicate this information to the hardware.

1.13 Input Output Variables and Programming

The Programming Tool automatically defines a memory location for each configurable Input and Output for every module in the system as well for every defined user variable. The Programming tool will also define memory locations for system variables such as I/O Status, Module Input Voltage, Module Status etc. This memory is allocated regardless of whether or not a specific I/O is used in the system or referenced in the application code. During runtime, all program variables are defined and stored in RAM by the BIOS and is continually updated as required during operation. These variables are accessed by the application program simply by using their variable names such as Dig_1 or Ana_1. For instance to check if a switch has been closed you would write "If (Dig_1 = True)". To check to see if an output has noted an error you could write "if (HS1.open = True) or (HS1.short = True) ".

1.14 Programming Example

The following example illustrates the general constructs used and the screen displays for configuring Inputs and Outputs. The example is a relatively simple Valve Driver application that has been implemented using a DVC710 control module. The code consists of 6 parts namely:
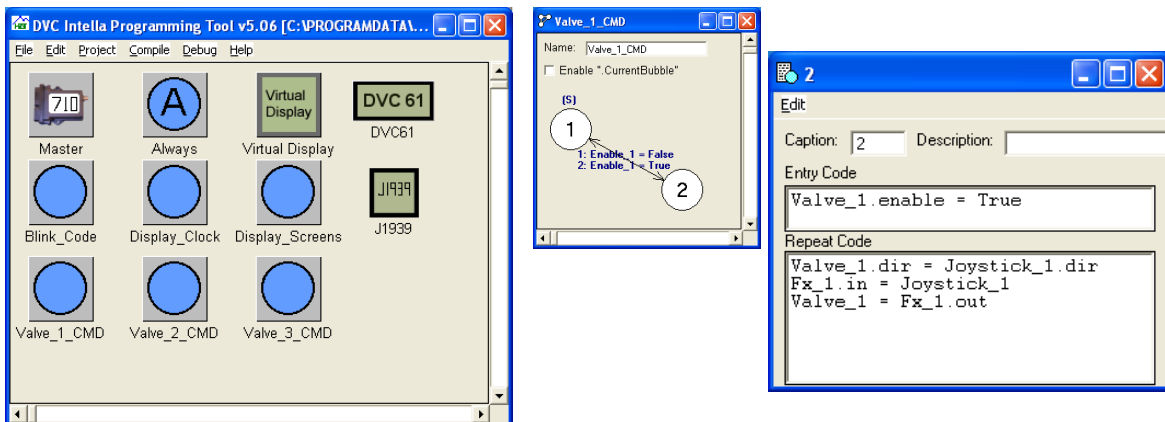Open loop test code for each I/O
A displayed elapsed time clock
The virtual display code for monitoring program execution and variables
Error checking for the status of the wiring connections in the Always code
The unit LED updates
J1939 Bus Monitoring

Intella Programming Tool Main Screen, Logic Sequence and Bubble Logic

1.15   Hints & Tips for code writing

The Intella programming environment allows the programmer to be creative in their programming style. Here are some suggestions as how a program may be structured.

Create machine function flowchart (a.k.a. Sequence of Operations)
This idea forces the development team to understand what the function of the machine will be before the program is written. Using flowchart notation, the development team documents each function of the machine. In the case of a hydraulic log splitter, maybe the first function would be to assure the engine is running, providing pressure for the pump. It is advised that the first flowchart of a project contain very simple steps in the functionality. Later revisions can combine simple functions into more complex functions. Insert as much information in this flowchart as possible. For instance, if there is a pressure relief valve in the machine, set at 1500psi, list that in the flowchart. Once the flowchart is finished, everyone should agree on the machine functionality. This step will help to minimize changes to the program at a later date. This step could take considerable time, but will make the program much simpler to write. Refer to Appendix F for an example.

Separate different functions into different logic Sequences and Bubbles.
It is not incorrect to place all of the program logic in the always code, but it can slow down the program and make it more difficult to troubleshoot. Time critical logic should be contained in the 'always' code. Different functions should be contained in individual Logic Sequences. For example, a pump control's logic can be contained in a bubble by itself. The logic to control the track drive of a bulldozer can be contained in a bubble by itself. These functions may not be as critical as something such as monitoring for a dangerously high pressure in a hydraulic circuit.

Use meaningful variable names.
The limit on variable size is 32 characters. A variable that isn't an input or output device is called an internal variable. An internal variable that is for Valve1 minimum current setting for the program loader monitor could be labeled Valve1MinCurA. This naming convention could be used throughout the program. An EEmem variable for a valve's minimum current setting could be named eeValve1MinCurA. Below are some other examples.

Joystick_1_AI1 – Analog Input # 1 (Variable names in capital letters designate an abbreviation.)
Output_shaft_RPM_UI1 – rpm Universal Input # 1.
EEC1.engine_speed_lo – The low byte of the engine rpm from the J1939 network.

eeDVC61_Contrast – EEmem variable.
ScreenCount – internal user defined variable.

Declare all variables in one location.
Declare all variable names in one area of one bubble. This area would include unsigned integers (Uint), EEmemory variables (EEmem), Timers (timer), constants (const) and others. Declaring all the variables in one location could make it easier to add another variable at a later time without duplicating the variable name and provides for a cleaner, more structured program. Variables can be organized by type or grouped by function depending on the preference of the programmer.

Comment important information into the program.
Information such as Programmer's name, date of program creation, revision history, and any description of something that is difficult to understand is appreciated by anyone offering assistance in troubleshooting the program. Explaining a complex math equation can be beneficial at a later date to refresh the programmer's memory of why a function was built the way it was and to assist in troubleshooting. Comment as much or little as necessary. Comments do not contribute to the compiled program (.PGM) file size. Remember to add a colon (') before starting comments. The Intella software interprets the (') that the information to the right is not to be compiled.

2 Software

2.1 System Requirements

Windows XP, Windows 7 / 8, or Windows Ultimate, (32 or 64bit)
40 megabytes of disk space to support a complete system install
PC with Serial Port - RS232 or USB port
For USB ports you need a USB to RS232 converter (i.e. Dongle)[1]
DVC707 / DVC710 Master Control module
DVC serial cable

2.2 Installation

Installation of the Intella™ Software Suite, installs both the Programming Tool and the Program Loader Monitor that are needed to support the DVC707 / DVC710.

To install the Intella™ Software Suite, close all program applications. Insert the Intella™ Software Suite CD in the CD-ROM drive, and wait for the installer program to execute. The installer will guide you through the installation procedure.

NOTE: Install any third party software i.e. USB to RS232 adaptor drivers before installing cables and hardware.

2.3 Software Overview

The Intella™ Software Suite includes two applications, the "Programming Tool" and the "Program Loader Monitor". These applications provide the means to configure, design, create, load, and monitor the user application. The Programming tool is used to configure the inputs and outputs and program the control logic for an application. The Program Loader Monitor is used to download the user application program files to the DVC Master Controller. It also performs real time monitoring of your system inputs/outputs. In the PLM, you can modify some input/output configuration settings (i.e. like analog ramp rates) and save them to a file that can be imported into the Programming Tool application. The Program Loader Monitor also allows the user to view and modify EEmem variables. This can be a convenient way to allow an operator to change a maximum pressure setting or input a customer job number while the system is running.
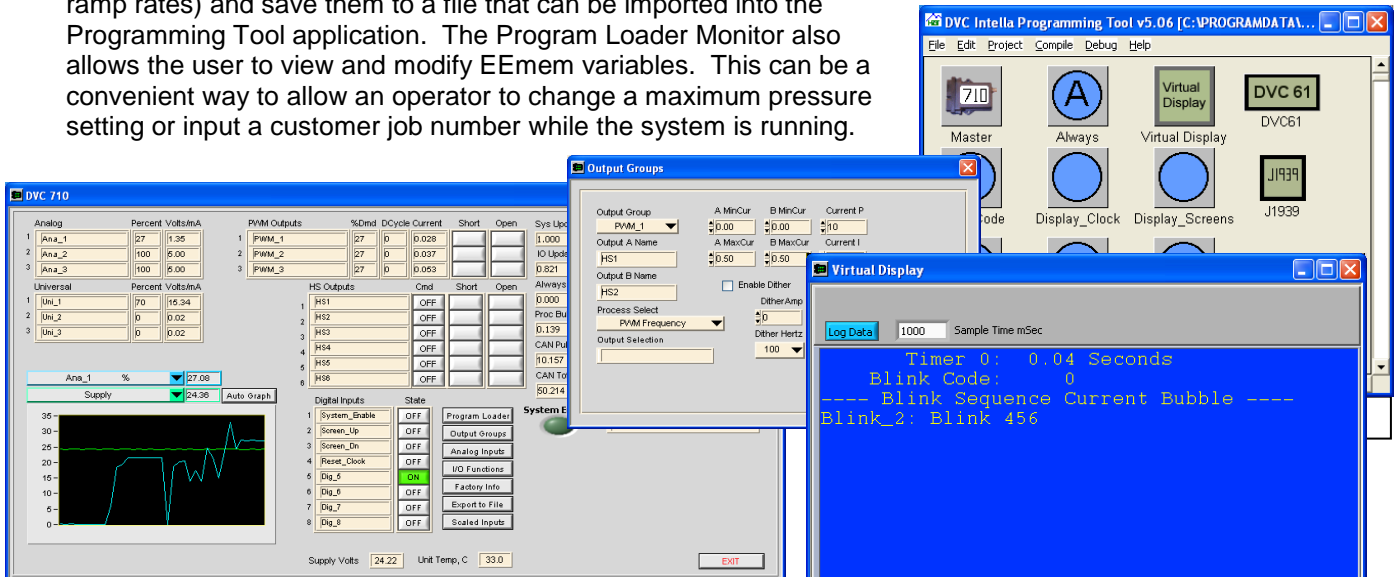
Intella™ Programming Tool and Program Loader Monitor Screens

---

[1] A list of HCT recommended USB to RS232 adaptors may be found on our website at, http://hctcontrols.com

3 Programming the DVC Family
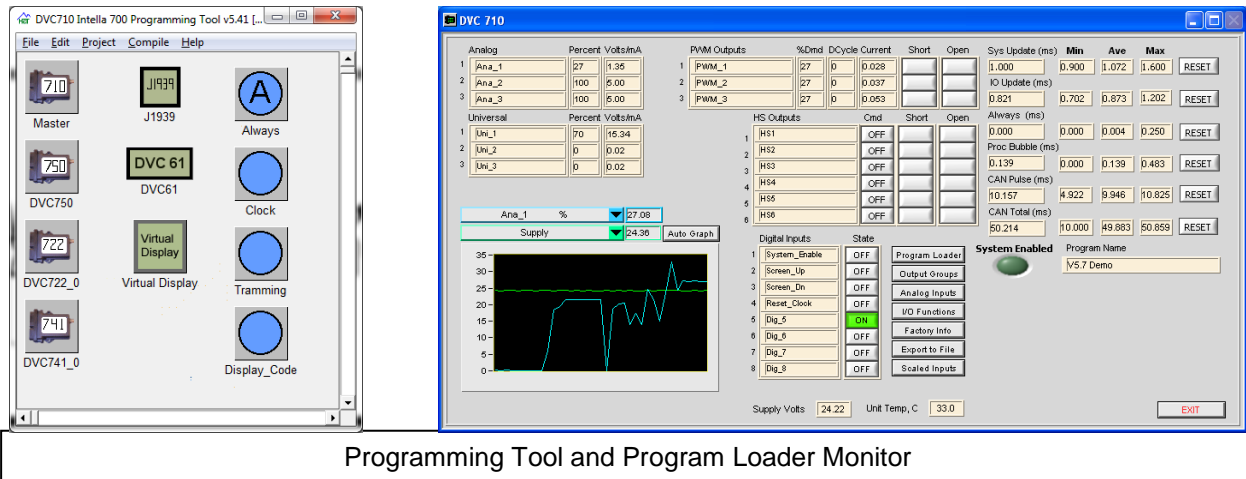
Programming Tool and Program Loader Monitor

The DVC modules and the Intella™ Programming Tool / Program Loader Monitor system have been designed with an effort to decrease the load on the programmer by reducing the amount of code required to successfully set up and control hardware. With this in mind, many functions that involve hardware configuration, output wiring, output validation (open/short testing) and module safety such as supply voltage and temperature monitoring is done automatically without the need for the programmer to write all the subroutines required to perform these tasks.

The "Programming Tool" is used to create your application for the DVC707/710. The "Program Loader Monitor" is used to load the user application into the DVC707/710 module and monitor the inputs and outputs in real-time as your application executes. Both programs are located in the Windows Start Menu under the c:\Program Files\HCT Products. In order to create a user application, the following steps generally should be followed:

a) Determine the project needs. In the design process compile a total list of digital inputs, pulse inputs, PWM outputs etc. An accurate I/O count is needed to determine the number and type of DVC modules needed for the project. Also, include display needs and allow for expansion with spare I/O when possible.

b) Define your external system components and how they will be connected to the DVC system.

c) Start the Intella Programming Tool or from the File menu select New or press CNTL + N, then from the File Menu, select Save As and name your project, i.e. hydraulic log splitter.

d) Add any planned expansion modules displays etc. then configure the DVC707/710 and expansion module's inputs and outputs used in the project. With the mouse on the screen, right click to choose the expansion module. Insert the expansion module, set the modules unique name and Mac ID#. Assign names to the I/O.

e) Create a flow diagram of machine function (i.e. Sequence of Operations) refer to Appendix F for additional information. Once the Sequence of Operations (SoO) is complete, it can be used to format the application and logic flow during programming. Refer to section 5 for programming examples. Refer to Appendix A for compiler keywords. Program the application as needed.

f)   Compile the program.

g)   Load the compiled program files into the DVC707/710.

h)   Run your system and debug your application.

3.1   Compiling Your Program to Create the Output Files

After configuring the Master Modules inputs and outputs and creating the program code, press CNTL + M or select the Compile menu item in the Project window and select Make to create the output files.   If there are no errors, the Programming Tool will create three new files. The two primary files needed for the next step are projectname.PGM and projectname.MEM. These are the two files that will be loaded into the DVC707 / DVC710. A third file projectname.BAK is generated this is the backup file.

NOTE: If the compiler detects an error during compilation, an error display will pop up and the line in your program with the error will be highlighted.

3.2   Loading DVC Files

Projectname.DVC files contain all the information saved in a project. The Programming Tool reads these files to open previously saved projects. Once open you can make modifications to the Input / Outputs, control code and system configuration. To open a project, click on the File menu item in the Project window and select Open. Finally, select the appropriate "projectname.DVC" file.

3.3   Saving DVC Files

To save your project press CNTL + S or click on the File menu item in the Programming Tool project window and then select the Save menu item. This saves the project file under the current filename. To save a project with a different name click on File and select "Save as" on the menu selection. Type a new filename and click save to create your new .DVC project file.

When naming files, it is recommended that a version number or letter be included in the file name. This way when major changes are made, the user can rename the file with the new version information before saving and there will be a better history of the project development available to the user for reference later on.

NOTE: When preforming a Save As operation, if an existing project name is selected; the existing file will be overwritten with the new DVC project information and the old project information will be lost.

3.4   Restoring DVC Files

When compiling or saving a project file, the Programming Tool re-names the .DVC file to a .BAK file then creates a new .DVC file for the project. This allows the user a one level history of the project that may be restored to the project if needed. When Restore BAK file is selected from the File menu, the Programming Tool automatically loads the last backup made, if one exists, for the current open project.

3.5   Loading PGM and MEM files

After a DVC project has been successfully compiled, it is ready to be loaded into the DVC master control module (DVC707 / DVC710).   During compilation the Programming Tool creates two files named projectname.PGM and projectname.MEM. The .PGM and .MEM files contain the users' application code in an executable format.   The .PGM file contains the compiled application code and is referenced by the Program Loader Monitor when you load the application into the DVC707/710. The .MEM file contains the

systems configuration information for all of the projects inputs and outputs and is loaded along with the .PGM file by the Program Loader Monitor

NOTE: .MEM files contain all of the DVC707 / DVC710's physical information as well as any expansion modules configuration information Function Curve set points, etc. If changes are made to the DVC configuration with the Program Loader Monitor, you can update the DVC application program with the new configuration data by doing the following; Using the Program Loader Monitor, save a new .MEM file by clicking on "Export to File". Then using the Programming Tool, open the current project and under the "File" menu select "Load Mem File". Select the .MEM file saved for the project and click Open. The program will automatically update all of the DVC physical information.

3.6   Selecting or Changing Your Project Type

Using the Project Menu of the Project window you can select from amongst two project types. These are:
- DVC710
- DVC707

NOTE: Only features available on the currently selected Master module will appear on the configuration screen. For instance, the configuration screen for the DVC707 will only show three Digital Inputs where the configuration screen for the DVC710 will show eight.

3.7   Master Module Configuration Screen

The DVC hardware and firmware features are very flexible and support many input and output configurations. To configure the module, access each of the features by clicking on their associated buttons. Click on a button to access the configuration options for that feature.

The following subsections give the definitions of each of the fields accessible in the DVC710 configuration window.

3.8   Program Name and Passwords

Program Name:
When the application executes, the Program Loader Monitor will display the program name.
Range: 16 Characters.   No spaces are allowed.

DVC710 Configuration and Setup Screen

Send Password:
Level 1 Password – Allows access to EEmem and other I/O settings from the Program Loader Monitor, will not allow the user to load programs or BIOS.
Range: 16 Characters.

NOTE: If a level 1 password is defined and a level 2 or level 3 password is <u>not</u> defined, then if the project were loaded into a module, that module would be effectively "Locked" with that application because it will not be able to be loaded with another application without a factory level password from HCT.
App Password:
Level 2 Password – Allows all Password Level 1 access plus allows the user to load applications. Restricts the user from loading a new BIOS.
Range: 16 Characters.
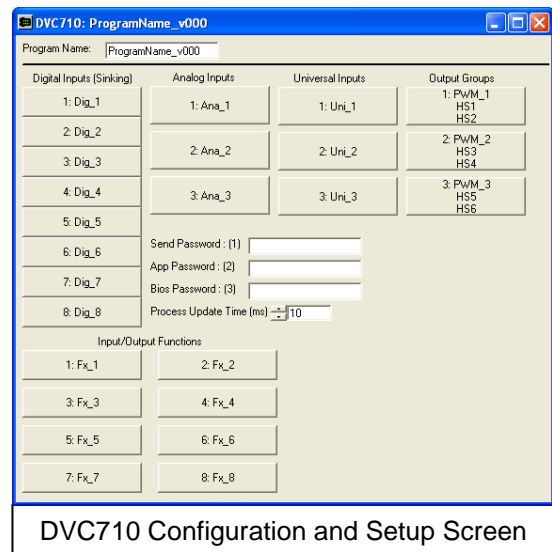
Bios Password:
Level 3 Password – Allows all Password Level 1 & 2 access plus allows the user to load a new BIOS and Change Can Bus settings.
Range: 16 Characters.

NOTE: If passwords are used, a Password Level of 0 in the Program Loader Monitor (i.e. no password entered or incorrect password) will not allow access to any changes to settings for the system.

3.9   DVC Program Loader Monitor Password Implementation

The password scheme is implemented to protect customers from software vandalism or unskilled users. First, the passwords are defined using the Programming Tool and are downloaded into the DVC707/710 when the project files are loaded. Next, the Program Loader Monitor asks you to enter a password for the level of access you wish to have for the run time environment. The Program Loader Monitor has 3 levels of password protection. The level of the password entered in the Program Loader Monitor determines your access and ability to issue commands. The three levels are 1: Send Changes, 2: Load Applications, 3: Load BIOS. Higher numeric levels include all of the abilities of the lower levels. If no password is entered when the Program Loader Monitor is run then default access is given to the user to view the status of the DVC's, factory information, EE memory (non-volatile memory where program variables can be stored in the event of power failure) and DVC expansion modules. However, if all password fields are left blank in the Programming Tool, level 2 access is given by the Program Loader Monitor as a default.

3.10   Process Update Time

The Process Update Time is the amount of time that the application code will take to execute one logic cycle (Always Code, Logic bubble Code, Transition Code and update the I/O).
 Range: 1mS to 20mS (it is recommended that this not be set less than 2mS)

3.11   Programming Tool Debug Feature

When using the Programming tool, selecting anywhere on the blank space of the main window and typing "debug" will provide the user with a new pull down menu on the title bar with the heading Debug. After compiling an application, the programming tool will produce several more files that may be accessed through this pull down menu. The application may not be edited through these files but they can be a useful tool in troubleshooting an application. The extended files are listed below.

- PROGRAM_NAME.HDR
- PROGRAM_NAME.ASM
- PROGRAM_NAME.MAP
- PROGRAM_NAME.OBJ
- PROGRAM_NAME.SRC
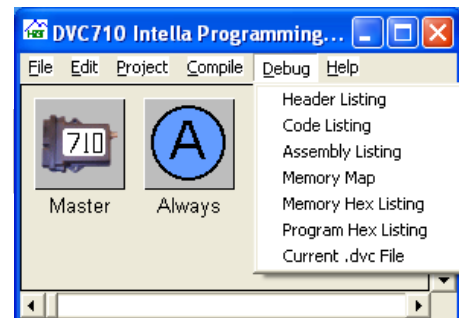
File Types Created by the Intella 700 Programming Tool
The Following is a definition or explanation of the file types created by the Intella 700 Programming tool for both normal and debugging operations.

DVC file - .DVC (Normal File Set)
The .DVC file is the application itself un-compiled in standard text. This is the file that the compiler uses to store bubble code in.

Program file - .PGM (Normal File Set)
The .PGM file is the compiled application file that is loaded into the DVC along with information from the .MEM file.

Memory file - .MEM (Normal File Set)
The .MEM file stores settings for the inputs and outputs etc.

Backup file - .BAK (Normal File Set)
The .BAK file stores the last .DVC file unedited each time the user compiles the code. If the user wanted to delete all changes from the current compiled code back to the last time the code was compiled the user may close the file, delete the current .DVC file and rename the .BAK file to .DVC and reopen the file. The file will open to the same condition as it was before it was compiled the last time.

Header Listing - .HDR (Extended "Debug" File Set)
The Header Listing file can provide the user with a quick way to look up variables that may be directly accessed and manipulated from within the application code.

Code Listing & Assembly Listing - .ASM (Extended "Debug" File Set)
The Code and assembly listings are two ways of looking at the current saved file and are derived from the current file set.

Memory Map & Memory Hex Listing - .MAP (Extended "Debug" File Set)
The Memory map is a listing of where each variable is stored n memory. The Hex listing is a Hexadecimal representation of the Memory Map.

Program Hex Listing (Extended "Debug" File Set)
The Program Hex Listing displays the current .DVC file in hexadecimal format.

3.12 Digital Inputs

Eight digital inputs are provided in the DVC710 controller, three on the DVC707. The user can use the Program Loader Monitor to monitor the state of the digital inputs on the DVC707 / DVC710.

Digital inputs are set by the opening or closing of a switch during system operation. The activation of a switch presents a voltage waveform to a DVC's digital input pin. The DVC hardware and software interpret the waveform and convert it to a true or false value for the application program. The false or true value (0 or not zero numeric values respectively) is passed to the application program via a program variable with the name of the input. The application program control logic then determines what to do given this input state.
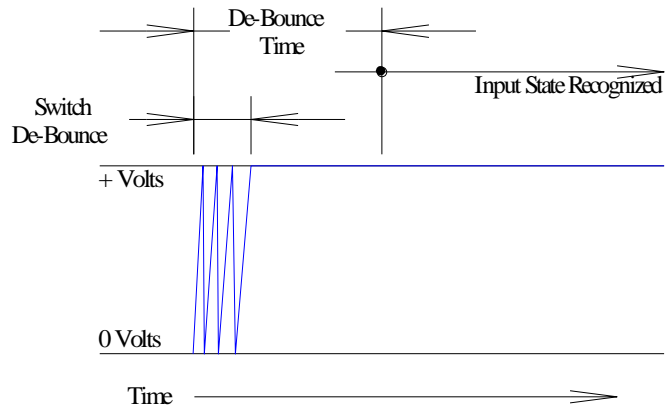
Digital Input Behavior, Voltage In vs. Time

The DVC's Digital Inputs hardware provides a path to GND through a 32KΩ resistor and must be pulled high to change states. If the user wishes to operate the input by using a switch to ground instead of power, an external pull up resistor may be used with the reference output as shown below. Each digital input may be configured as either Active High or Active Low. When configured as Active High, the input will report as False

Sinking (Externally Powered Switch)   Sourcing (DVC's 5V Ref Powered Switch)
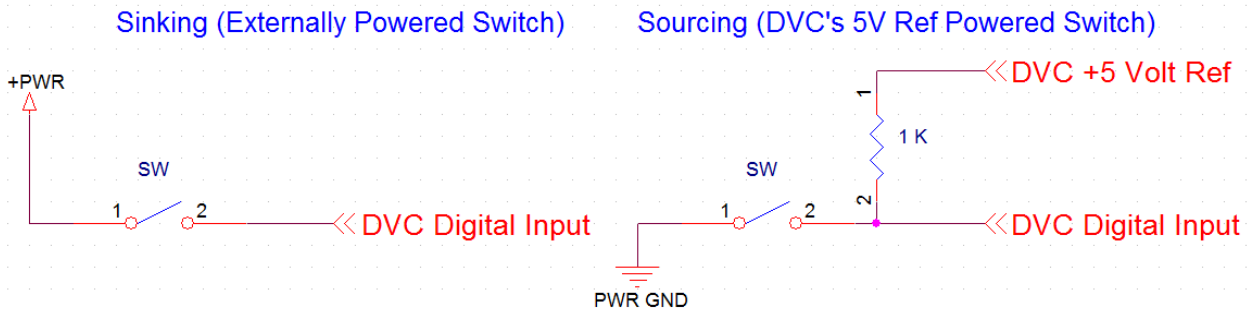
when open or at zero Volts and True when high or at >2.5 Volts. When configured as Active Low, the input will report as True when open or at zero Volts and False when high or at >2.5 Volts.
NOTE: The DVC circuitry senses the voltage change at the edge of the waveform and if the transition state is the same after the de-bounce time interval then the new state is considered valid the application will respond accordingly.

To configure a digital input, click one of the Digital Input buttons on the DVC screen. The Name field's value is the way this input will be referenced in your user application. It is a good idea to name the input to reflect its function.

Name
Name used in the bubble logic code to access this digital inputs state and its associated properties.
Range: 16 Characters with no spaces. Valid characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number. Compiler
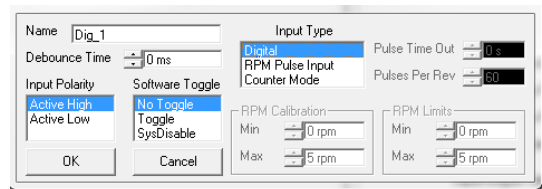
Keywords or other Names already in use are not valid. A valid example is "Boom_Extend".

De-bounce Time
The number of milliseconds the system will wait after a change in voltage at the input before accepting a change in input state.
Range: 0 to 9990ms in 10ms Increments

Input Type (DVC707 only)
On the DVC707, Digital inputs may be used as standard Digital Inputs, RPM Pulse Inputs or Counter Inputs. When selected as RPM or Counter Inputs, be sure to configure the pulse, timeout and limits settings. See Universal Inputs for more information about configuring Counter and RPM inputs.

Polarity
Active High is considered True, Active or On when > 2.5 Volts is sensed at the input pin.
Active Low is considered True, Active or On when < 2.5 Volts is sensed at the input pin.
Range: Active High, and Active Low

Software Toggle (Software Latch)
In Toggle Mode, the rising or falling of the digital input (with respect to De-Bounce and polarity Active High or Low) will reverse the state of the variable with each valid input pulse, latching the input variable at each occurrence. In No Toggle Mode, the input responds to the voltage level at the input at all times again with respect to De-Bounce time.
Range: Toggle, No Toggle

		Toggle	No Toggle
Polarity	Active High	Variable changes states when input goes from Ground to >2.5 Volts	Variable is true when input is >2.5 Volts
	Active Low	Variable changes states when input goes from > 2.5 Volts to Ground	Variable is true when input is ground

Digital Input Code Examples:

Code	Comments
If (Dig_1 = true) Then	if logic test True or False based on the state of the input

HS1 = Dig_1	Sets an output to the state of the Input
	This is the same as the following statement;
	
	if (dig_1 = true) then
	    hs1 = true
	else
	    hs1 = false
	end if

PWM_1.Dir = Dig_1	Set direction of a dual coil based on the state of the input
Dig_1 = False	Set the state of an Input to False (Toggle mode Only)

3.13   Analog Inputs

The Range for Analog Inputs is 0 – 5 Volts. The DVC710 has three Analog inputs while the DVC707 has two. Analog inputs return the value of the voltage at an input pin to the application as a percentage of the calibrated voltage range represented by a ten-bit value (0 (0%) -1023 (100%) decimal). Resolution is to the nearest tenth of a percent.

Each Analog input can be configured as a digital input by selecting the Active Low Digital Input checkbox in the lower right hand corner of the configuration window. When one of these is selected the Ana_1 name will be set to true when the voltage input < 2.5 volts.
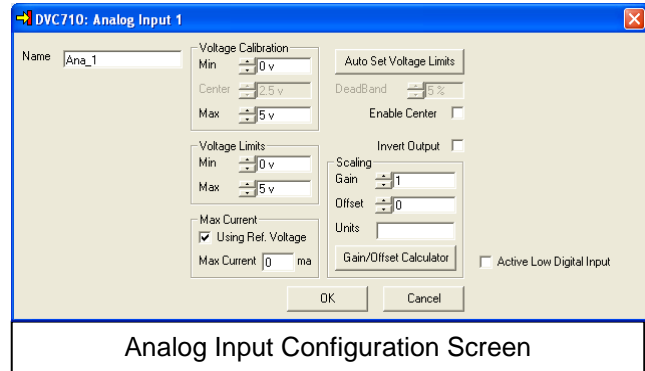
Analog Input Configuration Screen

NOTE:
Analog Inputs are internally pulled to +5V through a 1MΩ resistor. An external 2KΩ resistor between the input pin and ground may be used to pull down the input pin if desired.
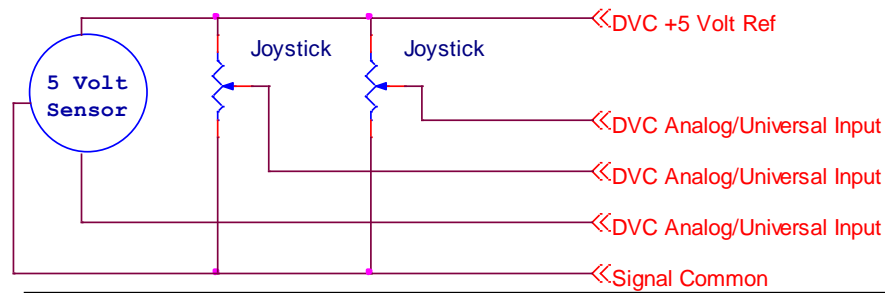
The DVC707 and the DVC710 have one common reference output that can be used by several sensors, potentiometers etc. as long as the total current load of 250mA is not exceeded.

Example of Multiple Sensors using 5 Volt Reference

The configuration of the analog input is done through the Programming Tool's Analog Input setup screen shown above. Some input fields may be disabled depending on the boxes checked (i.e. Enable Center). First, give the input a name that allows you to reference the specific input and its properties in your application. If the input requires a center, check the Center Enable Checkbox, and enter the direction names. When Enable Center is checked you may either use the Min Volts to Center Volts and Center Volts to Max Volts names in your application or simply refer to the "inputname.dir" variable in the application. Calibrate the input with a voltage meter or the Program Loader Monitor and fill in the Voltage Calibration Min, Max, and Center. Enter the Voltage limits. These values are used to detect an out of range condition on the input. To automatically set the voltage limits click on the Auto Set Voltage Limits button. The Invert Output selection will make the program variable value equal to 100% at MIN Volts and 0% at MAX Volts.

The following subsections give the definition as well as an overview of each of the fields in the Analog Input screen:

Name:

Variable name used in the application program to access this input's value as a percentage of the voltage range or true/false for a digital input and its associated properties.

Range: 16 Characters with no spaces. Valid characters are A-Z, a-z, 0-9, and "_".

Rules: The first character cannot be a number. Compiler keywords or other names already in use are not valid. A valid example is "Steering".

Min Volts to Center Volts:

Name: Access word for the input between 0 Volts and the Center Voltage

Range: 16 Characters with no spaces. Usable characters are A-Z, a-z, 0-9, and "_".

Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid. A valid example is "Steer Left".

Center volts to Max volts

Name: Access word for the input between Center Voltage and the Max Voltage

Range: 16 Characters with no spaces. Usable characters are A-Z, a-z, 0-9, and "_".

Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid. A valid example is "Steer_Right".

Voltage Calibration

Calibrate the input to the expected usable range of the sensor driving the input. The applications response to the input voltage will be scaled between the Min and Max voltages and the input will return 0% to 100% (0 – 1023) to the application.

Example of Center enabled 0 - 10 Volt Input

Example of 0 - 10 Volt Input

Min: The minimum voltage. Reports 0 or 0% to the application when center enabled is not checked and 1023 or 100% when center enabled is true except when invert output is selected.
Range: 0 to 4.99v must be less than Center and/or Max Volts

Center: The center voltage. Available when center enabled is checked. Reports 0 or 0% to the application except when invert output is selected and is the center point of the Deadband.
Range: .01 to 4.99v must be between Min and Max Volts

Max: The maximum voltage. Reports 1023 or 100% to the application except when invert output is selected.
Range: .01 to 5.00v must be greater than Center and/or Min Volts
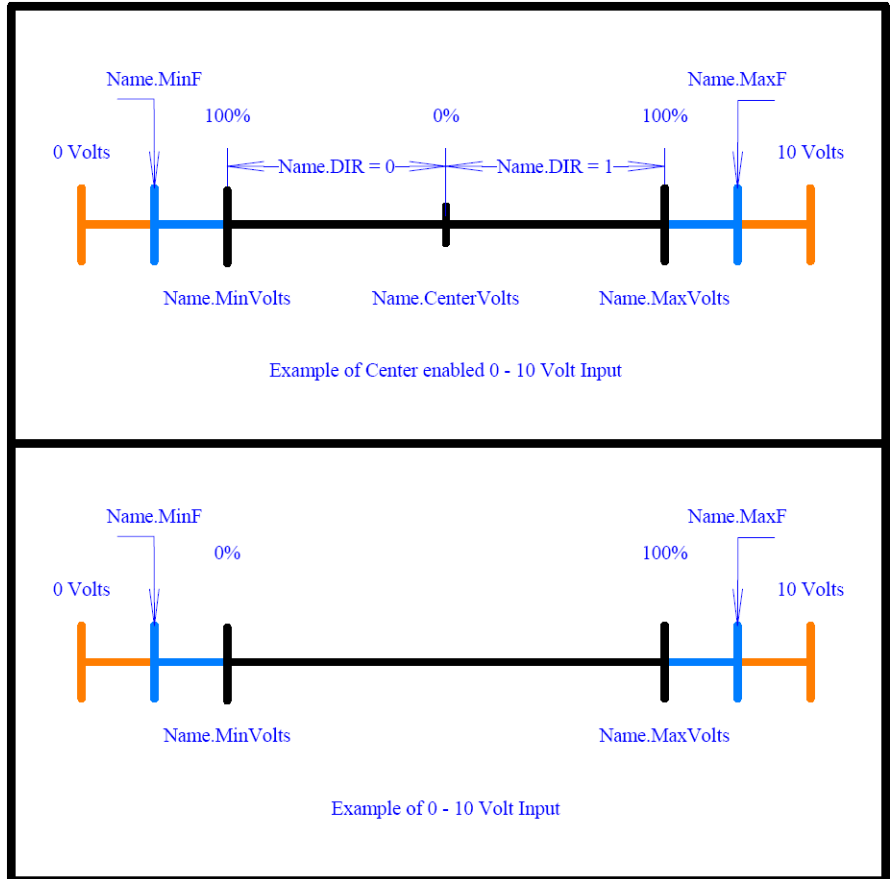
Voltage Limits
These values are used by the system to trigger an error flag on the input. The Voltage Limits may be used to detect an open or shorted input wire or overdrive conditions. The condition is reported to the application setting the Name.MinF and Name.MaxF variables. These variables can be reset through the program code.

Min: If the input voltage is less than this set point, Then "Name.MinF" is set to true.
Range: 0 to 4.99v
NOTE: If the voltage goes out of range, the user must reset Name.MinF variable through application code to false.

Max: If the input voltage is more than this set point, Then "Name.MaxF" is set to true.
Range: .01 to 5.00v
NOTE: If the voltage goes out of range, the user must reset Name.MaxF variable through application code to false.

Max Current Tool
The Max Current tool is a reference tool that keeps track of the total current expected to be used by the Reference Output. It is available on all analog and Universal input screens. The user may enter the expected current draw for the sensor being used when the sensor is to be driven by the Reference Output. If the sum total of all the entered currents from all the Analog and Universal Inputs exceeds 250mA, a warning will be displayed.

Using Ref. Voltage: If selected, the Max Current entered will be added to the total expected load for the Reference Output.
Range: Selected / De-Selected

Max Currnet: The amount of current that is expected to be drawn by the sensor used on this input.
Range: 0 - 250mA

Invert Output
When Invert Output is checked, the returned percentage values of the voltage range will be inverted. For example, what would normally be 100% becomes 0% and 0% becomes 100%
Range: Checked/Unchecked

Enable Center
When unchecked the percentage value returned to the application would be directly proportional to Min - Max with Min equal to 0% and Max equal to 100%. When checked, the center voltage is 0% and the Min and Max Volts will be 100% with respect to the Invert Output feature state.
Range: Checked/Unchecked

Deadband %
The Deadband percentage specifies the range of voltage above and below the center volts setting that is effectively center. The input will report 0% when within this range with respect to the Invert Output feature state. This can be useful for eliminating "creeping" when the joystick is centered.

Auto Set Voltage Limits
The Auto Set Voltage Limits will set the Voltage Limits based on the Voltage Calibration Settings. The voltage limits will be set to one-half of the difference between the Reference and Voltage Calibration values for both the min and max values.

Scaling

This feature is a reference tool that may be used to display the Analog / Universal Inputs as real units (i.e. PSI, GPM, Meters etc.) on the PLM in real time. This allows the operator to read the actual system performance without having to calculate percentages or voltage levels while working on the system. Use the Gain and Offset Calculator to obtain and insert the values for Gain and Offset. Enter an abbreviation for the units desired (i.e. PSI, GPM) for the scaled output
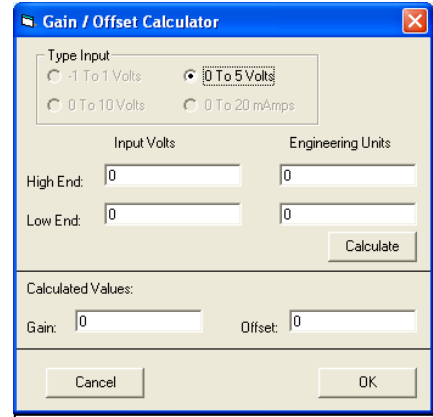
Gain / Offset Calculator

When analog input voltage values are to be displayed by the Program Load Monitor this facility will automatically scale the value displayed according to a linear equation of the form; $y = Gain * x + Offset$

Where x is the analog input's value and y is the scaled value. This is convenient to convert a sensor voltage to actual units like PSI.

The Gain / Offset Calculator has been designed to help the application developer calculate values for Gain and Offset.

To calculate the Gain and Offset values, open the Gain / Offset calculator and enter the values for voltage and engineering units (scaled units), then click Calculate. Copy the calculated Gain and Offset values from the calculator to the analog input screen and save you r project.

Gain and Offset Calculator

Calculator Variables:

High End
Input Volts = Maximum Sensor Voltage
Engineering Units = Maximum Sensor Value (i.e.5000 for 5000psi)
Low End
Input Volts = Minimum Sensor Voltage
Engineering Units = Minimum Sensor Value (i.e.500 for 500psi

Analog Input Sample Code:

Code	Comments
If (Ana_1 > 5%) Then	If Input % is greater than 5%
PWM_1 = Ana_1	Sets the PWM_1 outputs demand to follow the Analog Input
PWM_1.Dir = Ana_1.Dir	Set direction of a dual coil based on input (if centered enabled)
If (Ana_1.MaxF) then	Test if Maximum Volts threshold reached
Ana_1.MaxF = 0	Clear /Reset condition or flag (retry)

3.14 Universal Inputs

There are three Universal Inputs on both the DVC707 and the DVC710. These inputs are programmable to accept the most common sensor outputs.

Four types of inputs are supported and are selectable for each of the inputs. They are Analog Input, RPM Pulse Input, Counter Mode input and PWM Duty Cycle types. A fifth type of input is Quadrature and using this requires two Universal inputs to be used. These are inputs 2 and 3. Quadrature is a method of determining speed and direction using two pulse inputs.
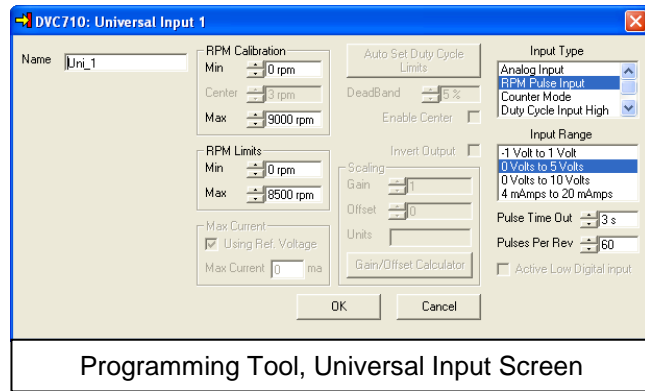
Programming Tool, Universal Input Screen

Universal Inputs support four voltage or current ranges they are: ± 1Volt, 0 to +5Volts, 0 to +10Volts or 4mA to 20mA.

To identify a Universal Input in your application program, fill out the name field or use the default. Each universal input needs at a minimum its input type and input range to be selected.

NOTES:
The Universal Input configuration window will display certain fields while deactivating others based on the Input Type and other options selected.

The DVC707 only supports 0 – 5 Volt and 0 – 20mA operation and does not support the Duty Cycle Input feature.

Input Fields
For Universal Inputs, the setup process is similar to the Analog Input setup. Refer to the previous section Analog Inputs of this manual for a complete description of the analog input features not listed here.

RPM Pulse Inputs
For RPM pulse inputs, specify the RPM Limits (min and max), and the RPM Calibration parameters (min and max). RPM Calibration values set the 0 and 100% variable return values. The RPM Limits set error variables when these values are met or exceeded. Because 0 RPM can never be counted, the Pulse Time Out field is the number of seconds the system will wait until the RPM variable is set to 0 by the BIOS and the pulse time out flag "*name*.LOS" is set. Pulses Per Rev is the number of pulses that the module will count for each revolution. The DVC707 / DVC710 can count pulses to a total limit of 24 kHz combined on all three inputs simultaneously.

Counter Mode Inputs
For Counter Mode pulse inputs, setup the min and max counts under Count Limits. The output value will be a percent of the min to max difference. The counter value may be read or set by the application. The counter is incremented on every falling edge of the pulse input. When the count reaches the max value it remains at that value until set to a new value by the application program.

Quadrature Inputs
Quadrature is a method of determining Direction and/or Position using two pulse inputs. For a Quadrature pulse input, setup the min and max count under Count Limits. Quadrature mode can only be selected with Universal input #2 and will automatically use Universal input #3 for the second pulse train. The output will be the percentage of the min to max count. The counter value may be read or set by the application. The counter

will be incremented or decremented on every rising and falling edge of both pulse trains based upon the phase of the two inputs. Direction is determined by the phase of the pulses going up and down.

The following gives the definition as well as an overview of each of the fields in the Universal Input screen not covered in the Analog Inputs section.

Input Type
Contains all the configuration selections for the universal input
Range: Analog Input, RPM Pulse Input, Counter Mode, Quadrature Mode and Duty Cycle Modes

Input Range
Select the range of the input voltage or current for the Input
Range: 0 Volts to 5 Volts, 0 Volts to 10 Volts, ±1 Volt, and 4 mA to 20 mA

Pulses Per Rev
The amount of pulses the processor will count for one revolution
Range: 1 to 9999

Pulse Time Out / Loss of Signal (.LOS)
The amount of time it takes before turning RPM to 0 in RPM mode and/or setting the *name*.LOS flag if no pulses are detected when in RPM, Counter and Duty Cycle modes.
Range: 0 to 10.0 seconds

Count Limits
The max and min value for the pulse counter.
Min: 0 to 65535
Max: 0 to 65535

Duty Cycle Input High / Low
The Duty Cycle Input High / Duty Cycle Input Low feature of the Universal inputs is intended to read the percentage of Duty Cycle Output from either an ECM or other sensor that uses a Voltage Duty Cycle output to report its information. The advantage of this type of input is that changes in system voltage do not affect the state of the sensor. The State of the input is reported to the application code as a 10 bit number and reported to the PLM as a percentage. The input will automatically read PWM signals from 30 Hz to 1.5 KHz. Duty Cycle High assumes that a high voltage with no pulses would be 100% and Duty Cycle Low assumes that a ground at the input would be 100%. When using the Duty Cycle Input feature, the application should monitor the name.LOS flag to ensure that the signal received by the external sensor is valid and current. During the event of an invalid signal, the application programmer may then decide what action to take.
Range: 5% to 95%

Universal Input Code Sample

Code	Comments
If (Uni_1 > 5%) Then	Test to see If Input % is greater than 5%
PWM_1 = Uni_1	Sets the PWM_1 outputs demand to follow the Universal Input
PWM_1.Dir = Uni_1.Dir	Set direction of a dual coil based Input (if center enabled)
If(Uni_1.MaxF) then	Test Maximum Volts threshold reached
If (Uni_1.RealRPM > 500) then	Test Actual RPM > 500 RPM
Uni_1.MaxVolt = Uni_1.RawVolts	Set voltage max to volts seen (Calibration Example)
If (Uni_1.LOS) then	Test loss of signal (pulse input only)
Uni_1.Counter = 512	Set counter to value (Counter or Quadrature type only)

3.15 Output Groups

The DVC707 and DVC710 each have three Output Groups. The DVC707 has one less PWM output than the DVC710. Each output group contains two discreet High-Side outputs and one proportional Low Side (PWM) output. Proportional current to a valve/pump etc. is accomplished by either one of the High Side Outputs or the user providing system voltage to the valve/pump and wiring the other side of the valve/pump to the PWM output. The DVC will adjust the Duty Cycle of the path to ground in order to regulate current through the valve/pump.

DVC controllers can control two kinds of devices. The first type is discreet or "Bang – Bang" The second type is Proportional (PWM% controlled) devices that are controlled by regulating current through the coil using PWM.

The output groups are designed to give the user a great deal of flexibility. The software gives the user the ability to control the voltage (High-Side) to the positive side of the coil and control the PWM current sinking capability (PWM OUT) from the negative side of the coil. Refer to Appendix E for a more in depth discussion of PWM, Dither and the PID technique used to regulate coil current.
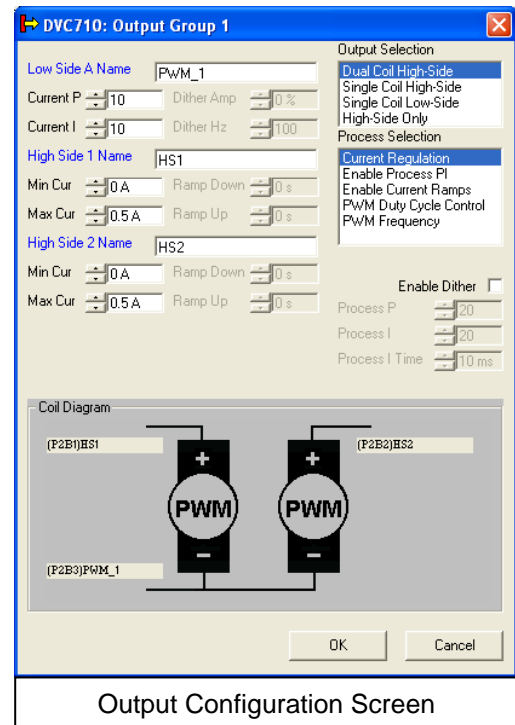
Output Configuration Screen

HIGH-SIDE OUTPUTS (HS OUT) – Qty 6
These outputs are designed to source power supply voltage when enabled. Each output is short circuit protected and will detect both open and short circuits. Open circuit detection is tested when the output is switched off and after its first use. The maximum output current is 3 Amps at each output.

PWM OUTPUTS (PWM OUT) – DVC710 - Qty 3, DVC707 – Qty 2
These outputs are designed to sink current to ground at the PWM frequency (19 kHz). Each output can be configured for a specified current range for maximum resolution in an application. All outputs are short circuit protected and have open circuit and short circuit detection.

Output Group Configuration
Each Output group can control from 2 to 3 coils representing 1 to 3 valves depending on the valve types. The four supported valve control configurations are namely:

- Dual Coil High Side
- Single Coil High Side
- Single Coil Low Side
- High-Side Only
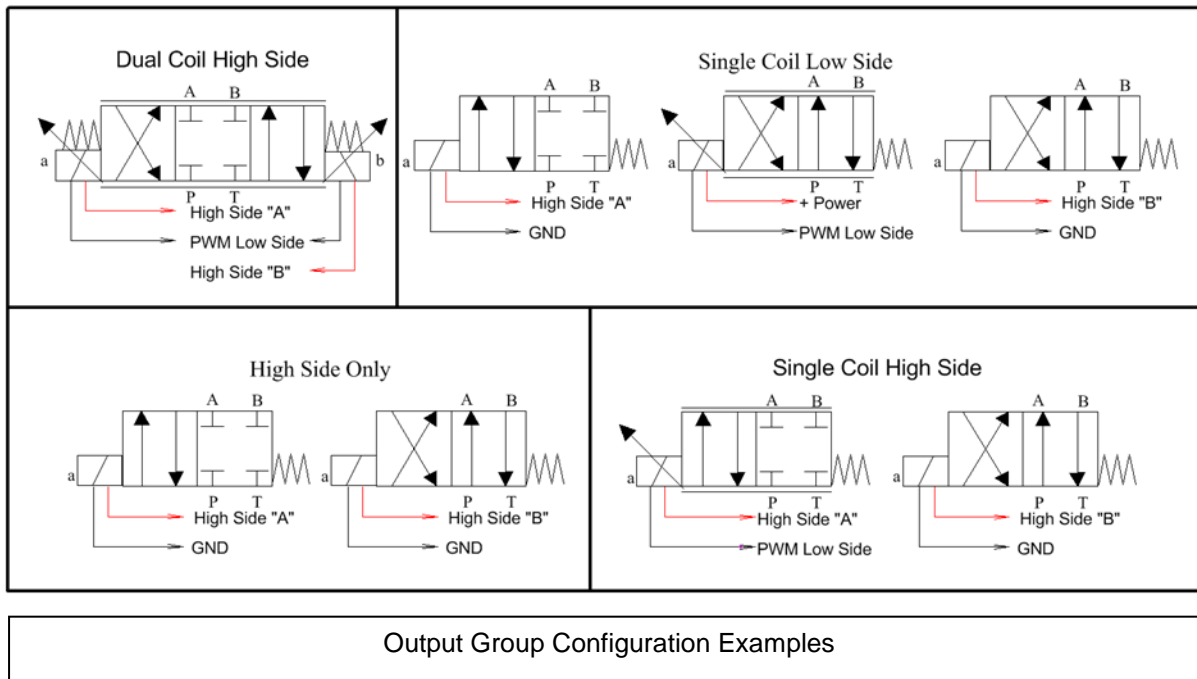
Output Group Configuration Examples

The following gives the definition as well as an overview of each field in the Output Groups Configuration Screen:

Low Side Name
The name used in your application code to access this PWM output and its associated properties.
Range: 16 Characters with no spaces.
Usable characters are A-Z, a-z, 0-9, and "_".

Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid. A valid example is "Boom_Up_Dn".

Current P
This is the value for the P term used in the DVC's PI Loop process for regulating current. See Appendix D
Range: 0 to 655.00
Recommended Range: 0 to 64

Current I
This is the value for the I term used in the DVC's PI Loop process for regulating current. See Appendix D
Range: 0 to 655.00
Recommended Range: 0 to 64

NOTE:
The Current PI process is separate and independent from any other application process, PI loop or Output Process PI mode.

Current P and I values are set to 10 and 10 by default. 95% of applications do not need adjustment.   Adjust the current P and I values if the Proportional output is detecting false opens/shorts or reacting too slow or fast. Typically these need to be adjusted when using dither frequencies below 150Hz or when using valve coils that have high inductance. See Appendix D

Dither Amp
How much current to be applied above and below the set point as dither. See Appendix D & E
Range: 0 to 100%

Dither Hz
The frequency setting of the dither. See Appendix D & E
Range: 1Hz to 500Hz

Enable Dither
Click this option to activate dither. See Appendix D & E
Range: True, False

High Side # Name
The name used in your application code to access this High Side output.
Range: 16 Characters with no spaces. Valid characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid.

Min Cur
The Min Coil current setting for Coil A in Single and Dual Coil selections corresponding to 0% demand.
Range: 0 to 3 A
NOTE: the text on how to set this value in your application described in the Max Cur section below.

Max Cur
The Max Coil current setting for Coil A in Single and Dual Coil selections corresponding to 100% demand.
Range: 0 to 3 A

NOTE:
The Coil Current Gain constants are available to allow the user to change each output's Max Cur or Min Cur set points dynamically in your application. Coil current gains are the constants that the BIOS uses to convert an A/D value to represent an actual current value through the coil. The Coil current gain constants are calibrated and saved into the DVC controller's flash memory when during production testing at HCT's manufacturing facility.

Ramping the Output
To automatically apply ramps to the output, select Enable Current Ramps mode from the process selection menu. Ramps are applied after the demand of the output is calculated by the application. Therefore, with ramps applied you would see the current at the valve follow the desired slew rate while the demand shown on the PLM would indicate the fully demanded value that the output is ramping to.

Ramp Down
When the Enable Current Ramps mode is selected, Ramp Down is the ramp from Max to Min Current. Ramps are applied for all current / PWM% changes.
Range: 0.0 to 65.00 s

Ramp Up
When the Enable Current Ramps mode is selected, Ramp Up is the Ramp from Min to Max Current. Ramps are applied for all current / PWM% changes.
Range: 0.0 to 65.00 s

Output Selection Modes
This is where the user selects the output configuration type. See the output configuration drawing above.

- o Dual Coil High Side
- o Single Coil High Side
- o Single Coil Low Side
- o High-Side Only

Process P
This is the value for the P term used when in Enable Process PI Mode.
Range: 0 to 655.00

Process I
This is the value for the P term used when in Enable Process PI Mode.
Range: 0 to 655.00

Process I Time
This represents the time in between updates to the Integral portion of the output correction for the Process PI Loop. It is used when in Enable Process PI Mode.
Range: 0.10 to 10.00

Process Selection Modes
This is where the user selects the output control mode.
- o Current Regulation
- o Enable Process PI
- o Enable Current Ramps
- o PWM Duty Cycle Control
- o PWM Frequency

Each selection will activate the valid selection boxes.

Current Regulation
Sets output to the standard current regulation mode. The output will drive the demanded current as a percentage of the Min Cur to Max Cur Settings.

Current Regulation compensates for coil resistance variations typically caused by manufacturing tolerances, increases in operating temperature and any other wiring resistances to and from the terminals of the coil as well as variations in supply voltage.

Enable Process PI
NOTE: Enable Process PI can only be used in Single Coil Applications.

Enable Process PI provides a method for users to perform "Closed Loop" operations about a Set Point. When using Process PI, the module will drive its output in an attempt to achieve zero error between Set Point and Feedback signals. The Feedback signal can represent position, speed, pressure etc. The Set Point can represent type of input or a fixed or calculated variable.

Under Process PI control, current to the coil is increased or decreased in order to make the Feedback and the Set-Point signals equal (zero error). The outputs access variables are Low-Side Name.Setpoint and Low-Side Name.FeedBack. Both Setpoint and Feedback are standard variables i.e. 0 to 1023. When using the Process PI function, the Process P and Process I variables may be adjusted to determine how aggressively the DVC will command the output in response to changes in the error.

This feature is generally used when the programmer wants the DVC module to automatically control the behavior of an output in an effort to follow, maintain or adjust to certain input criteria. Under Process PI control, the current through the coil will increase to its maximum as long as the feedback is less than the set-point and decrease to zero if the feedback is greater than the set-point. Typically, Process PI control variables should be processed in the Always bubble. Note that when using the application simulator during development and not driving an actual coil the PWM command will never exceed the set-point %.

For example: if your system is capable of generating 0 to 3000 psi and you wish to generate 1500 psi you would set the Set Point variable to 50% or 512. Then typically in the Always code (since it updates every 10ms) you would read the pressure sensors input voltage to determine the current pressure and set the Feedback variable to reflect the current pressure. The controller would then attempt to drive the feedback to 50% or 1500 psi.

Enable Current Ramps
This is the same as Current Regulation mode with the addition of adding ramps to the outputs response.

PWM Duty Cycle Control
This mode turns off all current regulation and the output is driven directly as a function of duty cycle. When in PWM Duty Cycle Mode, care should be taken to ensure that output devices are not over driven.

PWM Frequency
The PWM Frequency mode allows the user to use the output as a variable frequency output for driving equipment that requires a PWM input at frequencies from 1Hz to 100Hz. In this mode, the Duty Cycle can be either fixed or variable as required by the programmer. The variable to control the Frequency is "PWMname.Frequency" and is set in increments of 1/10 of a hertz (100 = 10Hz). The variable to control the Duty Cycle is "PWMname.DutyCycle" and is set either as a percentage or as a 10 bit number (0 – 1023).

Programming the Different Output Group Valve Configurations
The variables used to control a DVC's proportional output differ depending on the outputs selected configuration and mode of operation.

NOTE: Should the programmer inadvertently use a variable that is not defined for a particular Output or Process Mode configuration you will get a compile error when you compile your application.

The standard list of variables used to control Output Group 1 is listed below using their default names:
PWM_1 – Sets / tests the outputs demand.
PWM_1.Enable – Enables or disables operation of the proportional output.
PWM_1.Dir – Sets / tests which HS output is used with the proportional output (Dual Coil HS Mode only).
HS_1 – Sets / tests the state of the HS output (Single Coil Low Side and High Side Only Modes only).
HS_2 – Sets / tests the state of the HS output (Single Coil High Side, Single Coil Low Side and High Side Only Modes only).

For each of the four **Output Selection** configurations a subset of these variables is used.

Dual Coil High Side (DCHS) mode (HS outputs are controlled by the BIOS)

PWM_1.Enable – Enable or disable the output.
PWM_1.dir – Set which side/direction the valve or pump is driving.
PWM_1 – Sets how hard the output is driven.
HS1 – On when PWM_1.dir = true (Automatically controlled by the BIOS)
HS2 – On when PWM_1.dir = false (Automatically controlled by the BIOS)

Single Coil High Side
PWM_1.Enable – Enable or disable the output.
PWM_1 – Sets how hard the output is driven.
HS1 – On when PWM_1.Enable = true (Automatically controlled by the BIOS).
HS2 – Sets or resets the state of the HS output.

Single Coil Low Side
PWM_1.Enable – Enable or disable the output.
PWM_1 – Sets how hard the output is driven.
HS1 – Sets or resets the state of the HS output.
HS2 – Sets or resets the state of the HS output.

High-Side Only
HS1 – Sets or resets the state of the HS output.
HS2 – Sets or resets the state of the HS output.

The five **Process Selection** configurations may require additional variables as well.

Current Regulation
Use standard variables listed above.

Enable Process PI
Use standard variables listed above.

PWM_1.Setpoint – Sets the target for the feedback input
PWM_1.Feedback – Sets the value of the feedback input for the PI process

The PWM_1.ProP, PWM_1.ProI and PWM_1.ProITime variables may be used in application code when in Process PI Mode.

Enable Current Ramps
Use standard variables listed above.

The HS1.RampUp, HS1.RampDown, HS2.RampUp and HS2.RampDown variables may be used in application code when in Enable Current Ramps Mode.

PWM Duty Cycle Control
Use standard variables listed above.

PWM Frequency
Use standard variables listed above.

PWM_1.Frequency – Sets the PWM frequency when in PWM Frequency Mode
PWM_1.dutycycle – Commands the outputs percentage of PWM

Output Group Code Sample

Code	Comments
PWM_1. Enable = True	Enable PWM to drive
PWM_1 = Uni_1	Sets the PWM_1 outputs demand to follow the Universal Input
PWM_1 = 50%	Set PWM or current to 50% of output range
PWM_1.Frequency = 750	Set the PWM Frequency for PWM_1 to 75Hz
PWM_1.Dutycycle = 512	Set the Duty Cycle for PWM_1 to 512 or 50%
PWM_1.Dir = Uni_1.Dir	Set direction based on position of input
If (PWM_1.Short) then	Test for shorted coil
If (HS1.Open) then	Test for open coil
HS1.RampUp = 100	HS1 Ramp Up from min to max = 1 second (.01 per)
PWM_1.MaxCurA = (650 * 100) / HC_Coil_Gain_OG1	Set PWM_1 Max Current to 650mA

3.16 Input Output Functions

Input Output functions are useful in applications where the desired output is not linear to the input. These functions can be used to vary the control resolution of the output at different points etc. They are also useful if the output levels are not known. The user can use the "Program Loader Monitor" to adjust the output levels to control the system correctly.

When using a function curve, the DVC's BIOS automatically calculates the outputs command value for a given input value. This takes one extra execution cycle typically 10ms to complete. The result of this is that a change at the input is delayed one Logic Cycle from being seen at the output. In other words, once you set an input value, the output value will be unchanged until the next logic sequence commanding the output is executed.
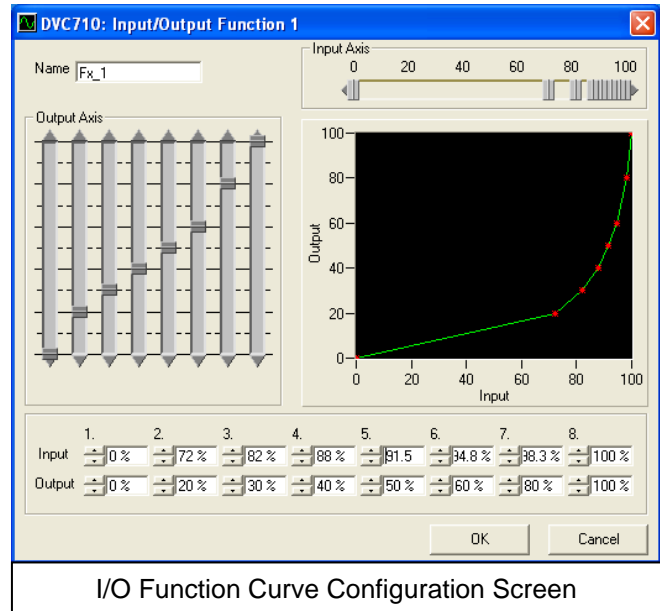
I/O Function Curve Configuration Screen

To minimize latency induced by using the Function Curves, it is recommended that the input code for function curves be placed either in the Always bubble or in a preceding logic sequence from the output command code. This way, the output command is processes immediately after the commanding input is passed through the Function Curve. See How Logic Sequences are executed for more information on how code is executed in the DVC modules.

NOTE: The DVC707/710 BIOS software does linear interpolation between consecutive points in the function curve.

Name
This is the access word for the function's associated properties.
Range: 16 Characters with no spaces. Valid characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid.

Input
The 8 movable points on the x-axis with each input be of ascending values
Range: 0 to 100 %

Output
The 8 movable points on the y-axis, one for each input or x-axis value
Range: 0 to 100 %

Input Output Function Sample

Code	Comments
Fx_1.In = Ana_1	Input to Fx_1 set to Ana_1 input %
PWM_1 = Fx_1.Out	Set PWM Output to Output% from I/O Function
Fx_1.X0 = EESAVX0	Set X0 Input % to % value from EE memory
Fx_1.Y0 = EEvarFx_1_Y0	Set Y0 to the value in EEMEM
EEvarFx_1_Y7 = Fx_1.Y7	Set EEMEM to the value of Y7

3.17   LED Indicators

The DVC707 / DVC710 have four Red/Green LEDs, positioned on top of the module. They are labeled, Module Status, CAN Status, % Current and Error Status.

DVC710 LED Layout

Operation
When a BIOS or application is being downloaded to the controller all LED's will be off. The Following is a list of the individual LED behaviors:

Module Status	
LED STATE	**MEANING**
Off	There is no power applied to the module.
On **GREEN**	The module is operating in a normal condition.
Flashing **GREEN**	Device is in standby state. May need servicing.
On **RED**	Module has an unrecoverable fault.
On **YELLOW**	System Disabled active
Flashing **RED**	Low Supply Voltage.

CAN Status	
LED STATE	**MEANING**
Off	There is no J1939 device (or other DVCs) in the project.
On **GREEN**	Communication established with another DVC module through DVC Devicenet.
Flashing **GREEN**	Waiting to establish communication with another DVC (i.e. DVC61) or J1939 Bus Enabled
On **RED**	The device has detected an error that has rendered it incapable of communicating on the network.
Flashing **RED**	The DVC Devicenet communication is in a timed-out state

% Current O/P	
LED STATE	**MEANING**
Off (Outputs Disabled) **GRN** (0-33%) **YEL** (34-66%) **RED** (66-100%)	
Flashing **GREEN**	PWM or High Side output Open circuit detected
Flashing **RED**	PWM or High Side output Short circuit detected

Error Status	
LED STATE	**MEANING**
Off	No errors
On **RED**	PWM1 Open or Short Detected
On **GREEN**	PWM2 Open or Short Detected
Flashing **YELLOW**	High Side Open or Short Detected
Multi Digit Blink Code	Application defined blink codes.

DVC710 Status LED

The programmer can send different single or multi digit blink codes to the status LED by using the application variable "Blinkcode". In the application code, the programmer would assign a 1, 2 or 3 digit non-zero value to the Blinkcode variable (i.e. Blinkcode = 501). The BIOS would then read this value, and then start flashing the Status LED to the assigned code, for example, in the example above (Blinkcode = 501) the Status LED would flash 5 times followed by a short pause then flash 10 times followed by a short pause then flash once then stop if no other code has been assigned. If a new code was assigned during the time that the code was flashing, there would be a longer pause before the next code began flashing. After the BIOS reads a blink code it will reset the Blinkcode variable to 0 allowing the application to test and see if the BIOS is ready for the next blink code assignment.

The following is an example of valid Blink Code assignments:
[0]   No Blink Code Assigned
[1 – 9]   Single Digit Blink Code
[10 – 99]   Two Digit Blink Code
[100 – 999]   Three Digit Blink Code
[>999]   Invalid assignment, BIOS would ignore this and reset the Blinkcode variable to 0

DVC707 Status LED

The status LED on the DVC707 is programmed by using the application variable "Blinkcode" (same as for the DVC710). For the DVC707, the programmer assigns a number between 1 and 65535 that acts like a countdown timer. The led will blink at the rate of 1Hz until it reaches zero again.

3.18   Program Variables

Program variables are storage locations in memory with unique identifiers which contain values that are paired with specific application functions. Program variables may represent the configuration or state of an input or output, a defined variable created by the programmer, an EEMEM location or storage space for J1939 data etc. Most application variables are open to control by the programmer, while certain variables while available for testing (mainly used for module configuration) should not be manipulated by the application program.

Miscellaneous Variables		
Name	**Description**	**Range**
Supply[1]	The Power Supply voltage. The value returned is in units of supply volts (sv)	
DVC_Temperature[1]	Internal DVC710 controller temperature. The value returned is in units of °C + 40. Therefore, – 40°C is returned as 0.	
FreeRunningTimer	16 bit counter that continually increments every 100 micro seconds. Counts from 0 to 65535 (6.5 seconds) then begins again. Could be used to track timing between two events.	
MACID	This variable returns the MACID of the DVC controller.	
HC_Coil_Gain_OG1[1] HC_Coil_Gain_OG2 [1] HC_Coil_Gain_OG3 [1]	Coil gain constant used by the BIOS to determine actual coil current from A/D values derived by the controller's current feedback circuits. Max_cur = (current_in_ma * 100) / HC_Coil_Gain_OG1	0 to 1023
LC_Coil_Gain_OG1[1, 3] LC_Coil_Gain_OG2 [1, 3] LC_Coil_Gain_OG3 [1, 3]	Same as above for Low Current application, (90mA or less Max Current)	0 to 1023
BlinkCode	Commands the Status LED. See The Status LED for more information.	0 to 65535 – DVC707 0 to 999 – DVC710

Universal and Analog Inputs		
Name	**Description**	**Range**
Name	Name of input	0% to 100% (0 – 1023)
Name.Dir	Inputs direction bit	False (Lower Side), True (Upper Side)
Name.RawVolts	Volts or Milliamps	0 to 1023 * Scale [2]
Name.RampVolts	Ramped Volts = RampVolts * Scale Factor	0 to 1023 * Scale [2]
Name.MinVolts	Name = 0% (i.e. Center not enabled)	0 to 1023 * Scale [2]
Name.MaxVolts	Name = 100%	0 to 1023 * Scale [2]
Name.MinLimit	Threshold for Name.MinF	0 to 1023 * Scale [2]
Name.MaxLimit	Threshold for Name.MaxF	0 to 1023 * Scale [2]
Name.RefMinLimit	Threshold for Name.MinRF	0 to 1023 * Scale [2]
Name.RefMaxLimit	Threshold for Name.MaxRF	0 to 1023 * Scale [2]
Name.CenterVolts	Set Point for Center Volts	0 to 1023 * Scale [2]
Name.Deadbandv	Plus and minus volts about CenterVolts	0 to 1023 * Scale [2]
Name.MinF	Set when Input Voltage is less than Min Limit	False (ok), True (Outside Limit)
Name.MaxF	Set when Input Voltage is greater than Max Limit	Range: False (ok), True (Outside Limit)
Name.LOS	Set after time out. For Universal Pulse inputs Only	False (Pulses ok), True (No Pulse Input)
Name.RealRPM	The Unsigned Integer Value of the RPM. For Universal Pulse inputs Only	0 to 9999
Name.Counter	Value of the Counter. Universal Pulse inputs Only	0 to 65535
Name.PulsesPerRev	Pulses Per Revolution RPM inputs only.	0 to 9999

Digital Inputs		
Name	**Description**	**Range**
Name	Name of the switch	False or Off, True or On
Name.RealRPM [3]	The Unsigned Integer Value of the RPM.	0 to 9999
Name.PulseTimeout [3]	Get/Set Pulse Timeout for Loss of Signal	0 to 65535
Name.PulsesPerRev [3]	Get/Set Pulses Per Revolution	0 to 9999
Name.Counter [3]	Get/Set Unsigned Integer Value of the Counter. Pulse inputs Only	0 to 65535
Name.LOS [3]	Loss of Signal flag set after time out. For Pulse inputs Only	False (Pulses ok), True (No Pulse Input)

Outputs Dual Coil High Side

Name	Description	Range
Name	Name for the PWM Output	0% to 100% (0 – 1023)
Name.Dir	PWM Outputs direction bit	False (Lower Side), True (Upper Side)
Name.Enable	PWM Outputs enable bit	True [PWM Enabled], False [PWM = 0]
Name.Short	PWM Output Short Status Flag	Off [Coil Ok], On [Coil Short]
Name.Open	PWM Output Open Status Flag	Off [Coil Ok], On [Coil Open]
HSEven#Name.Rampup	Set the ramp up rate (time to travel from 0% to 100%)	0.0 to 65.00 s
HSEven#Name.Rampdown	Set the ramp down rate (time to travel from 100% to 0%)	0.0 to 65.00 s
HSEven#Name.Short	Get the Coil Flag for Short Status	Off [Coil Ok], On [Coil Short]
HSEven#Name.Open	Get the Coil Flag for Open Status	Off [Coil Ok], On [Coil Open]
HSOdd#Name.Rampdown	Set the ramp down rate (time to travel from 100% to 0%)	0.0 to 65.00 s
HSOdd#Name.Short	Get the Coil Flag for Short Status	Off [Coil Ok], On [Coil Short]
HSOdd#Name.Open	Get the Coil Flag for Open Status	Off [Coil Ok], On [Coil Open]
Name.Cur	Current actual * CurGain = amps	0 – 3.5 amps
Name.RampCur	Ramped Current*CurGain= amps	0 – 3.5 amps
Name.CurErr	Current Error = RampCur – Cur	16 bit signed integer
Name.CurSumErr	Current Error accumulated over time	0 – 65535
Name.CurP	Current Proportional Term Constant "P"	0 – 255 [4]
Name.CurI	Current Proportional Term Constant "I"	0 – 255 [4]
Name.MinCurA	Minimum Current Coil A *.001 = amps	0 – 3.5 amps
Name.MaxCurA	Maximum Current Coil A *.001 = amps	0 – 3.5 amps
Name.MinCurB	Minimum Current Coil B *.001 = amps	0 – 3.5 amps
Name.MaxCurB	Maximum Current Coil B *.001 = amps	0 – 3.5 amps

High Side Only

Name	Description	Range
HSEvenName	Set/Get the state of the Output	Off, On
HSEven#Name.OpenDisable	Set the Disable the Outputs Open Detection Feature	0 [Enabled], 1 [Disabled]
HSEven#Name.Short	Get the Coil Flag for Short Status	Off [Coil Ok], On [Coil Short]
HSEven#Name.Open	Get the Coil Flag for Open Status	Off [Coil Ok], On [Coil Open]
HSOddName	Set/Get the state of the Output	Off, On
HSOdd#Name.OpenDisable	Set the Disable the Outputs Open Detection Feature	0 [Enabled], 1 [Disabled]

Single Coil High Side / Single Coil Low Side		
Name	**Description**	**Range**
(Low-Side/PWM) Name	Set or test the state Current Target or Process in percentage of min to max current 0 = 0 Current, .1% = Min Current, and 100% = Max Current	0% to 100% (0 – 1023)
Name.Enable	Set the PWM to 0 or enable the PWM	True [PWM Enabled], False [PWM = 0]
Name.Short	Get the Coil Flag for Short Status	Off [Coil Ok], On [Coil Short]
Name.Open	Get the Coil Flag for Open Status	Off [Coil Ok], On [Coil Open]
Name.Rampup	Set the ramp up rate (time to travel from 0% to 100%)	0.0 to 65.00 s
Name.Rampdown	Set the ramp down rate (time to travel from 100% to 0%)	0.0 to 65.00 s
Name.Frequency	Set PWM frequency	0 to 1000 for 0 to 100hz
Name.Dutycycle	Set PWM duty cycle	0 to 1023 for 0 to 100%
Name.Freqerror	Returns error 0 = No Error 1 = Frequency error, 2 means duty cycle error	
HSEvenName [5]	Set the Bang-bang Coil to On or Off	Off, On
HSEven#Name.Short	Get the Coil Flag for Short Status	Off [Coil Ok], On [Coil Short]
HSEven#Name.Open	Get the Coil Flag for Open Status	Off [Coil Ok], On [Coil Open]
HSOddName [6]	Set the Bang-bang Coil to On or Off	Off, On
HSOdd#Name.OpenDisable	Set the Disable Coil Open Detection	0 [Enabled], 1 [Disabled]
HSOdd#Name.Short	Get the Coil Flag for Short Status	Off [Coil Ok], On [Coil Short]
HSOdd#Name.Open	Get/Set the Coil Flag for Open Status	Off [Coil Ok], On [Coil Open]
Name.Cur	Current actual * CurGain = amps	0 – 3.5 amps
Name.RampCur	Current ramped Current*CurGain= amps	0 – 3.5 amps
Name.CurErr	Current Error = RampCur – Cur	16 Signed Integer
Name.CurSumErr	Current Error accumulated over time	0 – 65535
Name.CurP	Current Proportional Term Constant "P"	0 – 255 [4]
Name.CurI	Current Proportional Term Constant "I"	0 – 255 [4]
Name.MinCurA	Minimum Current Coil A *.001 = amps	0 – 3.5 amps
Name.MaxCurA	Maximum Current Coil A *.001 = amps	0 – 3.5 amps

Process PI Variables		
Name	**Description**	**Range**
Name.Setpoint	The desired % set point position for the output	0 to 100% (0 – 1023)
Name.Feedback	The % feedback position for the output	0 to 100% (0 – 1023)
Name.ProErr	Error = Set point – Feedback	16 bit signed integer
Name.ProSumErr	Error accumulated over time	0 – 65535
Name.ProP	Process Proportional Term Constant "P"	0 – 255 [4]
Name.ProI	Process Proportional Term Constant "I"	0 – 255 [4]
Name.ProItime	Update / Integration Time	0.0 to 650.00 s
Name.Cur	Current actual * CurGain = amps	0 – 3.5 amps
Name.RampCur	Current ramped Current*CurGain= amps	0 – 3.5 amps
Name.CurErr	Current Error = RampCur – Cur	16 bit signed integer
Name.CurSumErr	Current Error accumulated over time	0 – 65535
Name.CurP	Current Proportional Term Constant "P"	0 – 255 [4]
Name.CurI	Current Proportional Term Constant "I"	0 – 255 [4]
Name.MinCurA	Minimum Current Coil A *.001 = amps	0 – 3.5 amps
Name.MaxCurA	Maximum Current Coil A *.001 = amps	0 – 3.5 amps

I / O Function Variables		
Name	**Description**	**Range**
Name.In	The Input of the Transfer Function	0% to 100% (0 – 1023)
Name.Out	The Output of the Transfer Function	0% to 100% (0 – 1023)
Name.X0	The X0 of the input/output function	0% to 100% (0 – 1023)
Name.X1	The X1 of the input/output function	0% to 100% (0 – 1023)
Name.X2	The X2 of the input/output function	0% to 100% (0 – 1023)
Name.X3	The X3 of the input/output function	0% to 100% (0 – 1023)
Name.X4	The X4 of the input/output function	0% to 100% (0 – 1023)
Name.X5	The X5 of the input/output function	0% to 100% (0 – 1023)
Name.X6	The X6 of the input/output function	0% to 100% (0 – 1023)
Name.X7	The X7 of the input/output function	0% to 100% (0 – 1023)
Name.Y0	The Y0 of the input/output function	0% to 100% (0 – 1023)
Name.Y1	The Y1 of the input/output function	0% to 100% (0 – 1023)
Name.Y2	The Y2 of the input/output function	0% to 100% (0 – 1023)
Name.Y3	The Y3 of the input/output function	0% to 100% (0 – 1023)
Name.Y4	The Y4 of the input/output function	0% to 100% (0 – 1023)
Name.Y5	The Y5 of the input/output function	0% to 100% (0 – 1023)
Name.Y6	The Y6 of the input/output function	0% to 100% (0 – 1023)
Name.Y7	The Y7 of the input/output function	0% to 100% (0 – 1023)

NOTES:

1. Read Only; Do not try to set these variables.
2. Scale depends on Input Range (0 to 5 =.00489, 0 to 10 = .00977, 0 to 25ma = 0.02158)
3. DVC707 Only.
4. Recommend range; 0 – 64. (do not exceed 64)
5. Available in; Single Coil High Side Mode, Single Coil Low Side Mode and High Side Only Mode. Otherwise, Read Only.
6. Available in; Single Coil Low Side Mode and High Side Only Mode. Otherwise, Read Only.

4 Programming with the Intella™ Tool Set

4.1 Bubble Logic

DVC application programs consist of one or more code sections. The first section is called the Always code and the second and additional sections are called logic sequences. An icon in the main project window identifies each of the sections. These icons represent where the programmer actually writes the application code. Each application has an Always section and optionally any number of logic sequence sections. By right clicking in the project window, a menu will appear that will allow the addition of logic sequence icons.

Each logic sequence is composed of one or more "logic bubbles" that contain application code. Transitions specify under what circumstances code execution will move from the current bubble to another bubble.

Logic sequences, Bubbles, Transitions and the Always code have a defined way in which they are executed. The Always code is executed followed by the code for the active bubble in one logic sequence and finally the outbound transitions defined for the active bubble are evaluated. If one of the transition expressions is true the bubble pointed to by that transition will become the new active bubble the next time the logic sequence is executed. Upon completing this cycle, the Always code is executed again and the active bubble for the next logic sequence and its transitions are executed and evaluated so on and so forth. After the last logic sequence is executed the first one will be executed again during the next logic cycle. This Always code - logic sequence – transition evaluation cycle is repeated every 10ms or longer if the code is complex. In between these cycles, the DVC707/710 BIOS executes and records system input/output value changes and sends and receives CAN Bus messages. Given the frequent execution of the Always code it should contain your systems critical code such as error checking code or critical timing code. Note that the timing between executing each logic sequence is a minimum of 10ms times the number of logic sequences. Logic sequence code is usually where your normal system operation sequences and display code are programmed (i.e. open this valve when this digital input is switched on).

To further help you control the operation of your application from a timing perspective, Logic Sequences can be grouped to provide you a way of tuning the performance of your system while maintaining the logic sequence coding paradigm for different aspects of your system. Right clicking on a logic sequence will give you the ability to add the logic sequence to 1 of 9 groups. The grouped logic sequences are shown graphically connected by the black line. Generally, the non-critical performance parts of your application should be grouped together.
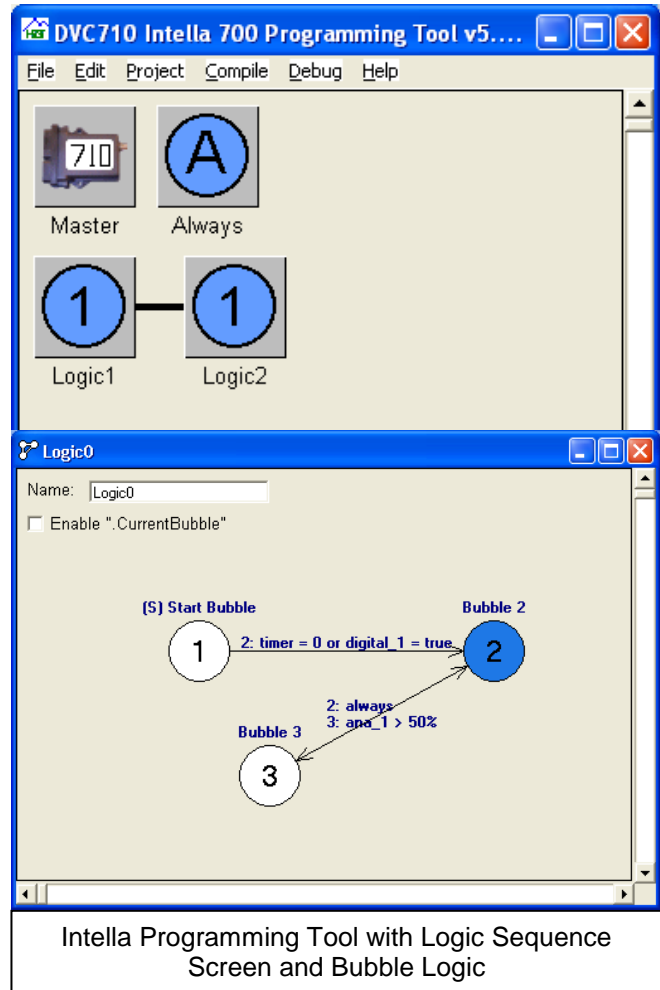

Intella Programming Tool with Logic Sequence Screen and Bubble Logic

Virtual Display updates, DVC61 Display updates, Open Loop Test, EEmemory change validation and LED updates are examples of non-critical parts of most applications.

Within each Logic Sequence Group, only one logic sequence and active bubble is executed during each visit from the BIOS. With reference to the project on the right, the Logic Sequences would be run in the following order with the numbers representing at which Logic Cycle the Logic Sequence is run with respect to t_0 (when started);

- o Logic Sequence 1 – 1, 4, 7, 10 … (30mS)
- o Logic Sequence 2 – 2, 8, 14, 20 … (60mS)
- o Logic Sequence 3 – 5, 11, 17, 22 … (60mS)
- o Logic Sequence 4 – 3, 12, 21, 30 … (90mS)
- o Logic Sequence 5 – 6, 15, 24, 33 … (90mS)
- o Logic Sequence 6 – 9, 18, 27, 36 … (90mS)

Logic Sequences can be copied and pasted in the active project or from one project to another by right clicking on them and selecting Copy or right clicking on open space in the programming tool and selecting Past Logic Sequence.
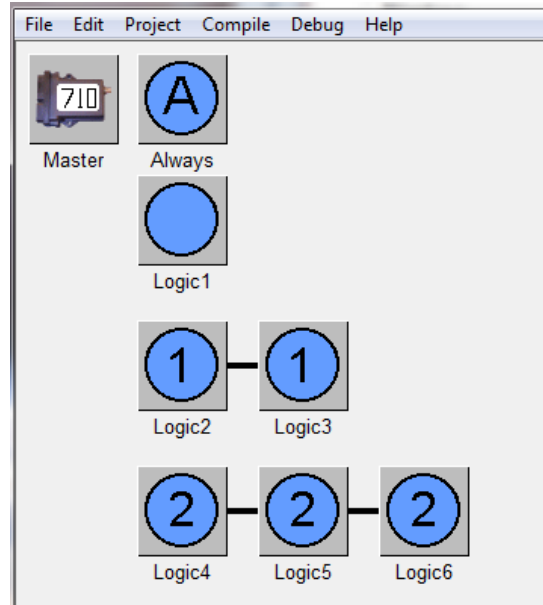
To access the Always code or a particular logic sequence's code, double click the icon. This will open a window for editing the Always code or the bubble diagram for the logic sequence selected. Double clicking a bubble will open its editing window. To delete an object, click the right mouse button on its icon and select Delete.

4.2   Always Bubble

The code in this bubble is executed at every logic cycle (typically 10ms) independent of other logic sequences in the application. Generally safety, error checking, closed loop process control etc. should be implemented in the Always bubble.

4.3   Logic Sequences

To add a Logic Sequence, Right click on the main screen of the Programming Tool and select Add Logic Sequence. Double click on the logic sequence to open it for editing bubble logic. Logic Sequences contains bubbles and transitions that are used to create a logical program flow for part of the user application. Bubbles are containers for the program code while transitions link the bubbles through conditional logic. Each Bubble represents a state in which the program will repeat the same set of programmed logic until a transition is evaluated to be true. When a transition statement becomes true, the program will change the active bubble to the bubble pointed to by the transition line and execute that code the next time that logic sequence is visited by the BIOS. Bubbles can have multiple transitions pointing to different bubbles in the same logic sequence.

Within the Logic Sequence screen there is a check box labeled Enable ".CurrentBubble". If this check box is selected, two more fields appear a Number field (sets the number of characters used in the string created for the current bubble text) and a Caption / Description Selection field (sets how the string will be displayed). This is used in conjunction with a Virtual Display or a DVC61 to display the caption and or description of the current bubble being processed and is useful for troubleshooting application code during development. If selected, set the display variable to "String" in order to display the Caption and or Description information of

the bubble currently being processed on the display. Remember to use no spaces in the text boxes for the Caption and Description fields within the bubbles for the Logic Sequence.

Also note that each logic sequence has one starting point bubble noted by the (S) above the bubble. This is the bubble that will be executed first at power up of the controller. The starting point bubble can be changed to any bubble in the logic sequence.

4.4   Adding and Editing Bubbles

To add a bubble to a logic sequence, right click on open space in the Logic Sequence window and select Add Bubble. To relocate a bubble click on it and drag and drop at will within the Logic Sequence window. The transitions if any will follow the bubble.

To edit bubble code, double click the left mouse button on the bubble. This opens a window with four text entry fields and 1 check box. The Caption field identifies the bubble in the logic sequence window and is merely a comment. This value is also used in the Transition Display for quick reference to the transition logic flow on the bubble screen.   The Description field will be displayed as a comment in the logic sequence window above the bubble icon if the visible check box is checked. Use the Entry Code box for program code that should be executed only once time when the bubble is transitioned to from another bubble. Use the Repeat Code box for the program code that will be executed each time the DVC processor visits the bubble before transitioning to another bubble. When transitioning into a bubble the Entry Code and Repeat Code are both executed the first time the bubble is run. After that only the Repeat Code is executed until the next transition condition is true.
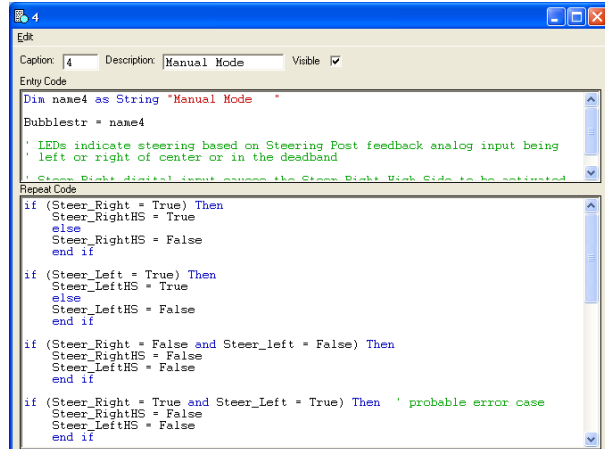
4.5   Adding and Editing Bubble Transitions

Bubble Transitions are used to navigate between bubbles. Transition statements must result as true or false. To add a bubble transition, click the right mouse button on a bubble and select Add Transition. Next, click on the Bubble to which you want the transition to point. Now double click on the transition line to open a window with three text fields and three check boxes. The box titled "Description" is for placing comments about the transition. The next two fields are for the transition logic expressions.
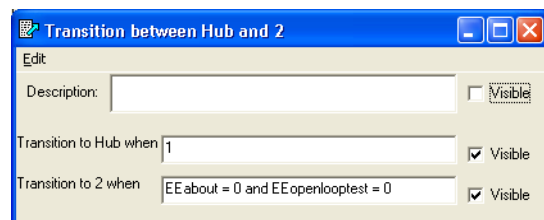
A transition logic expression is like the "Test" part of an "IF" Statement. Conditions can be combined using logical and Boolean operators. The visible check boxes cause the transition code to be displayed in the Logic Sequence window. An empty expression signals no transition defined between the bubbles whereas entering "1" or "Always" indicates a transition from one bubble to another always (after running the preceding bubble once).

Examples of transition expressions would be;
 "dig_1 = true OR ((joystick_left > 50%) AND (reset_timer < 1s))"
"RPM < 1800"

4.6 How Logic Sequences are executed in the DVC

The default rate for a Logic Cycle is 10mS (100 times a second). The "Absolute Maximum" rate is 1ms (1000 times a second) but the recommended maximum rate is 2mS (500 times a second). During each execution of a Logic Cycle, the processor updates the system input/output values, communicates with other modules over the CAN Bus executes the Always code, the active logic bubble in a logic sequence and its out bound transition expressions. Using default values, individual logic sequences will be executed typically every 10ms * the number of logic sequences in the application. For example, an application with three logic sequences would execute a particular logic sequence once every 30ms. Grouping of logic sequences can be used to change the frequency of execution of a particular logic sequence. For instance, assigning 2 logic sequences out of a total of 3 to a group would mean that one logic sequence would execute every 20ms while each of the two logic sequences that were grouped would execute once every 40ms. Multiple groups of logic sequences can be defined. Logic sequences not assigned to a group can be considered to be in their own group for purposes of this discussion. Only one bubble within one logic sequence of a group will be executed during each Logic Cycle. After one pass through all of the groups then the process is repeated with a new logic sequence in each group being executed. When no more logic sequences are defined in a particular group then the first logic sequence in the group is executed again, so on and so forth. This execution pattern may be thought of as a main loop with mini loops in each group.

Each time the active bubble of a logic sequence is executed the execution starts at the top of the repeat bubble code and proceeds to the end of the code after which time the DVC's BIOS checks the transition conditions and executes any true transition conditions. A true transition condition for a bubble causes the new bubble's entry code to be executed during the next (and first) execution cycle of the new bubble followed by the repeat code. In the graphic below, it is assumed that there are no groupings of Logic Sequences.
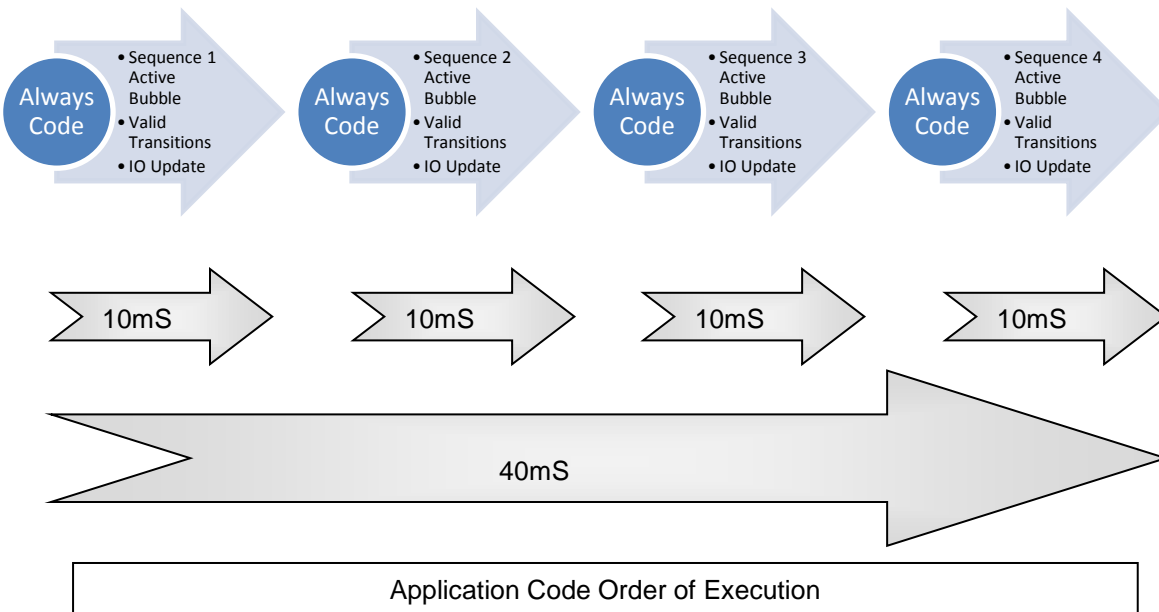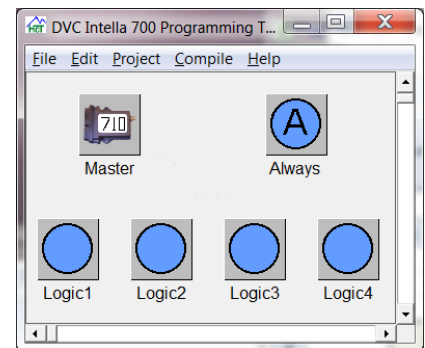
4.7　Program Statements and Operators

The Intella™ programming tool supports the following basic-like statements:
Refer to Appendix B for examples of how to use program statements and logical operators.
Programming statements including keywords are all non-case sensitive.

Code	Comments
dim VarName as uint	Declares a 16 bit (0 to 65,535) unsigned integer as a variable.
dim VarName as timer	Declares a 16 bit variable that once set will decrement at 10ms intervals until it reaches zero.
dim VarName as eemem	Create a 16 bit variable that can be stored in permanent non-volatile memory.
private VarName as uint/timer/string	Create a 16 bit variable that can only be referenced within the logic sequence where it is defined.
const VarName = Value	Create a 16 bit variable with a fixed value that can be accessed by its name.
If (*test*) Then	If statement, see below for Logical Operators
ElseIf (*test*) Then	The else if condition, see below for Logical Operators
Else	The else condition
End If	The end of an If Statement
Var = Algebraic Statement	Algebra Statements can include +, -, *, /, On, Off, True and False
'Comment	Comments are started by using a " ' "
0xFFFF	Syntax for Hexadecimal Notation

Logical Operators

Code	Comments
AND	Returns **true** if both operands are true and **false** otherwise
OR	Returns **true** if either or both operands are true and **false** if both operands are false
XOR	Returns **true** if the first operand is true exclusive of the second operand and **false** otherwise
NOT	Returns the opposite of its operand, **true** id the operand is false and **false** if the operand is true
<	Less than, returns **true** if the first operand is smaller in value than the second operand and **false** otherwise
>	Greater than, returns **true** if the first operand is larger in value than the second operand and **false** otherwise
=	Equal, returns **true** if the both operands are the same value and **false** otherwise
!= *or* <>	Not Equal, returns **true** if the first operand contains a different value than the second operand and **false** otherwise
>=	Greater Than or Equal, returns **true** if the first operand is greater than or equal to the second operand and **false** otherwise
<=	Less Than or Equal, returns **true** if the first operand is less than or equal to the second operand and **false** otherwise
On *or* True	Test equivalent to > 0, Sets a variable to 65535 (0xFFFF)
Off *or* False	Test equivalent to = 0, Resets a variable to 0 (0x0000)

Bitwise Operators

The logical operators AND, OR and XOR may be used to preform Bitwise operations when used in an algebraic statement rather than in an If statement test.

Example Code
```
    '- Reset the lower three bits of Variable_0
Variable_0 = Variable_0 AND 0xFFF8

    '- Test, Ana_1 > 0, AND Dig_1 = true, AND Dig_2 = false
if (Ana_1) AND (Dig_1) AND (NOT Dig_2) then
    '- Set Bit 1 of Variable_0
   Variable_0 = Variable_0 OR 0x01
    '- Test, Ana_1 > 0, AND Dig_1 = true, AND Dig_2 = true
elseif (Ana_1) AND (Dig_1) AND (Dig_2) then
    '- Set Bit 2 of Variable_0
   Variable_0 = Variable_0 OR 0x02
    '- Test, Ana_1 < 50%, AND (Dig_1 = true, OR Dig_2 = true)
elseif (Ana_1 < 512) AND ((Dig_1) OR (Dig_2)) then
    '- Set Bit 3 of Variable_0
   Variable_0 = Variable_0 OR 0x03
end if

    '- Declare a Constant as a variable and set it to the value, 512
Const Fifty_Percent = 512
    '- Set the PWM Output to a value of 512 (50%) using the constant "Fifty_Percent"
PWM_1 = Fifty_Percent
```

Some statements unique to the Bubble Language are:

Code	Comments
A = 100**%**	The "**%**" after a number will return a scaled value of 0 to 1023. Percentage numbers may include a decimal e.g. *PWM_1 = 75.3%*
TimerA = 100**ms**	The **ms** after a number will scale the value to units of 10ms. e.g. *Timer0 = 250ms*
TimerA = 1**s**	The **s** after a number will scale the value to units of seconds. With a resolution of 10mS. **S** numbers may include a decimal e.g. *Timer1 = 5.82s* (5,820mS)
Supply > 20**sv**	The **sv** after a number will scale the value to units of supply voltage. Supply is the voltage powering the unit. **sv** numbers can include a decimal, e.g. *Var = 13.8sv*

NOTES:
1 All variables are Global (may be accessed from anywhere in the application) unless declared as Private (may be accessed only in the logic sequence that it was declared in).
2 Private variables located in different logic sequences may contain repeated names.
3 The DVC710 will support up to 512 EEMEM variables.
4 The DVC707 will support up to 128 EEMEM variables.

4.8 EE Memory

Electronically Erasable Memory (EE Memory) is memory that is maintained when there is no power to the DVC (nonvolatile). The DVC707 has 120 and the DVC710 has 512 usable EE memory locations that can be used to interface to the compiled, running DVC application program. For instance, if, during troubleshooting, the user wanted to change between different virtual display screens, the programmer may create an EEmem variable named 'virtual_screen'. By programming the DVC, the contents of the variable 'virtual_screen' could be monitored to determine which virtual display screen is active. EE memory locations are all unsigned 16-bit values that can store any number from 0 to 65535. EE memory names can be 32 characters in length. Actual EE memory is not used by the application while the program is running. A mirror copy of the EEMemory is stored in the DVC's Random Access Memory (RAM) this copy is accessed directly by the application and is designed to prevent over usage of the physical EEMemory hardware. There is a special command to save this mirror copy to the EE memory and another one to copy the EE memory to RAM. The EE memory hardware has an approximate 1 million writes guarantee. If a new value where to be stored every minute, the DVC would be guaranteed to run for 1.9 Years before memory burnout.

To save new values to the EE memory, execute this line of code in your application: "EECommand = EEWrite". In between writes you will need to reset the EEcommand, "EEcommand = 0". During an EEWrite command, EE memory will only be written to if one or more of the EE memory variables has changed. A typical sequence of code to update EE memory might look like the following:

```
Dim eeVar0 as uint
if (Ana_1 > eeVar0) then      'trap and record highest value seen on Ana_1
   eeVar0 = Ana_1
   eecommand = eewrite
else
   eecommand = 0
end if
```

To declare an EE memory variable use this line of code: "Dim VarName as EEmem"

To save all of the EE memory uses this command: "EECommand = EEWrite"

To restore previously saved EEmemory values to program memory use this line of code: "EECommand = EERead"
EERead (is rarely used but) would be used if you had changed an EE memory variable in your application but had not saved it to permanent memory and wished to reset the variable to the permanent EE memory value.

NOTE: When the DVC707 / DVC710 are powered up, the stored EEMemory values are automatically loaded into the Programs Random Access Memory (RAM). Therefore, you do not need to start your program with an EEread command.

4.9   Long Unsigned Integer Math

All numeric calculations in the DVC application code are executed with 32 bit resolution. Intermediate values are stored as 32 bit unsigned integers. However, only the lower 16 bits of the numeric result can be stored into the applicable variable's memory location. This allows for intermediate values to temporally grow larger than 65k to about 4 billion. However, your final result will be restricted to be less than or equal to 65535. For values larger than 65535, simply perform the calculation again and shift right at the end by dividing as needed, then store this value in a separate variable.

The DVC does only integer math calculations with division resulting in truncation. When you perform division in a calculation the result will be an integer value with no fractional part or remainder saved. For instance $1 \div 2$ would equal 0 rather than 0.5 for any subsequent calculation. Also note that calculations in parentheses will be performed first. For instance the expression $100 \times 2 \times (1/2)$ will equal zero while $100 \times 2 \times 1/2$ will equal 100.

Since the DVC does only unsigned math, negative numbers are not explicitly saved. To calculate a difference between two variables that may result in a negative number, the user may write code like this:

```
If (a < b) then
   Diff = b-a
Else
   Diff = a-b
End if
```

Alternatively, the user may wish to use an offset to ensure that results of a calculation cannot result in less than zero.

5 Programming Examples

This section illustrates how the DVCs are programmed. The first example is a traditional Hello program. Hello introduces you to the basic steps in writing a DVC application and using multi digit blink codes. The second example is an Elapsed Time Clock that introduces you to logic sequences. For both of these examples you only need to have a DVC707 / DVC710 and the DVC Programming Tool / Program Loader Monitor software installed. Other examples are not explained in detail, but key points are highlighted and explained.

5.1   Hello Program

This is an example program to introduce you to most of the DVC programming concepts.
It is designed to operate with a DVC707 / DVC710 connected to your Windows PC computer using the DVC RS232 cable and a +12 Volt DC 1.0 amp power source. The DVC RS232 serial cable is used to load your application into the DVC controller's memory from your Windows PC. The Status LED is programmed differently on the DVC707. See below for more information.

Program Description
DVC710 – The Hello program will flash the Status LED on the DVC710 with a code of "2, 5" followed by a code of "3, 6".

DVC707 – The Hello program will flash the Status LED on the DVC707 with a code of 2 followed by a code of 5.

DVC710 Programming Steps
Open the Programming Tool by double clicking on the Programming Tool icon on the desktop or in the Windows Start Programs Menu HCT Products folder. Double click on the DVC710 Master icon to open the Master DVC configuration window.
Enter Hello in the Program Name field.

Next, enter the Always bubble code. Double click the Always icon in the Project window. Enter the code as shown into the edit window.

```
dim Toggle as uint

if (blinkcode = 0) then
   if (Toggle = 0) then
      blinkcode = 25
      toggle = 1
   else
      blinkcode = 36
      toggle = 0
   end if
end if
```
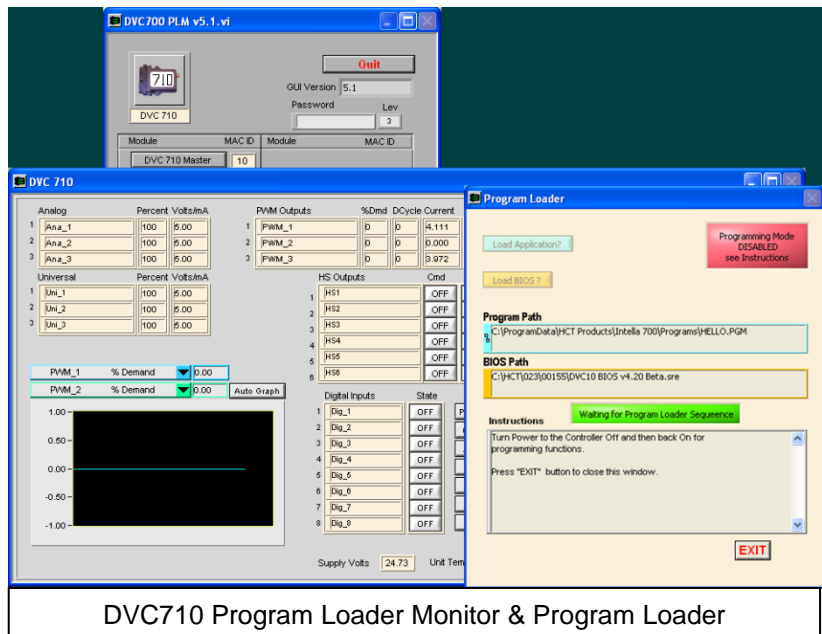
Select the Make item from the Compile menu in the Project window. When prompted save your project as desired. This completes the code generation, compilation and project saving.

DVC707 Programming Steps

Open the Programming Tool by double clicking on the Programming Tool icon on the desktop or in the Windows Start Programs Menu HCT Products folder. Select DVC707 from the Project Type menu. Double click on the DVC707 Master icon to open the Master DVC configuration window.
Enter Hello in the Program Name field.

Next, enter the Always bubble code. Double click the Always icon in the Project window. Enter the code as shown into the edit window.

```
dim Toggle as uint
dim Tmr as timer

if (tmr = 0) then
   if (blinkcode = 0) then
      if (toggle = 0) then
         blinkcode = 2
         toggle = 1
         tmr = 4.0s
      else
         blinkcode = 5
         toggle = 0
         tmr = 8.0s
      end if
   end if
end if
```

Select the Make item from the Compile menu in the Project window. When prompted save your project as desired. This completes the code generation, compilation and project saving.

Next, load the compiled Hello program into a DVC Master Module. Open the Program Loader Monitor program by double clicking on the Program Loader Monitor icon in the c:\Program Files\HCT Products folder.

Now, click on the DVC707 / DVC710 Master Button to open the I/O Screen for the module.
Click the Program Loader button near the center of the window. The Program Loader window will open. Cycle power to the DVC707 / DVC710 this will initiate the boot loader process.

DVC710 - If the path and file name shown do not match the path and file name of the Hello

DVC710 Program Loader Monitor & Program Loader

program, click in the path window then, select the Load Application button.

DVC707 – Select the Load Application button.

Using the file locator window navigate to the project application program and select and download the hello program located where it was saved and compiled.

DVC710 – Select Load Program

When prompted cycle the DVC's power again. The DVC should now run the Status LED as described above for the module that is being used.

5.2 **Elapsed time Display**

This example is designed to operate with only a DVC707 / DVC710 connected to your computer using the DVC RS232 cable and a +12 volt dc 1.0 amp power source. The DVC's RS232 serial cable is used to load your application into the DVC controller's memory from your PC and read data from the DVC for the Virtual Display.

This example introduces;
- Logic sequences
  - Bubbles
  - Transitions
- Virtual Display
- Input Configuration

Intella Programming Tool, Elapsed Time Application
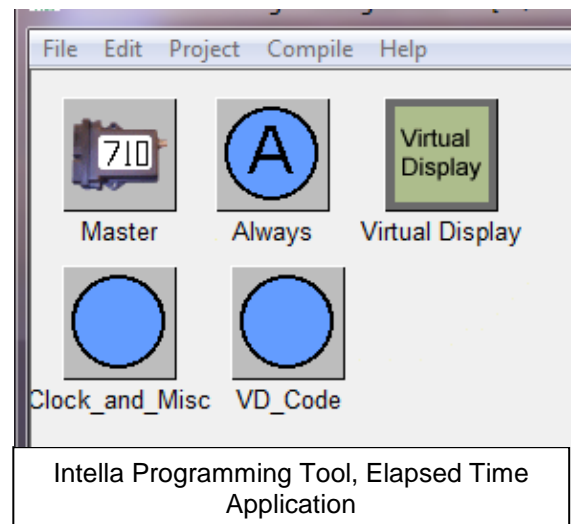
The Elapsed Time program is designed to display Elapsed time from Power Up or a Reset Clock Input (from Dig_1) on the Virtual Display.

The Virtual Display is a debugging tool in the Program Loader Monitor that enables you to display application information as your program executes. To ensure the maximum system bandwidth possible for an application, it is good practice that the Virtual Display and supporting code be removed prior to the production release of an application if the application does not normally use it. This is because that even when the PLM is not attached to the DVC, the Virtual display code is still running and using system resources.

In the programming tool, open a new project and select the module that you will be working with, DVC707 / DVC710. Save and name the new file as desired.

Add a Virtual Display and 2 Logic Sequences to your project. Right mouse click in the project window and select the three items one at a time.

Next configure the Virtual Display by double clicking on the Virtual Display
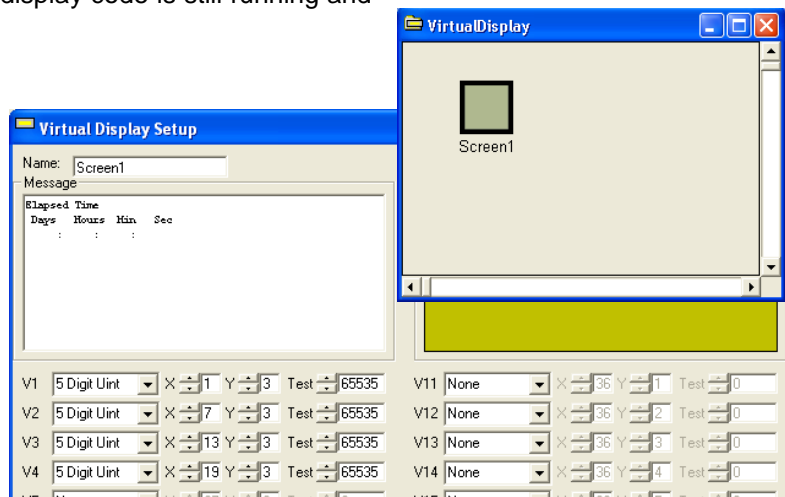
icon in the project window. The window shown appears without the Screen1 icon. Right click in the Virtual Display window and select "Add Screen". Now double click on the Screen1 icon to open the Virtual Display setup window.

Enter the data displayed into the Virtual Display Setup window fields as shown. These fields are used to format the screen's display output. Note the Test Display on the right. It displays what your actual Virtual Display window will look like when your application executes. Then close the Virtual Display Setup Screens.

Next configure the digital input that be used to reset the clock. Double click the DVC710 icon in the project window and select the Dig_ 1 input button. Then configure the input as shown with the name "Reset_Clock", a 100mS Debounce time, Active High and No Toggle.
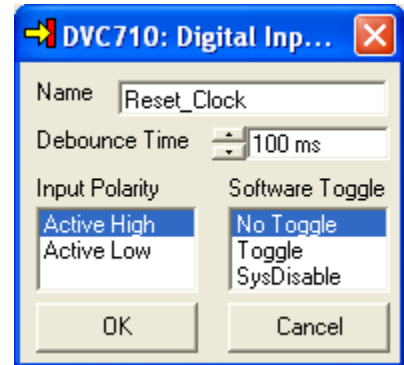
Next, double click on the Logic0 icon in the project window. This will bring up a blank window where we will add bubbles and specify the bubble transition conditions. First change the Name field to "Display_Clock". Next, right click in the Display_Clock window and select "Add Bubble". Repeat this again until there are 6 bubbles displayed.

Open Bubble 1, enter "Reset" for the description and enter the following statement in the Repeat Code:
'Reset all UINT's
if (reset_clock) then
    day = 0
    hr = 0
    min = 0
    sec = 0
end if
Now, close bubble 1.

Then, right click on bubble 1, select "Add Transition", move the mouse to bubble 2 and click. A line connecting the two bubbles will be displayed.
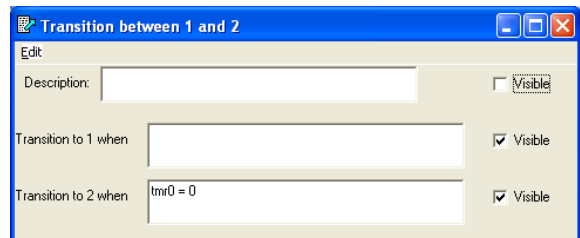
Next, double click on the line and a Transition dialog box will appear. Enter "tmr0 = 0" in the "Transition to 2 when" text box. We will use bubble 2 to initialize the timer interval. After it executes we want to go unconditionally to bubble 3 where the actual Clock code begins.

Now open Bubble 2, enter "Reset_Timer" for the description and enter the following statement in the Entry Code:
Tmr0 = 1s   'Set timer to one second
Now, close bubble 2.

Now open Bubble 3, enter "Seconds" for the description and enter the following statement in the Entry Code:
```
' Test the Seconds count
' increment seconds, or reset seconds count
' and set toggle if 60 seconds has elapsed
if (sec = 59) then
    sec = 0
    toggle = 1
else
    sec = sec + 1
end if
```
Now, close bubble 3.

Now open Bubble 4, enter "Minutes" for the description and enter the following statement in the Entry Code:
```
' Reset toggle,
' Test the Minutes count
' increment minutes, or reset minutes count
' and set toggle if 60 minutes has elapsed
toggle = 0
if (min = 59) then
    min = 0
    toggle = 1
else
    min = min + 1
end if
```
Now, close bubble 4.

Now open Bubble 5, enter "Hours" for the description and enter the following statement in the Entry Code:
```
' Reset toggle,
' Test the Hours count
' increment hours, or reset hours count
' and set toggle if 24 hours has elapsed
toggle = 0
if (hr = 23) then
    hr = 0
    toggle = 1
else
    hr = hr + 1
end if
```
Now, close bubble 5.

Now open Bubble 6, enter "Days" for the description and enter the following statement in the Entry Code:
```
' Reset Toggle
' Increment Days
toggle = 0
day = day + 1
```
Now, close bubble 6.

Now add the rest of the transitions for the Display Clock Logic sequence. Add a transition from bubble 2 to 3. Open the transition and enter "always" in the "Transition to 3 when" text box. Close the Transition.
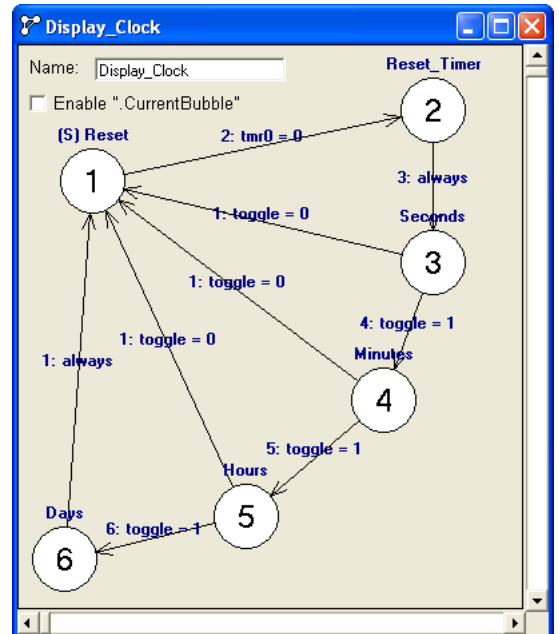
Add a transition from bubble 3 to 4. Open the transition and enter "toggle = 1" in the "Transition to 4 when" text box. Close the Transition.

Add a transition from bubble 3 to 1. Open the transition and enter "toggle = 0" in the "Transition to 1 when" text box. Close the Transition.

Copy and paste the transition from 3 to 1 to bubbles 4 to 1, 5 to 1 and 6 to 1 as follows; Right Click on the transition and select "Copy Transition". Right Click on bubble 4 and select "Paste Transition" then click on bubble 1. Repeat for the remaining two bubbles.

Copy and paste the "Toggle = 1" transition from bubbles 3 to 4 and paste to bubbles 4 to 5 and 5 to 6.

Copy and paste the "always" transition from bubbles 2 to 3 and paste to bubbles 6 to 1.

Arrange the bubbles on the Logic Sequence screen so that the transitions and bubbles are easy to follow, see example.

Next, open the second Logic Sequence; name it "Display_and _Def". Open the logic bubble and write the following code in the "Repeat Code"
virtualdisplay.screen = screen1 'Sets the Virtual Display Screen to Screen 1
virtualdisplay.v1 = day        'Points the Displays V1 variable to the UINT, "day"
virtualdisplay.v2 = hr         'Points the Displays V2 variable to the UINT, "hr"
virtualdisplay.v3 = min        'Points the Displays V3 variable to the UINT, "min"
virtualdisplay.v4 = sec         'Points the Displays V4 variable to the UINT, "sec"
Close the bubble.
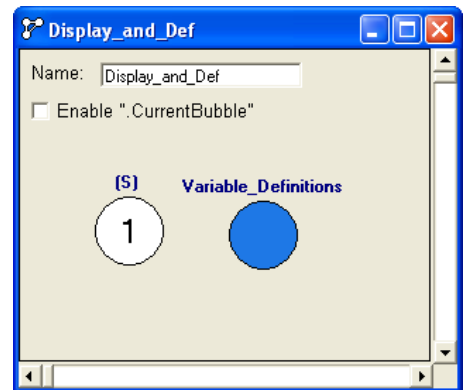


Add a second bubble to the "Display_and _Def" Logic Sequence and name it "Variable_Definitions" in the description field. Type the following statements into the Entry Code to define the applications user defined variables:
'***** Timer variables *****
dim tmr0 as timer

'***** Undefined Integer variables *****
dim sec as uint
dim min as uint
dim hr as uint
dim day as uint
dim toggle as uint
Close the bubble.

NOTE: There is no transition to the "Variable_Definitions" bubble, this bubble is used as a convenient place to define and store User Defined Program Variables. No code will run from this bubble and it will not be visited by the BIOS during program execution.

This application is now ready to be compiled and loaded into a DVC710. The Elapsed Time Clock may be viewed on the Virtual Display Screen selected from the DVC710 PLM Main Screen. Remember, as the clock increments up, it can be reset by toggling "Reset_Clock" (Dig_1) at any time.
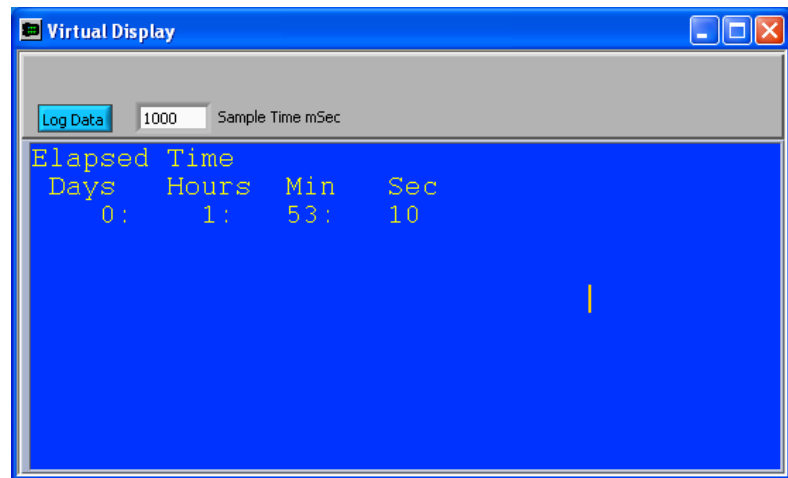


Virtual Display with Elapsed Time Application Running

NOTE: as an example, below is the equivalent of the Display_Clock Logic Sequence written as a nested if statement. In the compiled program, this would require less space.

```
if (tmr0 = 0) then          ' Test for 1 second timeout
   if (sec = 59) then       ' Test Second Counter
      if (min = 59) then    ' Test Minuet Counter
         if (hr = 23) then  ' Test  Hour Counter
            hr = 0          ' Reset Hour Counter
            day = day + 1   ' Increment Day Counter
         else
            hr = hr + 1     ' Increment Hour Counter
         end if
         min = 0            ' Reset Minuit Counter
      else
         min = min + 1      ' Increment Minuet Counter
      end if
      sec = 0              ' Reset Second counter
   else
      sec = sec + 1        ' Increment Second Counter
   end if
   tmr0 = 1S              ' Set Timer for 1 Second
end if
```

5.3 Process PI Closed Loop Control Example

Process PI (used in single coil operation only) is a feature of the DVC707/710 that makes it easy to control a valve's current as a function of two inputs. The first input represents the Set Point. This is the desired system response. The second input is called Feedback and represents the systems current state (position, speed etc.). When selected in the output setup screen, the following sample code will enable the output when the Set Point is above zero and control the DVC's output in an attempt to make the Feedback input to equal the Set Point input:

First set up the Inputs and Output as Shown;
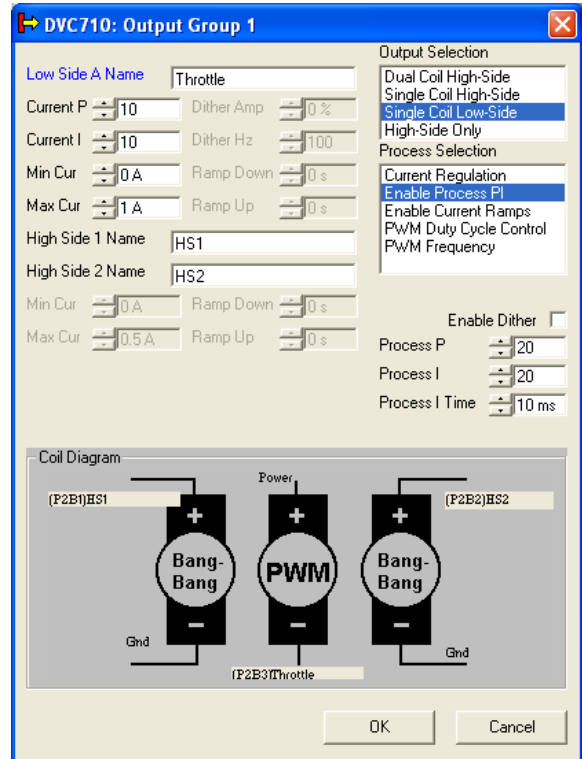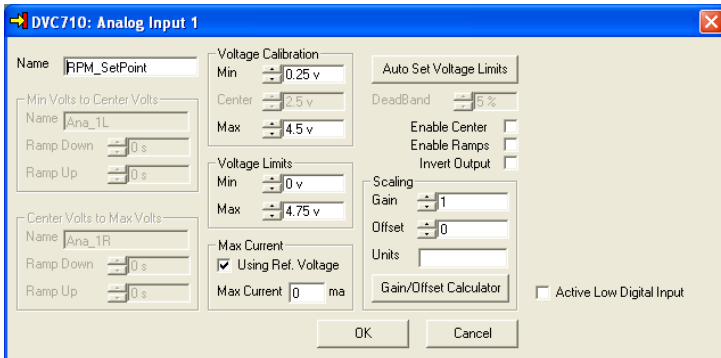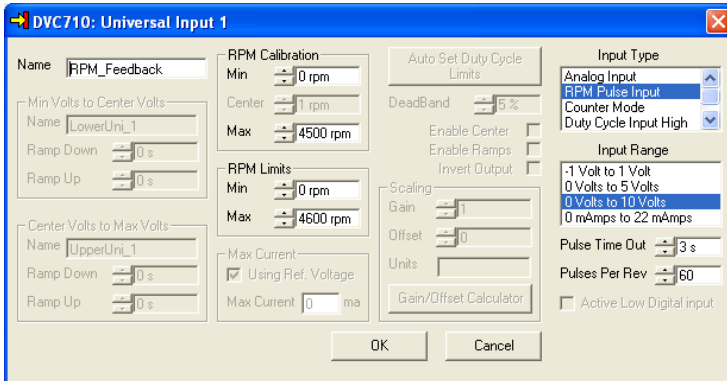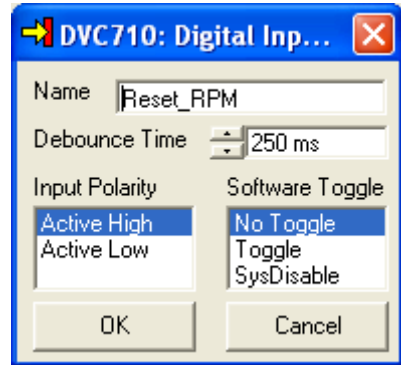
Second, add the following statements to the Always Code;

```
' This code will Monitor the engines RPM "RPM_Feedback" and adjust
' the throttle to maintain the desired SetPoint "RPM_SetPoint"
' regardless of engine load by monitoring the RPM Feedback and
' adjusting the Throttle Output to compensate for error.
' It will also shut down the output if the SetPoint exceeds it's
' Maximum Voltage Limit until it is reset by the Digital Input
' "Reset_RPM"

if (RPM_SetPoint.MaxF) then
    Throttle.Enable = 0    'Sets the enable bit to Off or Disabled
    Throttle.SetPoint = 0  'Forces the outputs Setpoint to 0
else
    Throttle.Enable = RPM_SetPoint    'Sets the enable bit for the output to "On" when SetPoint >0%
    Throttle.SetPoint = RPM_SetPoint  'Points the outputs SetPoint variable to the RPM_SetPoint Input
    Throttle.Feedback = RPM_Feedback  'Points the outputs Feedback variable to the RPM_Feedback In
end if

if (Reset_RPM) then       'Test the "Reset_RPM" switch and
    RPM_SetPoint.MaxF = 0  'reset the RPM_SetPoint Max F Flag
end if
```

Finely, Compile the application and load into a module for testing.

During testing, adjust the Throttle.ProP, Throttle.ProI and Throttle.ProITime for the desired response.

6 DVC Expansion Modules

The DVC707 / DVC710 have a fixed number of Inputs and Outputs for standalone operation. If your system requires more inputs or outputs than one DVC master module can provide, you can add expansion DVC modules or additional DVC707s or DVC710s and have them communicate over the CAN Bus through DVC to DVC communications (DVC Devicenet) or J1939. Additional DVC707s or DVC710s are configured as master controllers for a portion of your application. Each master controller must have its own individual application program and can communicate with expansion modules.

The decision on whether to add expansion DVC modules or more master DVC modules to a project to fill out its I/O needs is dependent on the needs of the project. Almost always expansion modules are less expensive and fast enough to perform in most applications. However, if the project required multiple tightly controlled closed loop operations, it may be better to use a master controller for each of these operations because they can control their outputs without the latency induced from having to report I/O status to a master controller then wait for its commands to be returned over the CAN bus.

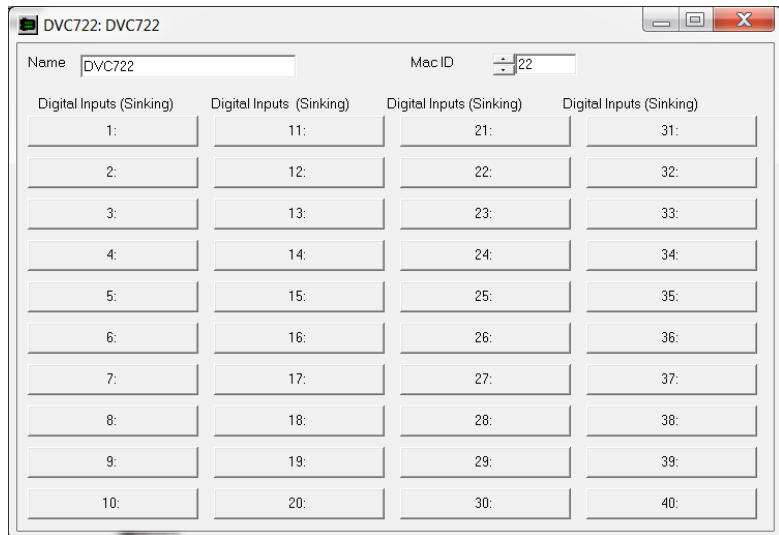At this time the DVC707/710 can talk to the following units:

- o **DVC722** – 40 Digital Input expansion module

- o **DVC741** – 12 High Side digital output module

- o **DVC750** – Universal Input Output expansion module

- o **DVC61** – 4x20 character screen display with 12 display variables and 5 single pole double throw digital inputs

- o **DVC Master to DVC Master** – The ability multiple DVC710s and/or DVC707s to share information via DVC Devicenet

6.1   DVC722 & DVC725

The DVC22 and DVC725 screens are broken up into three areas:
- o   Name
- o   MAC ID
- o   Individual Input Configuration

The Name is used to uniquely identify a particular DVC expansion module. The MAC ID tells the DVC707 / DVC710 how to address the Input Module when communicating over the CAN Bus and must be unique with respect to other modules in the system.   Inputs are configured individually by pressing the numbered buttons to open their set up screens.
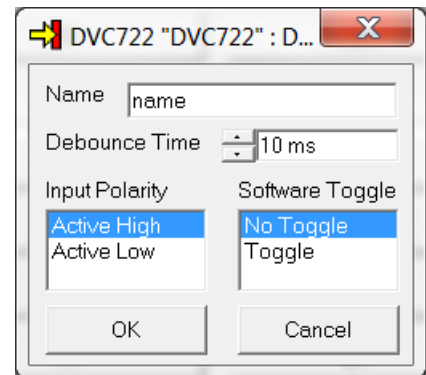
Digital Inputs are Boolean inputs that are either true or false. Zero (0) is considered false by the processor while anything other than zero is interpreted as true. The numerical value of a digital input in the application is either 0 or 65535.

An Input is enabled for use in the application program when a name is assigned to it using the Name textbox for that input. The Debounce Time setting is used to filter out momentary spikes on the input. The Input Polarity determines what voltage level is interpreted as a true or false, or which edge causes a software toggle. Software toggle latches the state of the program variable until the next valid pulse is detected on the input.

Name
The name used in the bubble logic screen to access this variable and properties.
Range: 16 Characters with no spaces. Usable characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number. Compiler Keywords or duplicate names are not valid.

De-bounce Time
The amount of milliseconds to wait before accepting a change in input states
Range: 0 to 9990ms in 10ms Increments

Polarity
Polarity has two types of control depending on the state of the software toggle.
- o When **Active High** is selected, the variable is true when the input is high (+ Voltage).
- o When **Active Low** is selected, the variable is true when the input is low (Ground).

Range: Active High, and Active Low

Software Toggle
The Toggle feature latches the input state on / off on a valid rising or falling pulse with respect to the Polarity setting. A valid pulse is a pulse with a period that satisfies the debounce time.
- o When **Toggle** and **Active High** are set, the variable changes states on valid rising edges.
- o When **Toggle** and **Active Low** are set, the variable changes states on valid falling edges.

Range: Toggle, No Toggle

DVC722/725 Program Variables		
Name	**Description**	**Range**
DVCname.Status	Get state of the flag	0 = Online, 2 = Offline
DVC.name OR name	Get / Set state of the flag	True, False

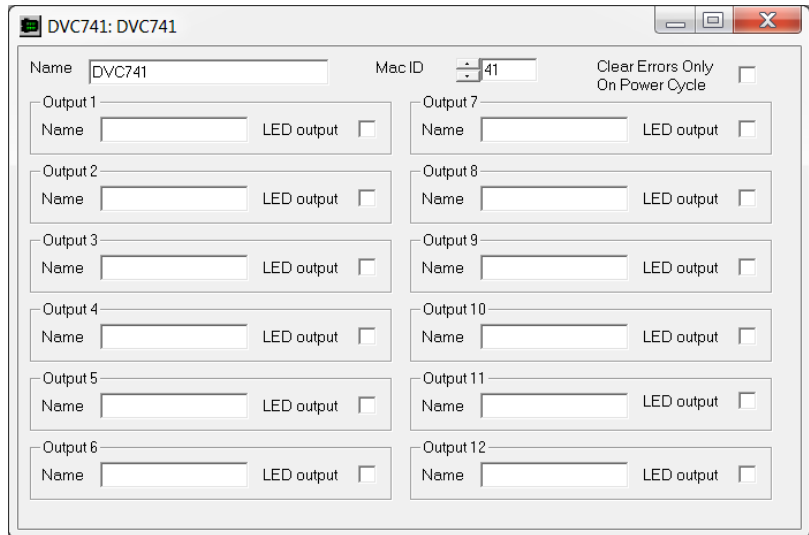| DVC722 Example Code	
Code	**Comments**
If (DVCname.Status = 0) then	Do something if the module is online
DVC741.HS7 = DVC72x.Dig_25 OR HS7 = Dig_25	Set the output to follow the state of the switch

NOTES:
1. Unlike the DVC707 or DVC710, the input state of a DVC722 or the DVC725 <u>cannot</u> be set or reset by the application. The input must be set or reset by the physical input. For example if an input is set to toggle mode and the input is toggled on with a pulse on the input during operation, a second voltage input is required on the input to reset the input flag from its present state.
2. For information on how to configure a DVC725 to run in J1939 CAN Bus Mode, see the DVC725 Product manual.

6.2   DVC741 & DVC745

The DVC41 and the DVC745 screens are broken up into three areas:
- o   Name
- o   MAC ID
- o   Individual Output Configuration.

The Name is used to uniquely identify a particular DVC expansion module. The MAC ID tells the DVC707 / DVC710 how to address the Output Module when communicating over the CAN Bus and must be unique with respect to other modules in the system. To configure the DVC74x's outputs, give the output a Name then check the LED box if that output is used to drive an LED.

High-Side Outputs are Voltage Sourcing outputs that are either true (on) or false (off). The Output is enabled for use in the application program when a name is assigned to it using the Name textbox for that output.

Name
The name used in the application code to access the output.
Range: 16 Alpha/Numeric characters only with no spaces.

LED Output
If this Output is controlling an LED, check this box. This selection will configure the DVC741s internal circuits to pull down the output and prevent the output from dimly driving the LED when off.
Range: Checked or Unchecked

DVC741 Program Variables		
Name	**Description**	**Range**
DVCname.Status	Get state of the flag	0 = Online, 2 = Offline
DVC.name OR name	Turn the output on / off	True, False

DVC741 Example Code	
Code	**Comments**
If (DVCname.Status = 0) then	Do something if the module is online
DVC74x.HS7 = DVC722.Dig_25 OR HS7 = Dig_25	Set the output to follow the state of the switch
NOTE: Variable Names assume default module name "DVC741" & "DVC722"	

NOTES:
1. For information on how to configure a DVC745 to run in J1939 CAN Bus Mode, see the DVC745 Product manual.

6.3   DVC750

The DVC750 Universal I/O Module is an expansion module designed to operate in conjunction with a DVC707 / DVC710 Master Module. The DVC750 has the following I/O;
- o   Four analog (0 to +5Vdc) Analog Inputs
- o   Two analog/pulse (0 to +5Vdc) inputs
- o   Eight digital inputs
- o   Three output groups that are the same as the output groups on a DVC710 Master module. They provide a total of 6 High Side outputs and 3 PWM outputs.
- o   Six +5vdc reference voltage pins are provided to supply power to external sensors.
- o   Six function curves for use with the DVC750's I/O

Over current protection for the DVC750's reference outputs is provided by a 120mA fuse mounted in the module.   A 120mA current limit applies to the sum of the current from the six reference outputs. While an individual reference output may supply up to 120mA, the total current from all reference outputs may not exceed 120mA. The DVC750's RS232 port is used for setting its MAC ID, CAN Bus baud rate and for device monitoring.

The DVC750 communicates to the DVC707 / DVC710 Master module over the CAN Bus using DVC Devicenet. The DVC707 and DVC710 application code reads and sets the input/output variables that are transferred between the modules.

Two different CAN Bus communication mechanisms are used between the DVC707 / DVC710 and DVC750;
- o   High-speed mail
  - o   Mail In – Data from the DVC707 / DVC710 to the DVC750
  - o   Mail Out – Data from the DVC750 to the Master module
- o   Direct memory transfer of data.

The Mail Out interface from the DVC750 to the DVC707 / DVC710 is limited to 4 variables and is accomplished in a single CAN Bus message. The Mail In interface from the DVC707 / DVC710 to the DVC750 is limited to 8 variables and is accomplished with two CAN Bus messages. The direct memory transfer mechanism requires multiple CAN messages to transfer DVC750 I/O status to and from the DVC707 / DVC710 and is reserved for slower types of I/O. Commands from the DVC707 / DVC710 directly to the proportional outputs on a DVC750 are restricted to the Mail In communication because of its higher speed requirement. The names of the Mail inputs and outputs are mapped to DVC750 I/O variable names. The DVC707 / DVC710 and DVC750 BIOS send and receive the mail data as Device Net messages.

Mail Outputs (DVC750 to DVC707 / DVC710) are used to transfer Analog Pulse counts, Analog RPM values, etc. to the DVC10 for processing. Mail Inputs (DVC707 / DVC710 to DVC750) are used for controlling proportional outputs. The lower speed direct memory transfer is used for other commands and data transfer. Mail Outputs are sent every 10ms to the DVC707 / DVC710 while Mail Inputs are sent to the DVC750 every 20ms. Direct memory transfers are completed every 40ms.

The DVC750's I/O configuration screen is very similar to the DVC710's screen with the addition of the Name, MAC ID and I/O Assignment fields. Also note the different number of Analog/Pulse Inputs and that only six function curves are provided instead of eight.

The Name field is used to identify this DVC750 module and serves as the prefix for DVC750 I/O names in application code.

The MAC ID tells the DVC707 / DVC710 how to address the DVC750 when communicating over the CAN Bus and must be unique with respect to other modules in the system.
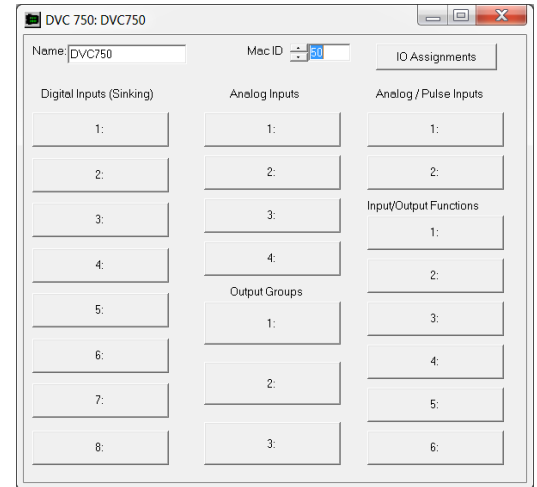
Digital Inputs
The DVC750's Digital Input Screen is laid out like the DVC707 / DVC710 Digital Input screens.

Analog Inputs
The DVC750's Analog Input screen is laid out like the DVC707 / DVC710 Analog Input screens.

Analog / Pulse Inputs
The DVC750 Analog / Pulse input screen is laid out like the DVC707 / DVC710 Universal Input screen without the Voltage Range selection. Otherwise, it is used in the same manner as the Universal Input screen for the DVC707 / DVC710.

Output Groups
The DVC750 Output Group screen is laid out like the DVC710 Output Group screen and is used in the same manner.

Input / Output Functions
The DVC750 Input / Output Functions screen are laid out like the DVC710 Input / Output Functions screen.

I/O Assignments
The I/O Assignments screens are used to set up Input to Output relationships, implement I/O Function Curves and map or name Mail variables.

Output Groups
There are three Output Groups screens. By using the pull down menus on the right, you may assign different available inputs to control the listed output variables. Available selections depend on the setup of the output group (i.e. Dual Coil High Side etc.). To control an output feature directly through application code, leave the check box in that row open (unchecked). Otherwise, use the pull down menu in that row to define a direct relationship with available DVC750 inputs etc.
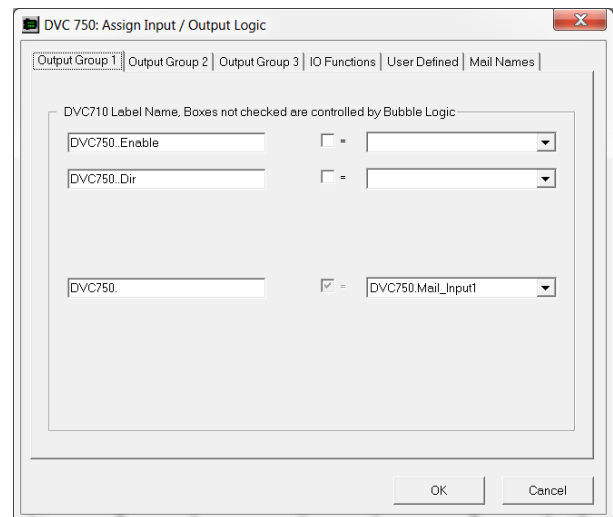
I/O Functions
The I/O Functions screen allows the user to assign individual inputs to each I/O Function curve. Use the pull down menus on the right side of the screen to assign an input to the associated Function Curve. For more information see §3.16 Input Output Functions

User Defined
The User Defined screen is used to assign DVC750 I/O information to each of the four Mail Outputs. Use the pull down menus to assign available variables to the Mail Outputs.

Mail Names
The Mail Names screen is used to assign names to the eight Mail Input variables and four Mail Output variables of the DVC750. These variables are available for use in the application code using the prefix of the module name (i.e. DVC750.Mail_in1).
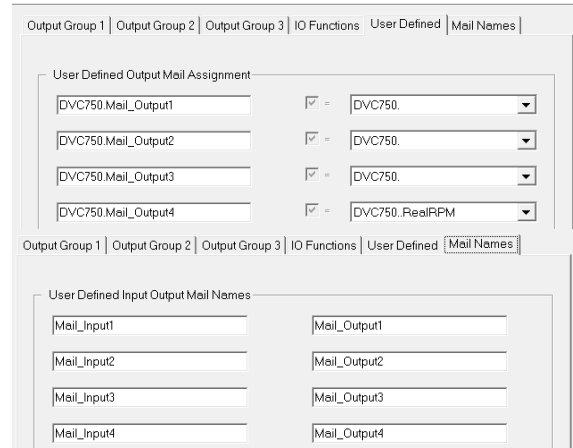
- o Mail Inputs are inputs to the DVC750 from the application code and are used as inputs to DVC750 I/O Function Curves or as a PWM outputs command.
- o Mail Outputs are data out to the application code from the DVC750 and are used to send selected DVC750 I/O status information to the application code for use as needed.

Mail Inputs / Mail Output names:
Range: 16 Characters with no spaces. Usable characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid.

DVC750 Program Variables
The Input and Output variables are very similar to the DVC707 / DVC710 output group variables. To access these variables in the Bubble logic, use the format "*ModuleName.VarName*" . For example the proper variable name for an Analog/Pulse Input named "Line_Speed" set up as a RPM Pulse Input on a DVC750 named "Winch_Control" would be "Winch_Control. Line_Speed.RealRPM".

Remember: The Output Groups PWM command must be controlled with the Mail In.

DVC750 Program Variables		
Name	**Description**	**Range**
DVC750name.Status	Get state of the flag	0 = Online, 2 = Offline
DVC750.ClrMinMax	Reset the MinF and MaxF flags on all analog inputs.	True, False
DVC750.ClrShorts	Reset all Short flags on the DVC750s outputs.	True, False
DVC750.ClrShorts	Reset the Open flags on all the DVC750s outputs.	True, False

| DVC750 Example Code	
Code	**Comments**
DVC750.PWM_1.Dir = Ana_1.Dir	Sets the direction for PWM_1 output on the DVC750 to follow the direction of Ana_1 on the DVC10
DVC750.PWM_1.Feedback = DVC750.Ana_2	Sets the feedback for PWM_1 to Ana_2 on the DVC750
DVC750.ClrShorts = DVC750.Dig_2	Clears any short flags on the DVC750 when DVC750.Dig_2 is active
NOTE: Variable Names assume default module name "DVC750"	

6.4   DVC61

The DVC61 display module is a configurable 4x20 display device with 5 digital inputs. Multiple DVC61s can be supported when connected to the CAN bus while only one DVC61 can be connected on the serial port. To configure a DVC61 module and define its screens, add a DVC61 to the project in the programming tool. Then open the DVC61 screen.

DVC61 Module Configuration:

The Name field is used to identify the DVC61 module within the project. The MAC ID tells the DVC707 / DVC710 how to address the Output Module when communicating over the CAN Bus and must be unique with respect to other modules in the system. Connection Type is used to select between CAN Bus communication and RS232 communication.   Input Set Up is used to configure the five digital inputs.
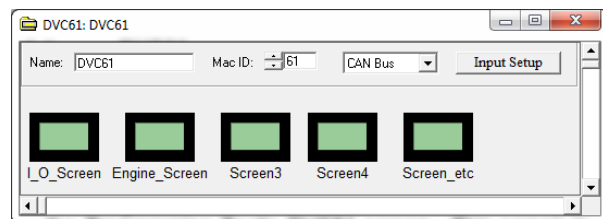
Use the Input Setup button to open the Input Setup screen. Each tab opens a screen for configuring an input. All DVC61 inputs are Tri-State, a Tri-State input can have 3 states, High, Low or Floating. If the input were pulled High (to supply), the inputs "A" bit would be considered True. If the input were pulled Low (to Ground), the inputs "B" bit would be considered True. If the input was neither pulled High nor Low, the input would be considered "Floating" and both input bits would answer False when tested. Each input can be enabled or disabled and contains standard features of digital inputs on other modules.

DVC61 Digital Input Setup
The DVC61 Input Setup screen allows the user to configure the 5 digital inputs of the DVC61. Select the tabs at the top of the screen to switch between the five inputs. The diagrams displayed show possible hardware configurations for the inputs.

Input 1A / 1B check box
Select to enable the input in the project.

Name
The name used in the bubble logic screen to access this variable and properties.
Range: 16 Characters with no spaces. Usable characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number, compiler keyword or another name already being used.

Debounce Time
The amount time in of milliseconds to wait before accepting a change in input state
Range: 0 to 9990ms in 10ms Increments

Software Toggle (Software Latch)
In Toggle Mode, the rising or falling of the digital input (with respect to De-Bounce and polarity Active High or Low) will reverse the state of the variable with each valid input pulse, latching the input variable at each occurrence. In No Toggle Mode, the input responds to the voltage level at the input at all times again with respect to De-Bounce time.
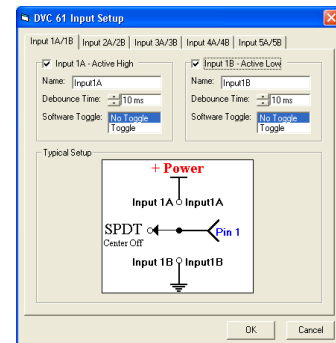
Range: Toggle, No Toggle

DVC61 Screen Definitions

To define a screen for the DVC61, open the DCV61s icon in the project window and then right click in the open area in the DVC61 Setup Screen and select Add Screen. When using multiple DVC61s in an application, all screen definitions for all DVC61s may be defined inside a single DVC61s Setup Screen. This is not only covenant but helps prevent duplication of screens in a project.

Configuration screens have fields for its name, supporting text and variable scaling / location definitions. There is also a representation of what the output screen will look like. Each defined DVC61 screen may display up to 12 program variables.

Name:
The name used in the bubble logic to access this screen and its properties.
Range: 16 alpha/numeric characters only with no spaces.
Rules: The first character cannot be a number, compiler keyword or another names already being used. Usable characters are A-Z, a-z, 0-9, and "_".

V1, … ,V12 Pull Down Menu of variable scaling formats. See below for an explanation of each selection.

X & Y These are the X and Y numeric text character position and line coordinates for the Display Variables. The Upper left corner is X, 1 Y, 1.

Test Value This is the value displayed on the test screen to allow the user to preview the finished screen while in the programming tool.

DVC61 Program Variables		
Name	**Description**	**Range**
DVC61.Status	Get state of the flag	0 = Online, 2 = Offline
DVC61.Screen	Name used to point to a defined screen to display	
DVC61.v1 through DVC61.v12	Name used to point to a defined variable to display on the current screen	
DVC61.BackLight	Set / Get the Back Light set point for the DVC61	0% to 100% (0 to 1023)
DVC61.Contrast	Set / Get the Contrast set point for the DVC61	0% to 100% (0 to 1023)
DVC61.InputName	Get the state of the input	True, False
NOTE: Variable Names assume default module name "DVC61"		

DVC61 Sample Code	
Code	**Comments**
DVC61.Screen = Screen1	Point to DVC61 Screen image to display
DVC61.v1 = ana_1	Screen Variable "V1" is set to display analog input 1's value
DVC61.BackLight = 100%	Turn on the back light at 100% brightness

Display Variable Scaling Definitions	
None	Variable not used
0 to 100%	Scales the displayed variable to a percentage of 0 to 1023. May be used with any 10 bit variable.
5 Digit Uint	Displays the variable in its raw format.
Supply V	Scales the displayed variable to units of supply voltage.
T, F	Displays an "F" for 0 and "T" for anything else
On, Off	Displays an "Off" for 0 and "On" for anything else
True, False	Displays an "False" for 0 and "True" for anything else
Up, Down	Displays an "Down" for 0 and "Up" for anything else
Fwd, Rev	Displays an "Rev" for 0 and "Fwd" for anything else
Left, Right	Displays an "Right" for 0 and "Left" for anything else
Remote, Ground	Displays an "Ground" for 0 and "Remote" for anything else
Fwd/Rev, Stop	Displays an "Stop" for 0 and "Fwd/Rev" for anything else
Yes, No	Displays an "No" for 0 and "Yes" for anything else
0.1	Displays 5 Digit Uint with a decimal point before the ones Colum
0.01(s)	Displays 5 Digit Uint with a decimal point before the tens Colum (also scales timers to seconds)
0.001	Displays 5 Digit Uint with a decimal point before the hundreds Colum
0.0001	Displays 5 Digit Uint with a decimal point before the thousands Colum
String	Displays a defined string from the application code

6.5   J1939

The DVC710 controller has two separate CAN ports. CAN Port 1 (connector B1 & B2) is capable of both transmitting and receiving both J1939 messages and DVC Devicenet messages. This port may be configured to use either protocol exclusively or both protocols simultaneously. CAN Port 2 (connector C1 & D1) is compatible for use with the J1939 protocol only. It is suggested that the second port be used for high bandwidth low priority messages. (e.g. to separate a J1939 Display from the main bus in order to reduce bandwidth demand on the main bus).

The DVC707 controller has one CAN Port. This port has the capability to transmit and receive both J1939 messages and DVC Devicenet messages but may employ only one protocol at a time. See below for differences in programming the DVC707 J1939 messages;

1. The DVC707 uses the prefix **DVC80** rather than **J1939** for addressing the ".NORSP" state.
2. Multi module systems that require J1939 communications should be designed using the DVC710.
3. The DVC707 does not support the following J1939 Message Features;
    a. Message Disable
    b. CAN Bus 2
    c. Specific Source Address

J1939 Device
To configure J1939 messages, the application program must have a J1939 device in the project (DVC80 for DVC707 Projects). J1939 Message screens are used to set up both send and receive messages for the project. The MAC ID on the J1939 setup screen is not used in this configuration. The Source Address on the J1939 setup screen is the default Source Address that will be attached to messages sent from the DVC707 / DVC710 unless a Specific Source Address is configured in the individual message screens.

DVC707 / DVC710 Module Configuration
With the Program Loader Monitor running and connected to the DVC master module, navigate to the Factory Information screen and select any combination of the following available options then select "Send Changes". (A Password Level of 3 is required to configure Factory Settings.)
- CAN Baud Rate = 125K, 250K or 500K baud (J1939 Runs at 250K baud)
- HCT-CAN on CAN 1 (DVC Modules or J1939, DVC707 only)
- J1939 on CAN 1
- J1939 on CAN 2

DVC J1939 Application Programming
The J1939 layout screen is similar to that of the DVC61 or Virtual Display. New J1939 messages are added by clicking the right mouse button on the screen and selecting "Add Message". The DVC707 / DVC710 can process hundreds of separate messages but keep in mind that each message costs both program memory as well as device memory.
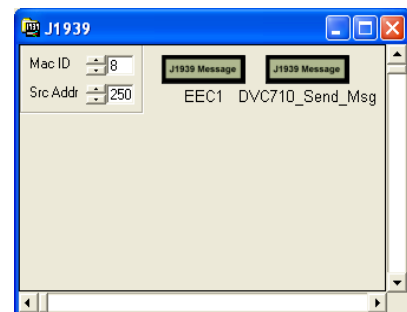
J1939 Message Set-up

The Programming tool does not contain a library of pre-defined J1939 messages however; preconfigured messages may be copied and pasted from other applications. Typically, SPN and PGN definitions may be found in SAE J1939 – 71 or the datasheet of the device that is being used. To add a blank message that can be configured, right click in the J1939 screen and select Add Message.

Command Name (Message Name)
The prefix used in the application code to access this message's data and status.
Range: 16 Alpha/Numeric characters only with no spaces.

Control
The Control field specifies if the message will be sent to or received from/to the DVC.
Range: Receive Data or Send Data

Maximum Time
The meaning of this field is dependent on the Control type. If the Control type is Send Data then this represents the period for message transmission from the DVC master controller. If the Control type is Receive Data then this field represents no response timeout period for the message. It is recommended that the user set the Maximum Time for receive messages to at least three times the expected transmission rate.
Range: 10ms to 10seconds.

PDU Format (PF)
The PDU Format field identifies one of two PDU formats able to be transmitted (PDU1 or PDU2). PDU Formats are described in the SAE J1939/21, Section 3.3.
Range: 0 to 255

PDU Specific (PS)
The meaning of this field is dependent on the value of PF. If the PF value is between 0 and 239 (PDU1), this PS field contains a destination address. If the PF field is between 240 and 255 (PDU2), the PS field contains a Group Extension (GE). The Group Extension provides a larger set of values to identify messages, which can be broadcast to all ECUs on the network.
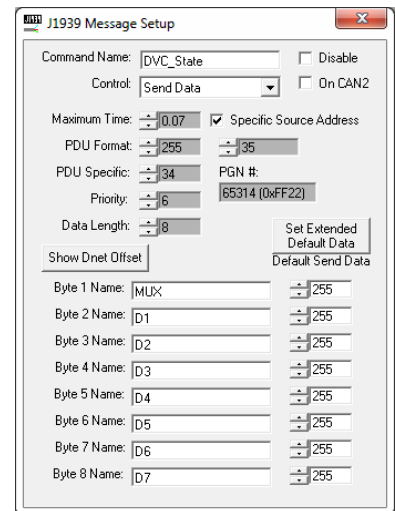**Range:** 0 to 255

Specific Source Address Check Box
Select to assign a specific source address to a message. If not checked, outgoing messages will contain the Default Source Address from the J1939 Main Screen and incoming messages will not be filtered for a source address. When checked, a field will open to allow the programmer to enter a specific source address.
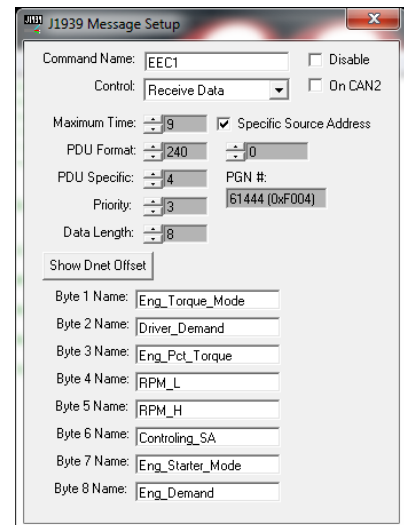
Priority
The Priority field assigns how important the message is to be processed. The highest priority is 0 and lowest priority is 7. (This field is not needed when "Control" type is "Receive Data")
Range: 0 to 7

Data Length
Data Length is the length of the message in 8-bit bytes.
Range: 1 to 8

Byte # Name
Byte Name is the suffix of the variable name to access a byte in application code. For instance, "EEC1.Eng_Torque_Mode" would be used to access byte 1 of the defined EEC1 message shown above.
Range: 16 Alpha/Numeric characters only with no spaces.

J1939 Message Program Variables		
Name	**Description**	**Range**
Name.ByteName	Get / Set current value of byte	0 to 255
J1939.Name.NORSP [1,2]	Get the state of the no-response flag	True, False True when a message has not been received in the "Maximum Time" period
J1939.Name.Disable [1,3]	Used to enable / disable message.	True, False

1. Requires prefix "J1939"
2. The DVC707 uses the prefix DVC80 rather than J1939 for addressing the "NORSP" state.
3. The DVC707 does not support the following J1939 Message Features;
    a. Message Disable
    b. CAN Bus 2
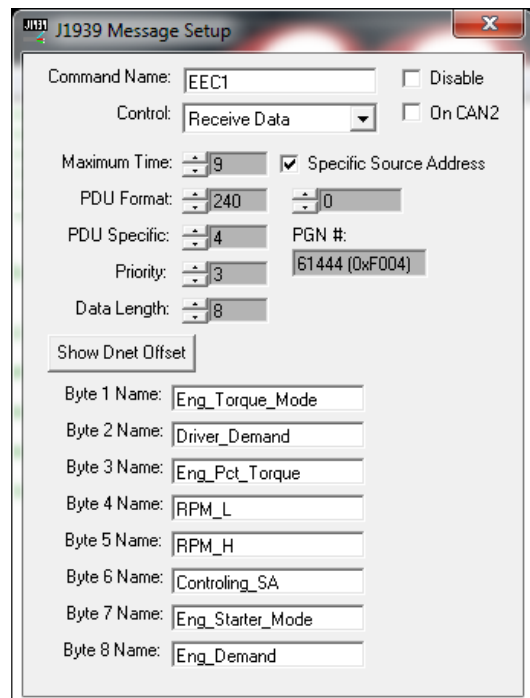   c. Specific Source Address

Message Example

This example will test the validity of the engine RPM message from EEC1 and if valid, set the engineRPM variable to the scaled real engine RPM or if not valid, reset the engineRPM variable to 0 and the error_status variable to 1.

```
dim engineRPM as uint
dim error_status as uint

if (j1939.eec1.norsp) then
   engineRPM = 0
   error_status = 1
else
   engineRPM = ((eec1.rpm_h * 256) + eec1.rpm_l) / 8
end if
```

Note: The Engine RPM example converts the two 8-bit RPM bytes into one 16-bit number, and then scales it from 0.125 RPM per bit to 1 RPM per bit.
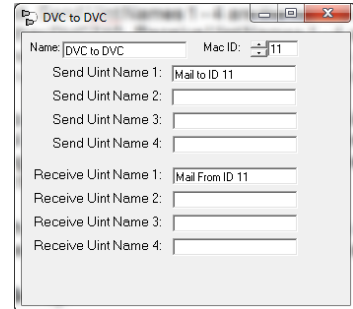

J1939 Message Setup

Command Name: EEC1   ☐ Disable
Control: Receive Data   ☐ On CAN2
Maximum Time: 9   ☑ Specific Source Address
PDU Format: 240   0
PDU Specific: 4   PGN #:
Priority: 3   61444 (0xF004)
Data Length: 8
Show Dnet Offset

Byte 1 Name: Eng_Torque_Mode
Byte 2 Name: Driver_Demand
Byte 3 Name: Eng_Pct_Torque
Byte 4 Name: RPM_L
Byte 5 Name: RPM_H
Byte 6 Name: Controling_SA
Byte 7 Name: Eng_Starter_Mode
Byte 8 Name: Eng_Demand

6.6 SAE J1939 Message Examples

SAE J1939 Description	Parameter	PDU2 (PF), (PS)	PGN Decimal	PGN Hex	Start Byte	Data Length	Value	Units	Offset
Elec Eng Cont #2 – EEC2	Accelerator Pedal Position	240, 3	61443	F003	2	1	0.4	%/bit	0
	Percent Load at Current Speed	240, 3	61443	F003	3	1	1	%/bit	0
Elec Eng Cont #1 – EEC1	Actual Engine % torque	240, 4	61444	F004	3	1	1	%/bit	-125
	Engine Speed	240, 4	61444	F004	4	2	0.125	Rpm/bit	0
Vehicle Distance	Trip Distance	254, 224	65248	FEE0	1	4	0.125	km/bit	0
	Total Vehicle Distance	254, 224	65248	FEE0	5	4	0.125	km/bit	0
Engine Hours, Revolutions	Total Engine Hours	254, 229	65253	FEE5	1	4	0.05	H/bit	0
Fuel Consumption	Trip Fuel	254, 233	65257	FEE9	1	4	0.5	L/bit	0
	Total Fuel Used	254, 233	65257	FEE9	5	4	0.5	L/bit	0
Engine Temperature	Engine Coolant Temp	254, 238	65262	FEEE	1	1	1	C/bit	-40
	Fuel Temperature	254, 238	65262	FEEE	2	1	1	C/bit	-40
	Engine Oil Temperature	254, 238	65262	FEEE	3	2	0.033125	C/bit	-273
	Engine Intercooler Temperature	254, 238	65262	FEEE	7	1	1	C/bit	-40
Engine Fluid / Level / Pressure	Fuel Delivery Pressure	254, 239	65263	FEEF	1	1	4	kPa/bit	0
	Engine Oil Level	254, 239	65263	FEEF	3	1	0.4	%/bit	0
	Engine Oil Pressure	254, 239	65263	FEEF	4	1	4	kPa/bit	0
	Coolant Pressure	254, 239	65263	FEEF	7	1	2	kPa/bit	0
	Coolant Level	254, 239	65263	FEEF	8	1	0.4	%/bit	0
Cruise Control/ Vehicle Speed	Wheel Based Vehicle Speed	254, 241	65265	FEF1	2	2	1/256	Km/h/bit	0
Fuel Economy	Fuel Rate	254, 242	65266	FEF2	1	2	0.05	L/h/bit	0
	Instantaneous Fuel Economy	254, 242	65266	FEF2	3	2	1/512	km/L/bit	0
	Average Fuel Economy	254, 242	65266	FEF2	5	2	1/512	km/L/bit	0
Inlet/Exhaust Conditions	Boost Pressure	254, 246	65270	FEF6	2	1	2	kPa/bit	0
	Intake Manifold Temp	254, 246	65270	FEF6	3	1	1	C/bit	-40
Inlet/Exhaust Conditions	Air Filter Differential Pressure	254, 246	65270	FEF6	5	1	0.05	kPa/bit	0
Vehicle Electrical Power	Exhaust Gas Temperature	254, 246	65270	FEF6	6	2	0.03125	C/bit	-273
	Electrical Potential (Voltage)	254, 247	65271	FEF7	5	2	0.05	V/bit	0
	Battery Pot. Voltage (Switched)	254, 247	65271	FEF7	7	2	0.05	V/bit	0
Transmisson Fluids	Transmission Oil Pressure	254, 248	65272	FEF8	4	1	16	kPa/bit	0
	Transmission Oil Temperature	254, 248	65272	FEF8	5	2	0.03125	C/bit	-273
Engine Fluid Level/Pressure	Injector Metering Rail 1 Pres	254, 219	65243	FEDB	3	2	1/256	MPa/bit	0
	Injector Metering Rail2 Pres	254, 219	65243	FEDB	7	2	1/256	MPa/bit	0

6.7 DVC Master to DVC Master

Multiple DVC master modules can talk to one another over the CAN Bus using DVC Devicenet. The DVC Master to DVC Master Configuration screen has a MAC ID, four Send Uint (mail) Names, and four Receive Uint (mail) Names. Mailboxes may contain up to 16 bits of information (or two bytes).

The MAC ID field is the MAC ID of the DVC Master module to whom this DVC Master module wishes to communicate with. Send Mail Names 1 - 4 are the names used to control the data to be sent to the other DVC Master module. Receive Mail Names 1 - 4 are the names used to access the values received from the other DVC Master module.

When more than 8 bytes of information needs to be transferred from one module to another, one mailbox may contain a MUX variable and the programmer could send or receive multiple packets of information between the two modules. DVC Master modules could also use J1939 messages to transmit information between each other.

Configuration Set-up
Name:
The prefix name for the send and receive variables for this DVC Master module to use.
Range: 16 characters

MAC ID:
The MAC ID of the DVC Master module to be communicated with.
Range: 0 to 63.

Send Uint Name 1 - 4:
This is the variable suffix Name for sending the data to the other DVC Master module.
Range: 16 Characters with no spaces. Usable characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid.

Receive Uint Name 1 - 4:
This is the variable suffix Name for the data sent from the other DVC Master module.
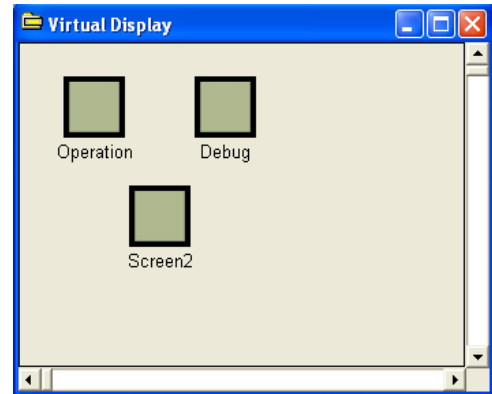**Range:** 16 Characters with no spaces. Usable characters are A-Z, a-z, 0-9, and "_".
Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid.

DVC Master to DVC Master Program Variables		
Name	**Description**	**Range**
DVCtoDVCName.Status	Get / Set current status of the DVC being communicated with	0 =Online, 2 = Offline
DVCtoDVCName.mailboxName	Test or set the two byte variable for the mailbox	0 to 65535

DVC to DVC Sample Code	
Code	**Comments**
DVCtoDVC.Mail_Out1 = 0xFEF1	Set Mail Output variable to 65265 (hex notation)
If (DVCtoDVC.Mail_In1 AND 0x80) then	Test the state of the Mail Input message bit 8

6.8 Virtual Display

The Virtual Display is a debug tool that allows DVC710 users to monitor program variables using a Windows PC or laptop computer. You can monitor up to 20 variables on each screen. The Virtual Display may be used to log data during program operation. To ensure the maximum system bandwidth possible for an application, it is good practice that the Virtual Display and supporting code be removed prior to the production release of an application if the application does not normally use it. This is because that even when the PLM is not attached to the DVC, the Virtual display code is still running and using system resources.

Add the Virtual Display to your project by right clicking the mouse in the project window and selecting the Add Virtual Display option. Only one Virtual Display can be added to a project but the Virtual Display may have multiple screen definitions. Double click the virtual display icon and right click in the Virtual Display window to Add screen definitions. Double click the screen icon to configure the screen's display attributes.

Virtual Display screens are defined and programmed the same way as DVC61 screens and programming. All Screens and variables are accessed with the prefix "VirtualDisplay" See the section about DVC61s in this manual for more information.

Virtual Display Screen

The Virtual Display configuration screen is like the DVC61 configuration screen except that the Virtual Display can display up to 20 variables at the same time.

Name:

The name used in the program logic to display this screen and its variables.

**Range:** 16 Alpha/Numeric characters only with no spaces.

Rules: The first character cannot be a number. Compiler Keywords or other Names already in use are not valid

V1, V2, V3 … V20:

V1, … ,V12 Pull Down Menu of variable scaling formats. See Display Variable Scaling Definitions for an explanation of each selection.

X & Y These are the X and Y numeric text character position and line coordinates for the Display Variables. The Upper left corner is X, 1 Y, 1.

Test Value This is the value displayed on the test screen to allow the user to preview the finished screen while in the programming tool.
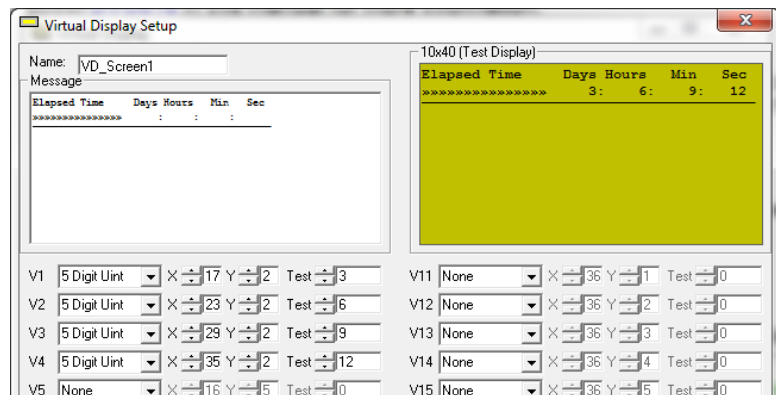
Virtual Display Program Variables		
Name	**Description**	**Range**
virtualdisplay.Screen	Name used to point to a defined screen to display	
virtualdisplay.v1 through virtualdisplay.v12	Name used to point to a defined variable to display on the current screen	

DVC61 Sample Code	
Code	**Comments**
virtualdisplay.Screen = Screen1	Point to the Virtual Display Screen image to display
virtualdisplay.v1 = ana_1	Screen Variable "V1" is set to display analog input 1's value

6.9 Virtual Display Data Logging Feature

Data displayed in the first two columns of the Virtual Display may be logged to a common delimiter text file (.CSV) and may be opened as a spreadsheet in any editor that recognizes the .CSV format. When formatting a Virtual Display Screen for use as a data logger, please observe the following formatting rules. This will ensure that the data collected is placed in a useable format within the .CSV file.

1. There may not be any spaces within a text string either on the screen as fixed text or written to the screen as a string. I.E. "Analog Input Raw Volts" should be written as "Analog_Input_Raw_Volts"

2. There must be a space between a label and its variable. I.E. "Analog_Input_Raw_Volts 65535" where 65535 is the 5 digit unsigned integer assigned to the label.

3. There must be a space between the column one variable and the column 2 label.

4. The operator will be able to select the logging rate through the PLM. The speed that each application can log reliably is different for each application. It is recommended that 1 second be used as the maximum data storage rate however the user may set the logging speed as fast as 1mS.
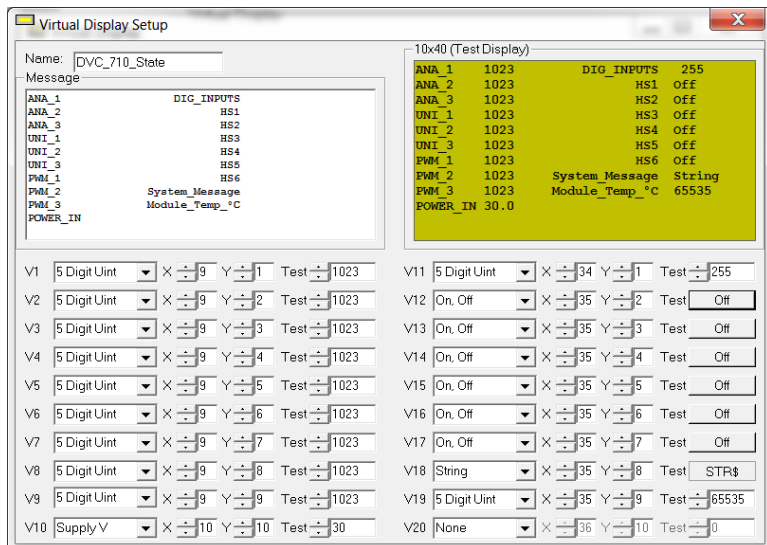
Virtual Display Screen Formatted for use as a Data Logger
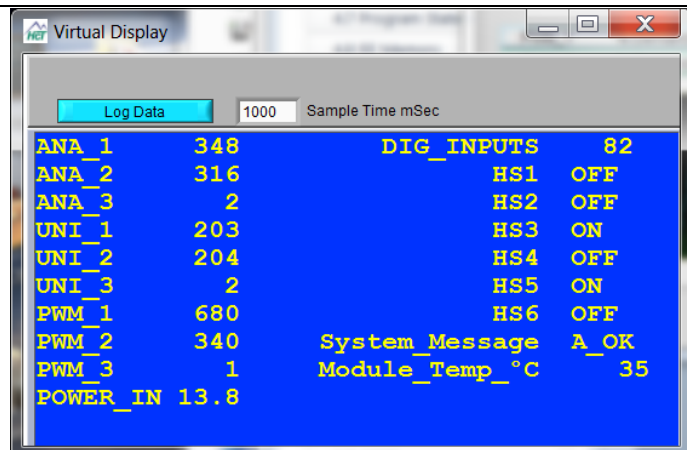
UNI_1	UNI_2	UNI_3	PWM_1	PWM_2	PWM_3	POWER_IN	DIG_INPUTS	HS1	HS2	HS3	HS4	HS5	HS6
203	204	1	354	694	1011	12.9	229	OFF	ON	ON	ON	OFF	OFF
204	204	2	17	357	698	13.6	225	OFF	ON	OFF	OFF	ON	OFF
48	204	1	25	315	656	13.5	222	ON	OFF	ON	OFF	ON	OFF
0	204	2	78	262	603	13.9	221	ON	ON	ON	OFF	ON	OFF
0	204	2	130	210	551	13.8	220	ON	ON	ON	OF	ON	OFF
203	204	2	183	157	498	13.4	220	OFF	OFF	OFF	ON	ON	OFF
81	204	2	225	115	456	13.5	219	ON	OFF	OFF	ON	ON	OFF
203	204	1	277	63	404	13.9	218	ON	ON	OFF	ON	ON	OFF
204	204	2	330	10	351	13.8	217	ON	ON	OFF	ON	ON	OFF

Example .CSV file Data from Data Logger Feature

6.10   Application Simulator

The Application Simulator is a useful development tool used for debugging your DVC707 / DVC710 application without physically being on the target machine. The Application Simulator will simulate the I/O for only the master module.

All of the inputs and outputs of the DVC707 / DVC710 can be simulated using PLM simulation screen. The names defined for I/O in the application are displayed on the simulation screen for easy reference.

To use the simulator you simply add it to your project using the DVC Programming Tool by right clicking

Program Loader Monitor Application Input / Output Simulator

in the Project window, selecting Application Simulator from the popup menu then compiling the application. No program code changes are required. To remove the simulator from a project, simply delete its icon from the project window and recompile.

After including the simulator icon in your project and compiling, you load your application into the DVC707/710 using the Program Loader Monitor. The Program Loader Monitor will present a simulation device in its main window. Click on the status button to display the simulation window shown above.

7 Program Loader Monitor
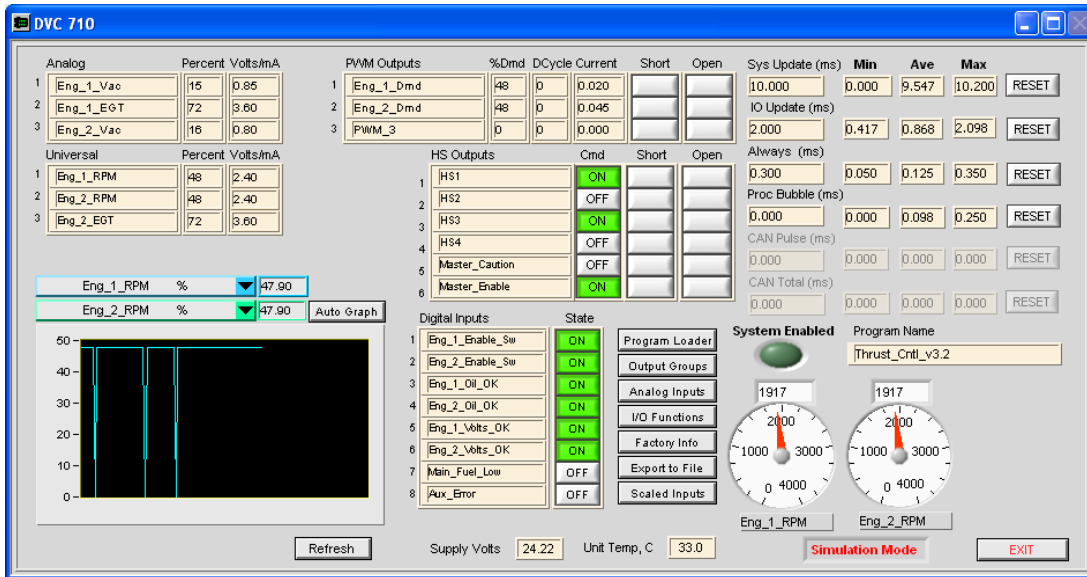
7.1 Introduction

The Program Loader Monitor is used to download programs to the DVC707/710 and to display information from all of the DVC modules connected together via the CAN Bus. It runs on a Windows PC and uses a RS232 cable to communicate with the DVC Modules. Data from DVC expansion modules (i.e. DVC721, DVC741 etc.) is transmitted through the DVC Master module to the Program Loader Monitor. Program Loader Monitors specific to an expansion module are provided and are used to examine or change the module's MAC ID and CAN Bus baud rate.

7.2 Connecting to the DVC707/710

Locate an open serial communications port on the PC. Plug one end of the DVC RS232 serial cable into the serial port (use a USB to Serial Converter if needed) on the computer. Plug the other end into the DVC serial cable weather pack connector on the DVC module.

NOTE: If the RS232 serial cable is connected and the Program Loader Monitor program is not running, the DVC Master module may enter programming mode on power up. If this happens, to exit programming mode, either unplug the serial cable and cycle power or start the Program Loader Monitor program and cycle power.

7.3 Starting the Program Loader Monitor

From Windows press the "Start" button and then select the file
All Programs\HCT Products\Intella 700\PLM5.*x*

If the PLM cannot find the correct com port, select it in the find DVC screen.
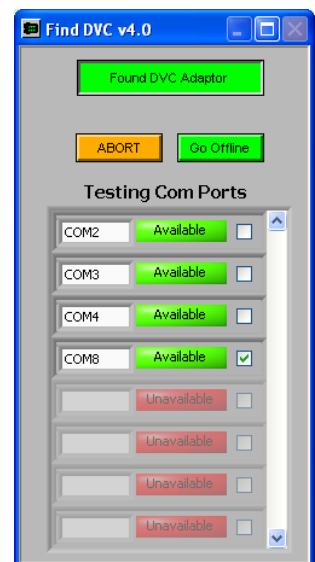
7.4   Main Program Loader Monitor Screen

This is the PLM's Main Screen. This window shows all of the modules in your application's project.   Each device in the project has a status button. Device Status buttons that are grayed out indicate that the DVC Master controller is not communicating with that module. Generally this is due to the devices having different CAN Bus baud rates, incorrect MAC IDs or simply a wiring issue.

To monitor a particular component click on its Device Status button.

A password may be required to gain access to certain features of the Program Loader Monitor.   The passwords are assigned using the Programming Tool. There are four password levels:

Program Loader Monitor
DVC710 Main Screen

Level 0 – If the system is password protected and a valid password has not been entered level 0 is assigned. This level allows you to view information about the system but not make any changes.

Level 1 – Access to I/O and system parameters and the ability to send changes.

Level 2 – Level 1 privileges and the ability to download application programs.

Level 3 – Level 2 Privileges and the ability to change MAC ID, CAN Bus baud rate or download BIOS programs.

DVC Master Display

The Program Loader Monitor is used to observe DVC707 / DVC710 actual input and output information while the application is running. The I/O names assigned in the Programming Tool are used as labels on the screen. The graph feature will plot two variables that may be selected from the pull down menus above the graph.   The PLM is updated as the lowest priority in the application so the while the graph is a good indication of what is happening on the I/O, it should not be used as an Oscilloscope. Dials will appear automatically as needed to display RPM or Counters when pulse inputs are assigned.


Program Loader Monitor on a DVC710 Master module

The Program Loader Monitor determines the DVC controller type when it is connected automatically and the main screen is modified to reflect the input outputs of that controller.

7.5 Program Loader

Use the Program Loader to download new software into the DVC master modules. The Program Loader Button is located at the top of the buttons near the center of the Master Display Screen. When selected, the Program Loader configures the serial port to force the DVC master module to go into programming mode on power up. After opening the Program Loader screen and cycling power, the Load Application button will become active.

To load your Application, either select the Load Application button to begin the download or click in the "Program Path" area to select a file then select Load Application. When the program has loaded, you will be prompted to cycle power to the DVC Master Controller again to boot up with the new program. If a new BIOS is required, please consult with HCT support engineers to obtain the BIOS file and a valid password.
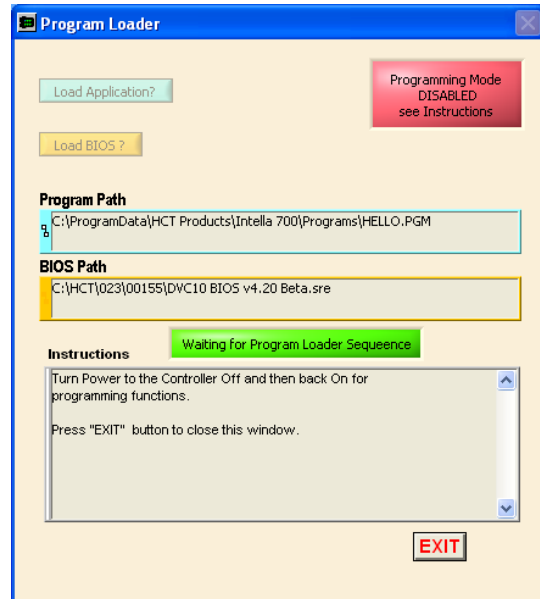
7.6 Output Groups

Click the Output Groups button located directly under the Program Loader button to open this window. The information displayed is basically the same as the information the Programming Tools Output Group configuration window. The Process selection as well as the configuration information that is active for the selected Process and Output Selection can be changed. Settings that may be modified will not be grayed out. After making changes, select Send Changes to update the DVC modules temporary memory (RAM). The DVC will then operates with the temporary values until the unit is reset or powered cycled.
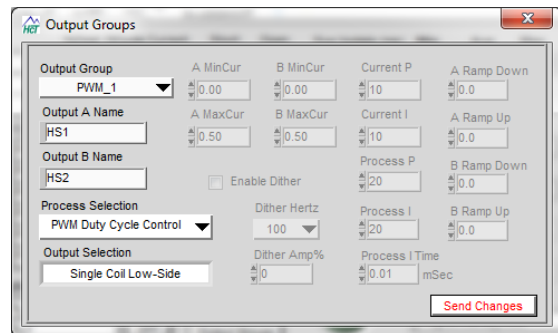
7.7 Analog and Universal Inputs

Click the Analog Inputs button located directly under the Output Groups button to open this window. The information displayed is the same as the information in the DVC Programming Tool. The user can change this information as needed. Again, active settings depending on the configuration will be available. After making changes, select Send Changes to update the DVC modules temporary memory (RAM). The DVC will then operates with the temporary values until the unit is reset or powered cycled.
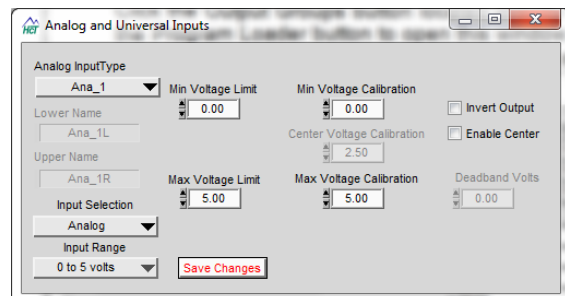
Note: All changes made through the program loader may be saved to a file and imported into the DVC Programming Tool by using the "Export to File" button on the main Program Loader Monitor screen.

7.8 Input / Output Functions

Click the I/O Functions button to open this window. Here, adjustments may be made to the Function Curves while the equipment is being operated. After making changes, select Send Changes to update the DVC modules temporary memory (RAM). The DVC will then operates with the temporary values until the unit is reset or powered cycled.
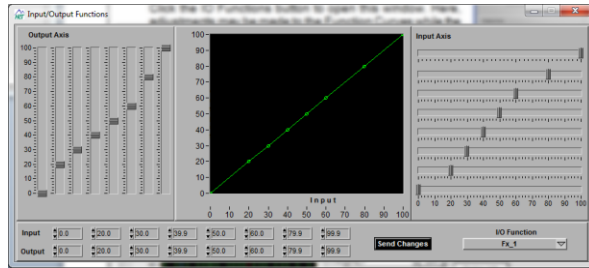
7.9 Factory Information

Click the Factory Info button to open the Factory Information screen. The information displayed is programmed by HCT manufacturing. With the correct Password authorization, the Baud Rate, Mac ID and CAN Bus configuration may be modified.
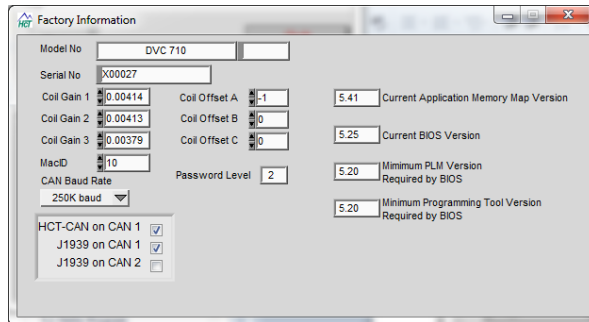
Note: All changes made through the program loader may be saved to a file and imported into the DVC Programming Tool by using the "Export to File" button on the main Program Loader Monitor screen.

7.10 EE Memory

Click on the EE memory Status icon on the Main Program Loader Monitor window to activate this screen. EE memory is non-volatile (it retains its values even when power is turned off). Memory variable values may be set by the operator through the PLM or by the application itself. A Pull Down Menu at the top of each column will allow you to page through the variables. You can change the variables values by selecting and entering a new value or by using the up and down arrows next to the variable's name. These new values can be sent to the DVCs EE memory by selecting the Send Changes button. Additionally, the Save to File button allows the user to save the EE variables and their respective values to a .DAT file on the PC. The Restore from File button allows the user to recall previously saved EE variable values from a .DAT file. The Push to Retrieve button initiates the read memory operation.
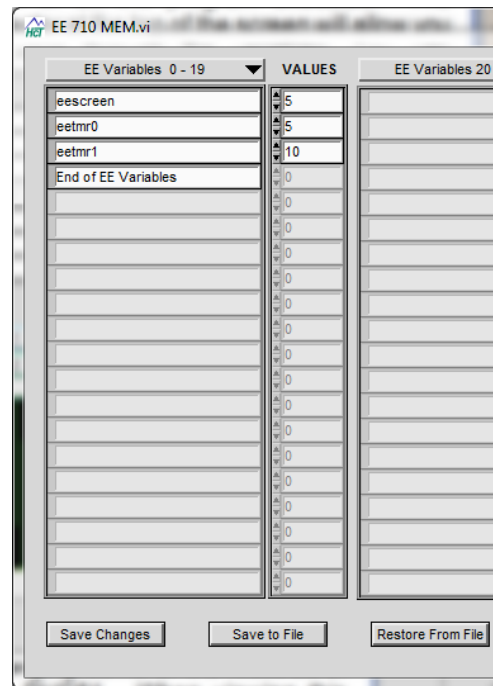
7.11   DVC722, 741 & DVC750 PLM Features

When the Program Loader Monitor is connected directly to the DVC722, DVC741 or DVC750 using an RS232 cable, one of the following screens will appear:

On the DVC722 and DVC741, the user can Modify the MAC ID, and Baud Rate, select "Send Changes" and cycle power for changes to take effect. This is done to match the DVC Master module's baud rate and set the MAC IDs for the project.

On the DVC750, the screen is much like the DVC707 / DVC710's screen and buttons open sub screens for different settings.

When plugged into a master module, each expansion modules screen can still be viewed but no changes can be made to their configurations. Also when plugged into the master module the input labels will contain the names assigned in the programming tool. (These names are not sent to the DVC722 or DVC741 over the CAN bus so they are only available through the master module.)
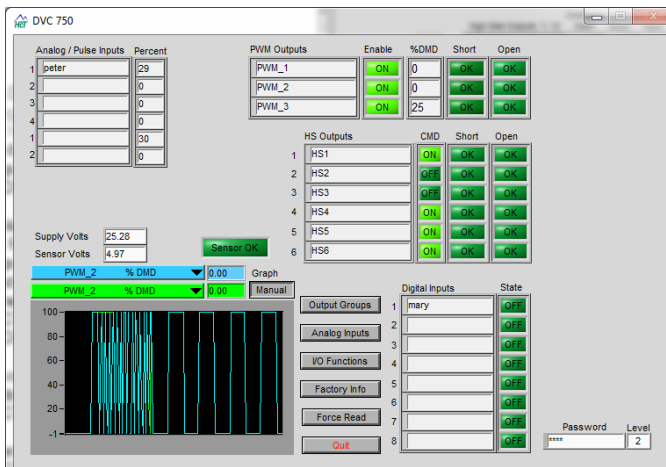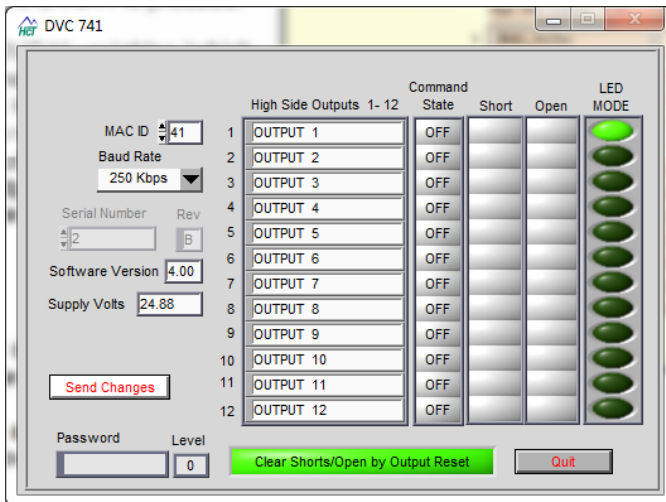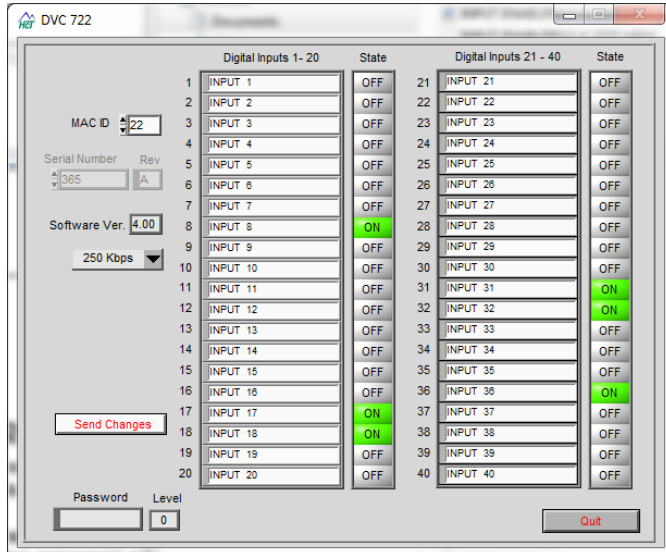
7.12   DVC61 PLM Features

The following screen appears when connected directly to the DVC61. This screen enables the user to monitor the status of the Inputs, Baud rate, Supply Voltage, Serial Number and MAC ID as well as changing some of the features of the DVC61.   When viewing this screen while connected to the DVC Master module, certain features will be "grayed out" because changes cannot be sent to the DVC61 over the CAN bus.
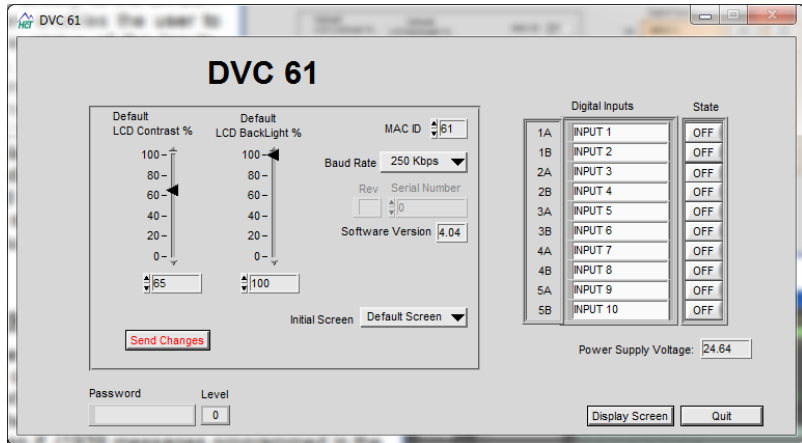
7.13   J1939 PLM Features

This screen appears when the Program Loader Monitor is connected to the DVC707 / DVC710 and the status button next to J1939 on the main menu screen is clicked. The J1939 status button appears on the main PLM Screen if J1939 messages are programmed in the application. The developer or user can monitor the status of J1939 messages that have been programmed in the application. Other J1939 traffic is filtered out and not available on this screen.

Messages
This text box located in the upper left hand corner of the screen indicates the number of messages programmed into the current application.

Coolant Temperature Indicator
This text box located in the upper left hand corner of the screen indicates the coolant temperature in degrees C or F as selected by the switch next to it on the screen, anytime the message 65262 containing SPN 110 is present on the bus.

Message Decoders
There are two separate message decoders at the top of the J1939 screen. The first will display the bit status of the selected message and byte. The second allows the user to apply gain and offset to a message from 1 to 3 bytes of data. For example, the Engine Coolant Temperature data is a 1 byte message with an offset of -40 and a gain of 0, while the EEC1, Engine RPM message is 2 bytes with 0 offset and a gain of 8.

PGN #_SS
This column allows the user to select a PGN number with a specific source address from a pull down list. The pull down list is derived from the messages defined within the application. The specific Source Address for the message is indicated by an under bar (i.e. message; 61444_0 would indicate the EEC1 message from source address 0). Data for this PGN number is displayed in the corresponding row.

PGN Name
This column allows the user to enter a name for the selected PGN on each row.

PGN Data Windows
This column allows the user to see the data for the PGN number for that row. It will change background colors to indicate a good response, timed out response (NoRsp) or a disabled message.

T/R Can
This column allows the user to determine whether the selected PGN number for that row is an incoming or outgoing message from the DVC Master module and what physical bus it is received or transmitted on.

Display Time/Date
This Check Box when selected will append the time and date to each message displayed as it is received.

Log Rate
This setting defines the amount of time in seconds between data samples when logging.

Log Messages to File
Allows for saving the J1939 messages to a .CSV file. When this button is pressed, the user is prompted to create a save file. Then the DVC Master will begin logging messages.

Exit Button
Activating this button will close the J1939 Loader Monitor Screen.

8 Application Notes

8.1   CAN Bus Configuration

DVC modules communicate using the CAN Bus. Each module (including the master DVC707 / DVC710) has an identifying CAN Bus MAC ID number. The two digits number MAC ID of each module in a project must be unique. Default MAC ID numbers are assigned in the Programming Tool for each type of module when the module object is added to the project. If using more than one module of any type, be sure to assign it a unique MAC ID number. Then each physical module in your project must be configured to match its location within the application. The Program Loader Monitor must be connected directly to an individual module's RS232 port in order to configure its MAC ID number.

CAN hardware architecture gives preference to modules with the lowest valued message header. This includes the MAC ID, Priority, PGN and Source Address's. Therefore, systems generating a lot of CAN Bus traffic should assign the lower MAC ID numbers etc. to the most critical modules / messages. A display module for instance could have a high MAC ID number or Priority depending on the system (DVC DeviceNet or J1939).

## 8.2   Driving PVG valves that require a PWM filter (HCT Pn: 999-10293)

To drive a PVG type valve that does not accept PWM using a DVC master module the user must install a PWM Filter (HCT PN: 999-10293). The PWM Filter is designed to install directly in the wire harness. Install the PWM Filter as follows;

Pin Out
- o   Input Voltage (DVC / PVG Valve System Voltage) to pin 1
- o   DVC Sig Com to pin 2
- o   DVC PWM Output pin to pin 3
- o   Output to PVG to pin 4

Mating Connector Parts
- o   Deutsch, DT06-4S – (1ea)
- o   Deutsch, W4S – (1ea)
- o   Deutsch, 0462-201-16141 – Sockets (4ea)
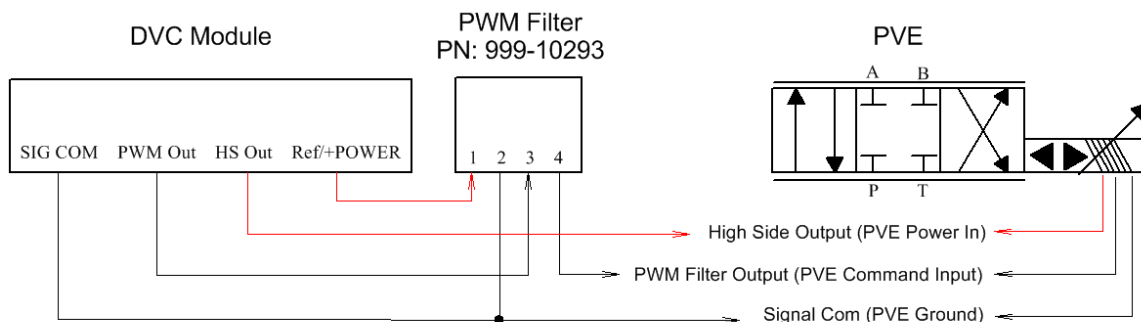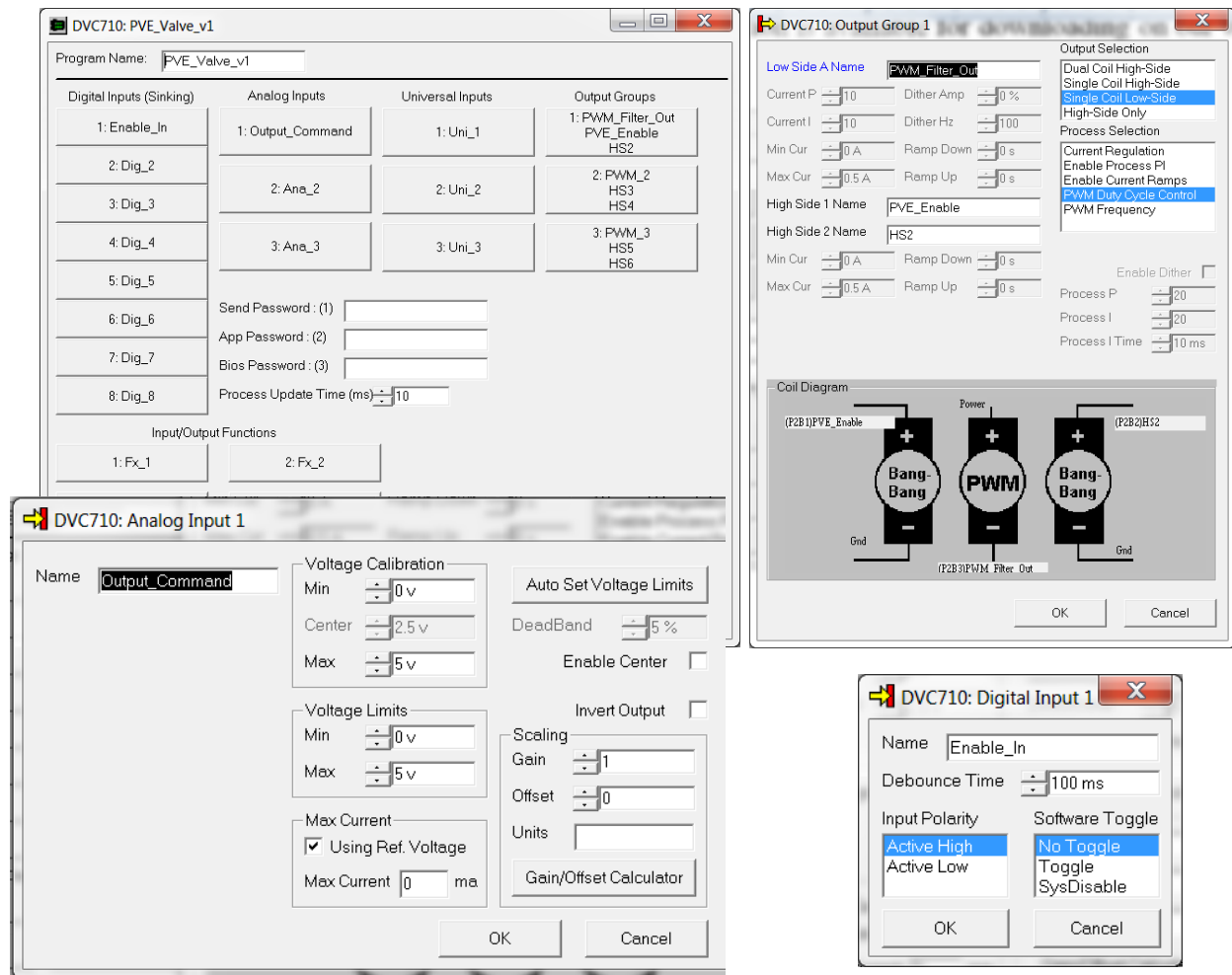
Programming the DVC to Drive the PVE Valve

The following points should be considered when programming the DVC to drive the filter output.

1. When commanding a device such as a PVE type valve using the PWM Filter, the output command to the device will be inversely proportional to the command for the PWM output. Therefore you must invert commands to the output in order to obtain standard directional outputs from the cylinder, motor, etc.
2. To prevent unexpected mechanical operation when initializing a system, enable and set the PWM output to a neutral setting (typically 50%) before enabling (applying power to) the PVE Valve with the HS output.
3. Run the PWM output group in Single Coil High Side, PWM Duty Cycle Mode.
4. When using a High Side Output to provide power to a PVE valve, set the variable *HSname*.opendisable to true to prevent false open detection on the High Side Output.

Sample DVC Code for PVE Valves

This code example includes all considerations listed above as well as a Ramp feature that may be adjusted through EEMEM. Valid settings for the EEMEM variable, Ramp_Scaler are, 0 – 100. The program will automatically clamp this at 100. This corresponds to about 5 seconds per side or 10 seconds end to end.

Module I/O Settings

Always Code

```
Output_Demand = 1023 - Output_Command
PWM_Filter_Out.enable = Enable_In
PVE_Enable.opendisable = 1

if (Enable_In = True) then
   if ((Output_Command < 460) OR (Output_Command > 563)) AND (Input_Ready = 0) then
      PWM_Filter_Out = 512
   else
      if (Output_Demand > PWM_Filter_Out) then
         if (Output_Demand > (PWM_Filter_Out + Ramp_Scaler)) then
            PWM_Filter_Out = PWM_Filter_Out + Ramp_Scaler
         else
            PWM_Filter_Out = Output_Demand
         end if

         if (PWM_Filter_Out > 1023) then
            PWM_Filter_Out = 1023
         end if
      else
         if (Output_Demand < (PWM_Filter_Out - Ramp_Scaler)) then
            if (PWM_Filter_Out > Ramp_Scaler) then
               PWM_Filter_Out = PWM_Filter_Out - Ramp_Scaler
            else
               if (PWM_Filter_Out > 0) then
                  PWM_Filter_Out = PWM_Filter_Out - 1
               end if
            end if
         else
            PWM_Filter_Out = Output_Demand
         end if
      end if

      Input_Ready = 1

   end if
else
   Output_Demand = 512
   Input_Ready = 0
end if

PVE_Enable = Input_Ready

if (Ramp_Scaler > 100) then
   Ramp_Scaler = 100
   eecommand = eewrite
else
   eecommand = 0
end if
'***** Program Variables *****
dim Ramp_Scaler as eemem
dim Input_Ready as uint
dim Output_Demand as uint
```

9 Safety is Everyone's Responsibility

Safe work practices must be observed in building the hardware connections, mounting the units to the machinery, and programming or operating the controllers.

9.1   Safety in building the hardware connections

Safety should be at the forefront of the development team's thoughts. Many times during development, technicians and engineers will fabricate test fixtures, care must be taken not to short circuit power supplies and output devices. Please adhere to all federal, state, and local laws regarding equipment use and safety.

9.2   Safety in mounting the DVC units

DVC modules are very rugged and are built to be mounted near the valves that they are to control and in almost any environment. Care must be taken to locate a safe place on the machine free from excessive heat, and moving parts that may damage the controller's wiring harness or physically harm the controller. Please adhere to all federal, state, and local laws regarding equipment use and safety.

9.3   Safety in programming or operating the controllers

Safety to personnel and machinery must be observed when programming or operating moveable functions. Test program changes before installation on live machinery to minimize safety hazards. When possible don't test a machine at full function and only test program modifications in a controlled environment. Make sure to let other personnel in the area know that a change is being tested and of the possible negative outcomes. Please adhere to all federal, state, and local laws regarding equipment use and safety.

10 Appendix A   Compiler Keywords

ALWAYS, ALWAYSCODE
AI1BITS0, AI2BITS0, AI3BITS0
BACKLIGHTON, BACKLIGHTOFF BLINKCODE
BREG9, BREG8, BREG7, BREG6, BREG5, BREG4, BREG3, BREG2, BREG1, BREG0, BITTEMP
DIGBITS
DVCLED_1, DVCLED_2, DVCLED_3, DVCLED_4
EECOMMAND, EEREAD, EEWRITE
Else, ElseIf, End if
FALSE, FAULTON, FAULTOFF, FAULTBLINK
IF, IFTEST, INIT
K9, K8, K7, K6, K5, K4, K3, K2, K1, K0, KNOKEY, KEND, KCLEAR, KF4, KF3, KF2, KF1, KHOME, KDOWN,
KUP, KENTER, KRIGHT, KLEFT
LEFTBIT
LONGREG0, LONGREG1, LONGREG2, LONGREG3, LONGREG4, LONGREG5, LONGREG6,
LONGREG7, LONGREG8, LONGREG9
MATHTEMP, MATHTEMP2
MS
Not
OFF, ON
OUT1BITS0, OUT2BITS0, OUT3BITS0
RIGHTBIT
S
STATUS_OFFLINE, STATUS_ONLINE, STATUSON, STATUSOFF, STATUSBLINK
SUPPLY
TRUE
Then
UAI1BITS0, UAI2BITS0, UAI3BITS0
WREG0, WREG1, WREG2, WREG3, WREG4, WREG5, WREG6, WREG7, WREG8, WREG9

DVC707 Reserved Variable Names (do not use)

Ana_3
Dig_4, Dig_5, Dig_6, Dig_7, Dig_8

11 Appendix B — Programming Statement Examples

Dim Fault as Uint	Declare "Fault" as a variable for use in the project
Dim Timer_0 as Timer	Declare "Timer_0" as a countdown timer
Dim Scale_Factor as EEmem	Declare "Scale_Factor" as a location in EEmemory
Const Low_Limit = 0x0100	Declare the constant "Low_Limit" to equal 256
PWM_1.Enable = True	Set PWM_1.enable to True
If (Dig_1 AND Dig_2) Then	If Dig_1 and Dig_2 are true then,
PWM_1 = Ana_1 / Scale_Factor	Set the value of PWM_1 to the equation (Analog input 1 divided by the value stored in the EEmemory location "Scale_Factor"
ElseIf (Dig_1 OR Dig_2) Then	If Dig_1 or Dig_2 are true then, set the value of PWM_1 to the Equation (Analog input 1 divided by the value stored in the EEmemory location "Scale_Factor" divided by 2)
PWM_1 = (Ana_1 / Scale_Factor) /2	
ElseIf (Dig_3 XOR Dig_4) Then	If Dig_3 is true exclusive of Dig_4 then, Set the value of PWM_1 to the Equation (Analog input 1 divided by the value stored in the EEmemory location "Scale_Factor" multiplied by 2)
PWM_1 = ((Ana_1 / Scale_Factor) * 2)	
ElseIf (Dig_1 != Dig_5) Then	If Dig_1 is not equal to Dig_5 then,
PWM_1 = 0x0200	Set the value of PWM_1 to 200hex (or 512 or 50%)
Else	If none of the above statements are true then,
PWM_1 = Low_Limit	Set the value of PWM to the constant "Low_Limit"
End If	End of the IF Statement
If (Uni_1 <= 5.5%) Then	If Uni_1 is equal to or less than 5.5% then,
HS1 = True	High-Side 1 equals True
HS2 = False	High-Side 2 equals False
ElseIf ((Uni_1 > 5.5%) AND (Uni_1 < 10%)) Then	If Uni_1 is greater than 5.5% and less than 10% then…
HS1 = False	
HS2 = True	
ElseIf (Uni_1 >= 10.5%) Then	If Uni_1 is equal to or greater than 10.5% Then…
HS1 = True	
HS2 = True	
End If	End of If Statement
HS4 = Uni_1	If Uni_1 is > 0 then, High-Side 4 equals True Else, High-Side 4 equals False
If (Dig_7) Then	If Dig_7 is True then,
Timer_0 = 2.5s	Set Timer_0 to 2.5 seconds
ElseIf (Dig_8) Then	If Dig_8 is True then,
Timer_0 = 500ms	Set Timer_0 to 500 milliseconds
End If	End of If Statement

Note:
To use Hexadecimal notation, use the prefix, 0x. For Example;
255 decimal = 0xFF, 1023 decimal = 0x3FF, 65535 decimal = 0xFFFF

12 Appendix C   Troubleshooting Systems

12.1   Basic Electronics Theory and DVC System Troubleshooting

To many people, Electronic Modules are mystical devices that sometimes must be used despite common sense and better judgment. This is a typical reaction because people do not understand what they can't see. In hydraulics, it is easy to see the results of varying pressure and flow through a device or to calculate the size of a hose needed for a specific function. Electronics and electronic principals are not far removed from hydraulics and hydraulic principals.

All of HCT's products involve electronics and HCT's goal is to make it easy for you to quickly get them to work in your systems. With a basic understanding and some simple tools, electronics can be easily and successfully applied.

Basic Electronics Introduction

In electronics, the basic medium of exchange is the electron. When a whole bunch of electrons get together with a destination in mind it is called voltage. **Voltage** is the "pressure" that causes the electrons to want to move. When the electrons "flow", it is called current. **Current** makes things happen. **Resistance** is the restriction to the flow of current. More voltage (i.e. pressure) will result in more current (i.e. flow) through a device with a given resistance. Current flowing through a conductor causes a magnetic field to surround the conductor. When current flows through a coil surrounding a magnetic material (i.e. spool) the magnetic fields are concentrated and produce a magnetic force. This Magnetic force is proportional to the amount of current flowing through the conductor, the number of windings in the coil and the magnetic material surrounded by the coil. The produced magnetic force can be used to cause valve spool movement. Just like forcing fluid through a restriction generates heat, forcing current through resistance also generates heat. This heat is measured in **Watts**. **Inductance** is the property of a coil to resist changes in current flow. **Capacitance** is the property of a capacitor to resist changes in voltage across its two terminals. A Capacitor might be thought of as an accumulator because more capacitance will cause more "shock absorbing" action to filter voltage (or pressure) spikes.

Useful formulas

E or V = volts, I = amps, P = watts, R = ohms

Voltage
$$E = IR, \ \ E = \frac{P}{I}$$

Current
$$I = \frac{P}{E}, \ I = \frac{E}{R}$$

Resistance
$$R = \frac{E}{I}, \ R = \frac{E^2}{P}$$

Power
$$P = IE, \ \ P = I^2 R$$

Voltage Divider
$$V_{OUT} = E_{IN}(\frac{R_{BOTTOM}}{R_{TOP+} \ R_{BOTTOM}})$$

Best Practices

The easiest way to prevent issues with electronic devices is to prevent them from the start. Following these few simple rules will prevent many issues from happening and will also save countless hours troubleshooting intermittent issues caused by something that was installed in a manner that "Should Work".

1. Make sure that the Power Supply is sufficient for the job. It should be able to carry all loads without sagging.
2. Use Fuses rather than Circuit Breakers and install them as close to the power source as possible.
3. Run dedicated cables both to and from the power supply. Do not rely on chassis ground as a return for the control modules or sensors.
4. Each sensor should have a minimum of two wires coming back to the module, signal in and signal ground.
5. Know your system and prepare accordingly. Some electric vehicles that run on batteries use isolated power supplies to run the electronics that also have isolated grounds. Ensure that all grounds are tied together electrically to prevent ground loops and damage to equipment.
6. Take the time to size each wire and cable to handle the voltage and current load expected. Just like in hydraulics, wires that are too small, like a hoses that are too small cause a voltage (pressure) drop and generate heat. A good rule of thumb is to size cabling for at least 1.5 times the expected load. For Power and Ground wires, two times the expected load won't hurt.
7. Use Twisted Pair Shielded Cables for all sensor and valve wiring especially when the cabling is run together, otherwise sensitive equipment may be affected by the PWM signals.
8. Run power, sensor and PWM cabling separately, when they have to cross, cross them at a 90° angle.
9. Avoid spring loaded terminal blocks and insist on the correct crimping tools for the connectors and pins that you are using.
10. HCT products are rugged and can be mounted on or near the valves that they control. However, mount the electronics away from heat sources such as exhaust manifolds or known heavy electromagnetic fields. Mount on separate mounting plate where warranted to prevent heat conduction from hydraulic components, engine components etc.

Get the entire valve shift you need

The equation $I = \frac{E}{R}$ shows us the relationship between current, voltage and resistance. Here we see that current is directly proportional to voltage and inversely proportional to resistance. Valve coils increase their resistance when they are hot, so the system designer needs to ensure that there is enough overhead voltage from the power supply to fully shift the coils when they are hot (see number one above).

12.2 Trouble shooting the electronics in your system

FIRST: Always pay attention to safety when working on any equipment and do not perform any procedure where you are not sure of all the possible outcomes where machinery may move or someone may become injured.

Some tools needed to troubleshoot your system are;
- o DVM – Digital Volt Meter with Current Mode
- o Jumper wires
- o Oscilloscope – This is not required, but can often save a lot of time as you immediately visualize the signal that you are looking at and will detect noise related issues quicker.
- o PLM – The PLM can monitor all of the modules Inputs and Outputs.

If equipment is not operating correctly and you suspect that there is a problem with the electronics, first check the power supply to the module. If the LED's are off, there is most likely no power to the module. If the LEDs are on, look at the LEDs and determine if an error code is flashing. See LED Indicators. After that, hook up the PLM and see if there are any errors reported on the screen. Look at all the Inputs and Outputs to see if

something is not as expected then follow that path as needed. If everything looks as you would expect, the problem is probably either outside the electrical system or incorrect settings that are not allowing the performance that you are expecting.

When it is determined that there is an issue with an input or an output, check the cabling and connectors for obvious problems. If none are found, use the volt meter to check continuity and voltages to figure out the root cause so you can make your repair.

12.3 Troubleshooting the CAN Bus Communication network

Most common problems with the CAN Bus are a bad connection or improper termination. For DVC expansion modules, Mac ID and Baud Rate would be first on the list for verification. Most equipment that already has a CAN Bus running has terminating resistors pre-installed before adding the DVC system however, if you are having issues, especially intermittently, make sure that the Bus is terminated correctly (see, SAE J1939-11 or ISO 11898). After verifying correct termination, isolate each individual module (just disconnect them from the Bus) one at a time. Often this will help you locate crossed wires, bad connections etc.

Both DVC DeviceNet and J1939 are able to co-exist on the same Bus without interference between the two. However, if the bandwidth for the existing J1939 Bus is already near 100%, you may want to isolate the DVC DeviceNet Bus to ensure consistent operation. To do this, On the DVC710, program all J1939 messages onto CAN 2 and Wire CAN 2 to the J1939 Bus while wiring all the DVC expansion modules on a separate Bus (don't forget to terminate). In the Factory Information screen, select CAN Bus 1 for DVC Expansion Modules and CAN Bus 2 for J1939 Only.

13 Appendix D   Current Regulation using PI techniques

PI, or **P**roportional, **I**ntegral control, is a powerful and popular method of regulating dynamic systems. Typical applications include speed control and position control. HCT uses a PI control function to regulate current through the DVC's proportional outputs. This allows us to compensate for variations in system parameters such as wire lengths, supply voltage, coil temperature etc. that can affect the positioning of a valves spool providing consistent and repeatable results. A basic understanding of how the controller maintains the current feedback PI loop will assist the user in aligning their system for the best possible performance.

The spool in a proportional valve moves due to a magnetic field that is created by current flowing through the valve's coil. When using a DVC, the amount current through the coil is a product of the supply voltage applied to the high-side coil contact, the amount of time (duty cycle) that the voltage is allowed to flow through the coil by the opening and closing of a switch on the low-side of the coil and the total resistance in the complete current path to and from the coil. PI control is a method by which the duty cycle of the applied voltage is varied to achieve the desired (average) current through the coil despite all of a systems dynamic variation that affects current. PI control also provides a means by which this desired current may be reached quickly and accurately. Pulse Width Modulation (PWM) is a term that describes pulsing the system voltage to control the flow of current through the valves coil and is represented as a percentage of Duty Cycle (DYCY) of the applied voltage. PI control increases or decreases the average current (by adjusting the PWM DYCY) with respect to the error between the measured current through the coil and the desired or commanded current.

Therefore, current through the coil can be represented as;

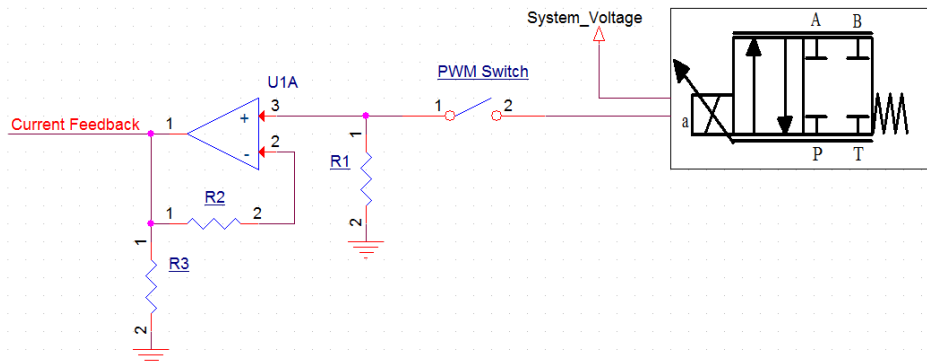$$I = \frac{E * PWM_{DYCY}}{R_{Total}}$$

Where,
I = Current
E = Applied Voltage
PWM_{DYCY} = Percentage of PWM duty cycle
R_{Total} = the total resistance of the circuit including the wiring, connectors and coil.

The basic proportional valve control and feedback circuit looks like the following:

Closed loop control means using a feedback signal to tell the system where it is currently so the system can determine what needs to be done to achieve the current target. Closing the "current loop" allows the DVC to automatically compensate for the dynamic characteristics of your system and is performed on each

proportional output individually. To perform this level of control, feedback is required from the system to tell the microprocessor where the system is so that it will know how much error there is and when it has arrived at the desired target. First, there must be a command from the user to tell the controller where the system should be. The difference between feedback and command is considered error. Because real systems are dynamic, the application of a fixed correction to the same error will not always produce the same result often causing overshooting or undershooting and/or system oscillation. Consider trying to align a satellite dish by having someone yell STOP when the signal is good, you would generally overshoot the optimum position given the time it takes the person to react to observing the good signal and relay the information. Overshoot is going past the desired setting because of a delay in getting the feedback, or a delay in stopping the change in the system. If you see that you have gone past the commanded system state and then reverse the movement to compensate, you can then overshoot in the opposite direction. Multiple overshoots are called oscillation. One solution to the delay problem is to slow down the system's rate of change so much that the delays and inertia are insignificant. This is usually not a good solution and can often end up in undershooting the error and never reaching the target value.

How the system responds to an error is adjustable by setting the **P** and **I** parameters. These settings determine the speed of correction, degree of over shoot and final error. These settings are named "Current P" and "Current I" and are located on the output set up screens in the programming tool and the Program Loader Monitor. They are used in combination to determine the outputs reaction each time the BIOS updates an individual output (these updates are separate and isolated from the logic cycle). The Proportional term (Current P) as its name implies will drive the output proportional to the error causing the system to change more rapidly as the error increases. However, as the error approaches zero, the proportional term also approaches zero and eventually is too small to cause a change in the output without help from the Integral term. Proportional control alone is the simplest to use, but will result in some steady state error (undershoot). Increasing the proportional term enough to limit the steady state error to a small value can cause overshoot and oscillation. The integral term (current P) is the product of error over time and drives the output harder as time in error increases. One way to look at Integral would be as a running average of the error multiplied by a constant. The integral term is typically slower to respond, but will result in an extremely small final error as any detectable error will continue to add up until the output changes in direction to correct it. This term can also cause oscillation if set too high, and will need the help of the Proportional term for fast machine response.

DVC products provide adjustable P and I tuning options for each Output Group on the module. For most applications, the default values work fine. However, some applications require special settings to obtain the best performance. Valve coils with high inductance values or using Dither, especially dither at or below 100 Hz will almost always require alignment of the Current P and I settings. When aligning the Current P and Current I settings, following these simple rules will help the user find the best and most accurate values as quickly as possible;

1. Using an oscilloscope to monitor the current through the coil is the best tool to measure the effects of small changes in the P and I set points. Even when using an oscilloscope, real time system operation should always be the finial acceptance test for any adjustment.
2. Do not use Values above 64.
3. Start by setting both the Current P and Current I to 2, increase the Current P set point until the system reacts with the desired speed.
4. Then adjust the Current I set point up enough to remove any latent error.
5. Keep in mind that using Dither and/or Ramp can affect how the output reacts and may induce false error detection with incorrect P and I set points. Adjust the P and I settings accordingly
6. Always thoroughly test the system after any adjustment to ensure that unintentional behavior has not been induced.

# 14 Appendix E Pulse Width Modulation (PWM) and Dither

HCT's DVC products provide selectable PWM and Dither capabilities for each proportional valve in your system. Our default settings work with most applications however; the user may specify different values when needed. The DVC product's control circuits and internal BIOS automatically control the application of the PWM and Dither signals in accordance with the selected settings programed in the application. If you need to adjust your system, a basic understanding of how PWM and Dither work to control a valve will help you achieve the best possible results.

Current flowing through a valve's coil creates a magnetic field (see Appendix D above). This field provides the force to move the valve's spool thereby adjusting the position of the valve. The voltage across the coil divided by the coil resistance is equal to the coil current. This current is supplied by an external power supply, which generally is a battery. The total circuit is made up of all the components from the power supply's positive terminal through the valve and the DVC and returning to the power supply's negative terminal. The circuit's accumulative resistance due to the connectors, wire lengths/gauge, valve coil, and switches determine the actual current. Very easy so far, but proportional valves are only useful if the current can be changed and controlled.

A potentiometer could be used to vary the resistance in series with a coil and set the coil current to a desired value. Adding resistance to control the valves current is inefficient and not practical. Use of the electronics allow for providing current regulation, Dither, circuit protection features, ramping, and the elimination of dead band. Pulse width modulation (PWM) is an efficient technique for driving current through a valve's coil that allows these features to exist. PWM does not waste any significant power or generate unnecessary heat.

How PWM works

  Coil current is controlled by turning a low resistance switch on and off at the PWM frequency.

Switch Closed
(On)
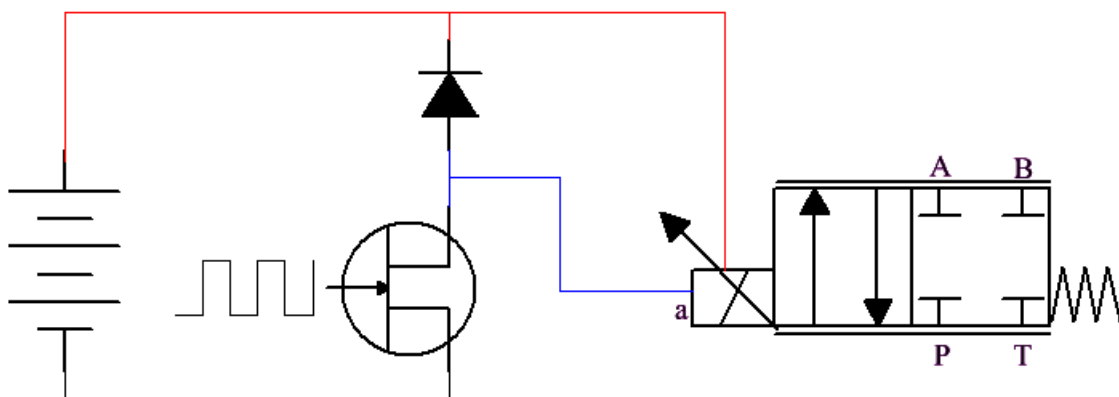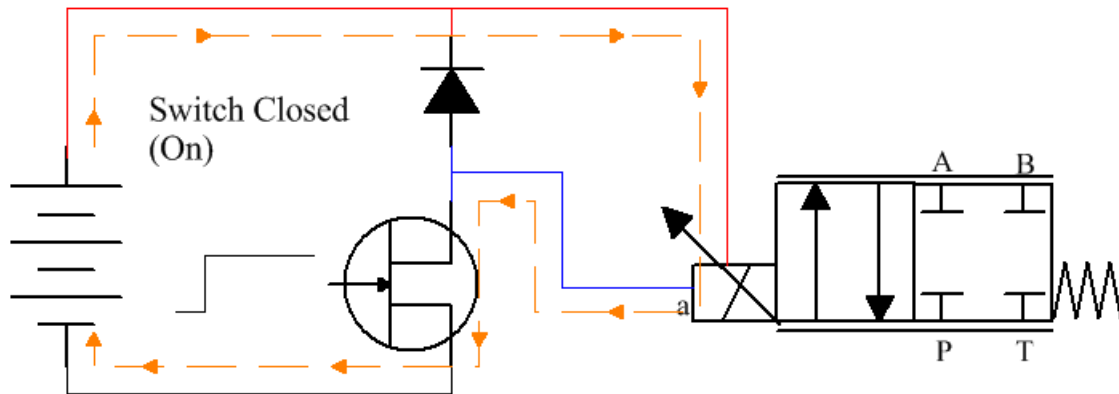
When the coil is on (or charging), the coil's magnetic field stores increasing amounts of energy as the current increases, much as a flywheel stores mechanical energy as the rotational speed increases. Inductance is the measure of the electrical inertia that acts to oppose increasing or decreasing the coil current. PWM takes advantage of this inductive effect by switching the power to the coil on and off. When the valve driver's PWM switch is closed, full power supply voltage appears across the coil and attempts to increase the current flow to the maximum. The coil prevents an instant change in current by appearing to have a larger resistance than it really does. This resistance decreases with time, so the current increases as long as the switch is closed. This continues until either the coil is saturated (has stored all the energy that it can), or the switch is opened. When the valve driver opens the switch, the coil attempts to maintain its current flow using the energy that it has stored up while the PWM switch was closed.

Switch Open
(Off)
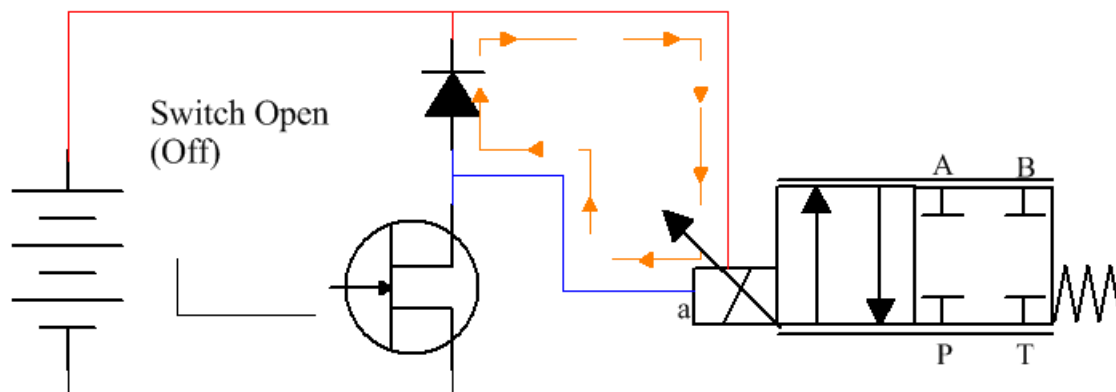
To do this, the coil reverses its voltage polarity, acts as a generator and drives its current through a diode (called a fly back diode). This diode requires a forward voltage of about 0.7 volts to make current flow. This current decreases until the magnetic field that held the energy has collapsed or as long as the switch remains open. The coil current flowing through the diode provides the force required to maintain the position of the valves spool. If the PWM frequency is high enough, the current will remain almost constant. At high PWM frequencies there is not enough time for the current to change much before the switch changes state again, and reverses the last change. The average value of coil current is relative to the PWM duty cycle, i.e. proportional to the time the switch is on compared to the time the switch is off. At 0% duty cycle, the switch would be always open and no coil current would flow. At 100% duty cycle, the switch is always closed and maximum current flows through the coil.

Static Friction is the tendency of the valves spool to want to remain at its current position and is referred to as Stiction. Once Stiction is overcome, friction decreases because the friction of a moving object is less than when it is stationary. Stiction can prevent the valves spool from moving for small control input changes, and then jump too far when the control input changes enough to overcome it. The magnetic force (driven by coil current) required to get the spool to start moving is greater than is required to keep it moving.

Hysteresis is the difference in a valves behavior dependent on whether the valve is opening or closing. Hysteresis can cause the spool shift to be much different for the same valve current depending on whether the valve is opening or closing. The friction of the moving spool (dynamic friction) is much less than static friction, but still resists the current's attempt to move it. The direction the spool is shifting determines where the spool ends up unless we do something to prevent the Hysteresis.
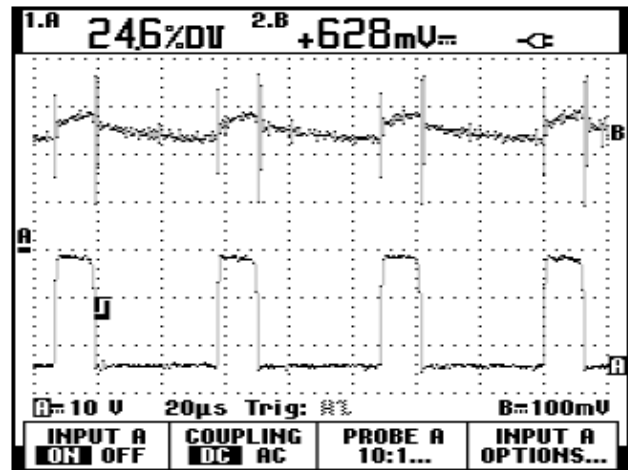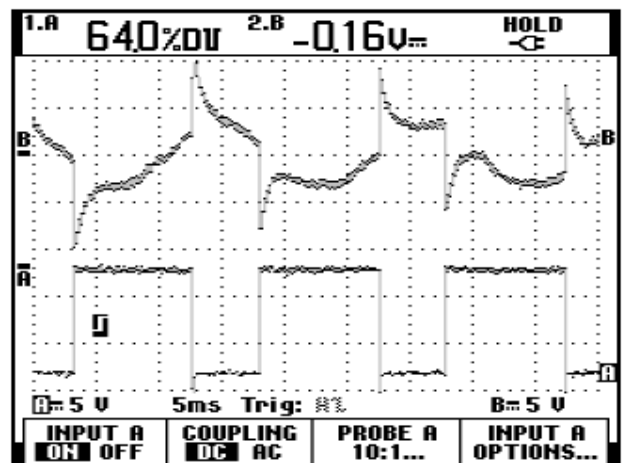
To prevent Stiction and reduce Hysteresis, we need to keep the valves spool moving back and forth enough to keep it from sticking but not enough to cause movement in the device that we are controlling. Valve Dither is a rapid, small movement of the spool about the desired shift point. To be effective, Dither must be large in amplitude and slow enough in frequency to cause the spool to move yet small enough in amplitude and fast enough in frequency not to cause visible pulsing or resonance in the system. These requirements can conflict. The goal is to provide just enough Dither to prevent Stiction and Hysteresis without inducing other problems.

To generate Dither, the DVC superimposes its high frequency PWM over a low frequency carrier signal. The low frequency carrier signal is set at 50% duty cycle at the selected Dither frequency.

Low frequency PWM used as Dither
Low frequency PWM, typically less than 300 Hz, generates dither as a byproduct of the PWM process. The amount of the dither effect changes as the average coil current changes. Here, the dither effect is a maximum at 50% duty cycle but decreases to zero at 0 and 100 % duty cycles. For Low frequency PWM to be most effective, it must be applied within the frequency range specified in the valves datasheet.

The dither current amplitude as a given average current is a function of coil inductance, PWM frequency and the applied voltage. The illustration to the right shows low frequency PWM at about 65 Hz (the bottom waveform) and a voltage representation of current (the top waveform). Notice how when the PWM voltage is switched off, the flyback diode is turned on and the voltage spikes then bleeds off as the coils energy is discharged through the diode. This happens because of the inductors (coil) natural resistance to changes in current. Then the diode shuts off as the coil is charged up again.
**High frequency PWM**

When the PWM frequency is high enough, typically above 5 KHz, the coil current will not have time to change significantly. No dither effect is produced by high frequency PWM. In this illustration (right) the PWM frequency is 19.5 KHz. Notice how much less peak to peak voltage is seen on the current waveform (top waveform) compared to the low frequency illustration.

As stated before, the DVC superimposes its high frequency PWM over a low frequency carrier signal. This low frequency carrier signal is set at 50% duty cycle and at the selected Dither frequency. The use of high frequency PWM with a dither generator maintains the full dither effect even approaching 0% and 100% duty cycle. Here, the dither waveform is produced deliberately and added to the output waveform. The finial illustration shows 120 Hz Dither added to the DVCs output at 50% amplitude.

This dither current waveform is regulated to maintain the desired coil current regardless of the inductance of the coil or applied voltage. Here the dither amplitude will decrease as the duty cycle nears 0% or 100%, but is constant over the rest of the current range. Typically, valves are run at 0% or 100% duty cycle so this effect is inconsequential. The dither generator allows the dither amplitude and frequency to be adjusted independently for maximum positive effect with a minimum of induced problems.
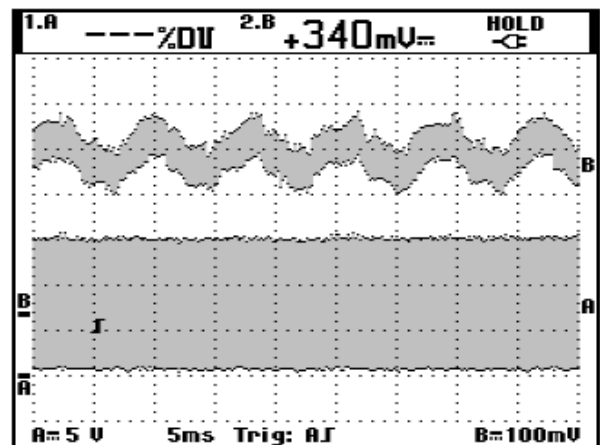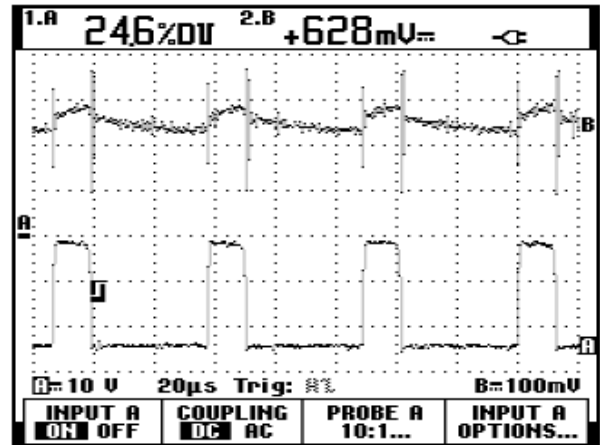
15 Appendix F   Flowchart (Sequence of Operations) example

This example will demonstrate how to create a flowchart and Sequence of Operations. This example may or may not work; it is for educational purposes only.

Programming Team:

            Paul – Programmer, Electronics Engineer
            Mark – Program Manager
            Amanda – Application Engineer
            Todd – Hydraulics Engineer

Date: 6/4/08

Program Description:

Using the HCT DVC710, control steering and propulsion functions of a new skid steer model
Using a J1939 equipped diesel engine.

Functional Description:

The operator has control over one joystick. Critical functions are the engine rpm > 1000rpm, and then the system looks for input from the joystick as indicated below. Include Counter Rotate mode activated by momentary switch located on the joystick. Use separate function curves for each valve direction and for Counter Rotate

Joystick forward – skid steer moves forward motion (Proportional)
Joystick back – skid steer moves in reverse motion (Proportional)
Joystick right – skid steer turns right (Proportional)
Joystick left – skid steer turns left (Proportional)
Joystick center – skid steer stops in present position

I/O needed

Inputs   Joystick Y axis (Throttle)   universal input 1, 0 – 5 Volts, Center = 2.5V
        Joystick X axis (Steering)   universal input 2, 0 – 5 Volts, Center = 2.5V

        Momentary Switch (Counter Rotate)   digital input, active high no toggle

        Engine rpm   J1939 signal

Outputs
        Left wheels pump FWD/REV control pwm   PWM_1, Min cur 0.2a, Max cur 0.9a
        Right wheels pump FWD/REV control pwm   PWM_2, Min cur 0.2a, Max cur 0.9a

---

Step #1

      Determine if Counter rotate active.

Step #1a   Counter Rotate Active

      Pumps in opposite directions, speed controlled by Joystick X axis position

---

Step #1b   Counter Rotate Not Active

           Pumps direction according to joystick Y axis direction Speed controlled by
           Joystick Y axis, Steering controlled by Joystick X axis
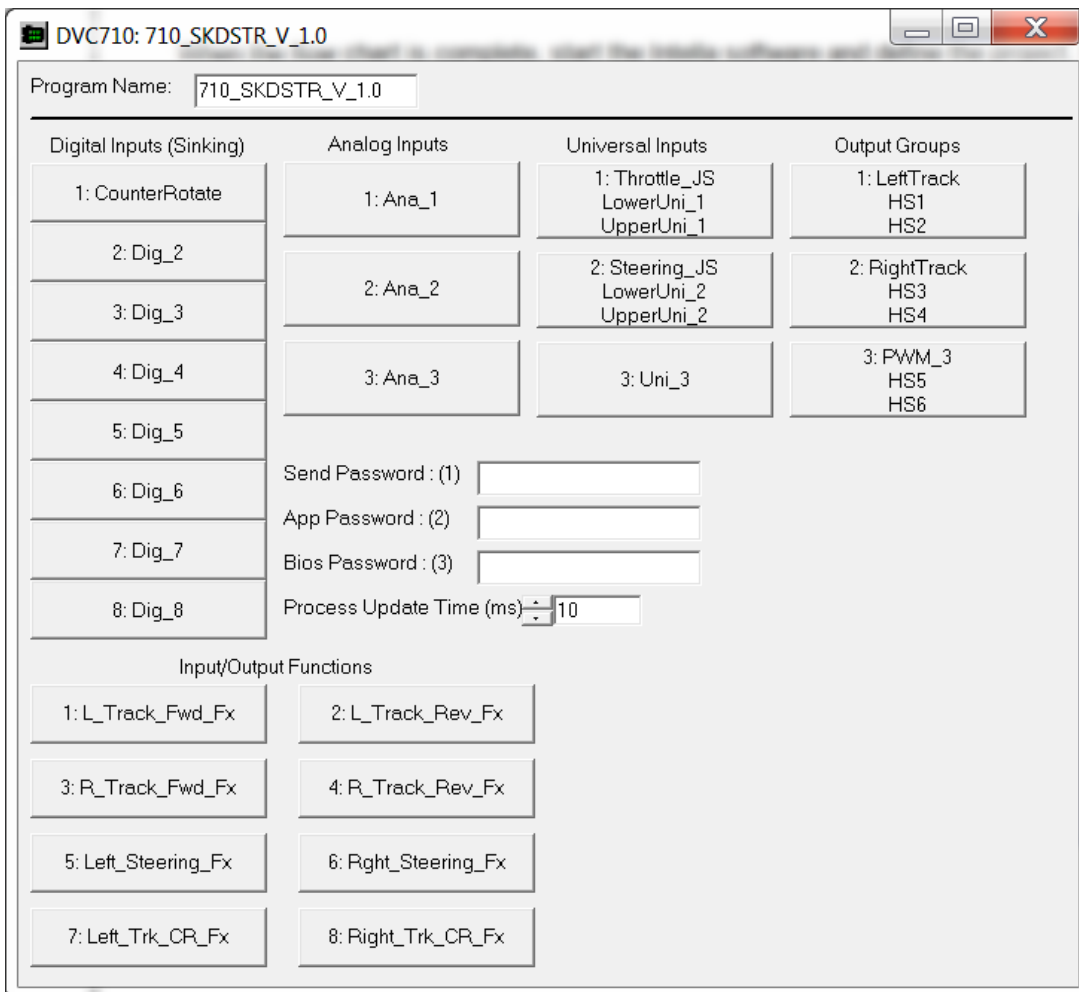
Step #2   If engine rpm is above 1000rpm
               Enable Outputs
               Else
               Disable Outputs

When the flow chart is complete, start the Intella software and define the project, then declare all I/O.

Then create the logic and write the code as needed.
```
' ***** Tramming Fx *****
L_Track_Fwd_Fx.in = Throttle_JS
L_Track_Rev_Fx.in = Throttle_JS
R_Track_Fwd_Fx.in = Throttle_JS
R_Track_Rev_Fx.in = Throttle_JS
Left_Steering_Fx.in = Steering_JS
Rght_Steering_Fx.in = Steering_JS
Left_Trk_CR_Fx.in = Steering_JS
Right_Trk_CR_Fx.in = Steering_JS


' ***** Tramming Control *****
if (CounterRotate) then
   if (Steering_JS.dir) then
      LeftTrack.dir = 1
      RightTrack.dir = 0
   else
      LeftTrack.dir = 0
      RightTrack.dir = 1
   end if
   LeftTrack = Left_Trk_CR_Fx.out
   RightTrack = Right_Trk_CR_Fx.out
else
   LeftTrack.dir = Throttle_JS.dir
   RightTrack.dir = Throttle_JS.dir
   if (Throttle_JS.dir) then          'FWD
      if (Steering_JS.dir) then
         LeftTrack = L_Track_Fwd_Fx.out
         if (R_Track_Fwd_Fx.out >= Rght_Steering_Fx.out) then
            RightTrack = R_Track_Fwd_Fx.out - Rght_Steering_Fx.out
         else
            RightTrack = 0
         end if
      else
         if (L_Track_Fwd_Fx.out >= Left_Steering_Fx.out) then
            LeftTrack = L_Track_Fwd_Fx.out - Left_Steering_Fx.out
         else
            LeftTrack = 0
         end if
         RightTrack = R_Track_Fwd_Fx.out
      end if
   else
      if (Steering_JS.dir) then
         LeftTrack = L_Track_Rev_Fx.out
         if (R_Track_Fwd_Fx.out >= Rght_Steering_Fx.out) then
            RightTrack = R_Track_Fwd_Fx.out - Rght_Steering_Fx.out
         else
            RightTrack = 0
         end if
      else
         if (L_Track_Fwd_Fx.out >= Left_Steering_Fx.out) then
            LeftTrack = L_Track_Fwd_Fx.out - Left_Steering_Fx.out
         else
```

```
                LeftTrack = 0
            end if
            RightTrack = R_Track_Rev_Fx.out
        end if
    end if
end if

if ((((EEC1.Engine_Speed_H * 256) + EEC1.Engine_Speed_L) / 8)  > 1000) then  'Allow Tramming if RPM > 1000
    LeftTrack. Enable = true
    RightTrack.enable = true
else
    LeftTrack..enable = false
    RightTrack.enable = false
    LeftTrack = 0
    RightTrack = 0
end if
```

16 Appendix G   HCT Terminology and Definitions

Always bubble – Logic bubble executed once each Logic Cycle before the code in the active bubble in a Logic Sequence.

Analog Input – An input that measures voltage levels and reports the state of the input as a 10 bit number, 0 – 1023 (or as a percentage of the 10 bit number, 0 – 100%).

Bang Bang Valve – Discrete function on/off type valve.

Compile – Convert the application code to machine language and create the output files, .PGM and .MEM that will be loaded into the DVC Controller. (Any syntax errors in the program will be flagged during the compiling process)

Counter Mode Input – Input counts pulses, the name.Counter variable may be reset through application code.

Current Regulation – Regulating output current as demanded by the application.

Digital Inputs – An input that measures voltage and reports that state as either a logical 0 (off) or a logical 1 (on).

Dual Coil High-Side Output – Output Group Configuration that uses both High Side Outputs and the PWM Output in conjunction in order to command a dual coil valve.

DVC – Digital Valve Controller.

EEMemory – Electronically Erasable Memory (EE Memory) is memory that is maintained (nonvolatile) when there is no power to the DVC710 / DVC707.

Enable Current Ramps – An output process configuration where current will be ramped up or down based on the ramp time set-points.

Enable Process PI – An output process configuration where the application program attempts to follow a desired set-point value by monitoring a feedback input and adjusting the current to the output accordingly using Proportional / Integral control.

Entry Code – The code in a Logic Bubble that is run once each time the bubble becomes active.

High-Side Only Output – Output Group Configuration that enabled use of only the two High Side discrete outputs associated with an Output Group.

Input/output Functions – Patented feature of HCT, allows the programmer to simplify math functions required to shape the relationship of an input to an output by simply configuring a graph.

Logic Bubble – An object in a Logic Sequence containing application code.

Logic Sequence – An Object in the application containing Logic Bubbles and Transitions.

Output Group – A subset of the outputs on a DVC consisting of one proportional Low Side PWM output and two High Side Discreet outputs. May be configured to drive one to three valves or devices.

Process Update Time (ms) – Sets period of a Logic Cycle. The process time from 1 to 20ms, the default is 10ms. (settings of less than 2mS is not recommended)

Program Loader Monitor(PLM) – PC Computer application used to monitor the DVC systems Inputs and Outputs and two download programs into DVC modules.

Programming Tool – PC Computer application used to create and modify DVC application code.

PWM Duty Cycle Control – An output process configuration where the application directly controls the percentage of duty cycle of the PWM output without respect to coil current.

PWM Frequency – An output process configuration where the programmer may control the frequency of the PWM output signal.

Repeat Code – The Code in a Logic Bubble that is run each time the bubble is visited by the BIOS as long as the Logic Bubble is active.

RPM Pulse Input – An Input that monitors voltage pulses and calculates a RPM variable with respect to the pulses per rev set-point.

Single Coil Low-Side Output – Output Group Configuration that uses the PWM Output in conjunction with system power to command a single coil valve. The remaining two High Side Outputs are available for use with discrete (on/off) valves or devices.

Single Coil High-Side Output – Output Group Configuration that uses one High Side Output and the PWM Output in conjunction in order to command a single coil valve. The remaining High Side Output is available for use with a discrete (on/off) valve or device.

Transition – An object in a Logic Sequence linking Logic Bubbles with conditional statements.

Universal Inputs – An analog Input that may be configured for different functions including, Analog Input, RPM Pulse Input, Counter Mode, Quadrature Mode or as a Digital Input.

17 Appendix H   Interfacing with PV780, PV450 and PV380 Displays

The PV780, PV450 and PV380 graphical displays communicate over the J1939 bus and are easily used in a project. The programmer would treat the display as a node on the Bus by configuring and programming messages to interact with the display in the DVC application. Then program the display to receive these messages, act on them as desired and return information to the DVC modules as needed. The displays are programmed using the Power Vision development kit.   HCT offers both sales and training of the Power Vision tools as well as assistance in project programming of the displays.

High Country Tek Inc.
208 Gold Flat Court
Nevada City, CA, 95959.

Customer Service
Phone: 1 530 265 3236

www.highcountrytek.com

High Country Tek Inc. was started in 1980 as a high quality contract electronics manufacturing company and we have grown over the years to expand not only this aspect of our business, but also moving into the arena of providing our many successful customers with innovative and elegant electro-hydraulic control solutions.

We are able to offer our own cost effective range of dedicated function, specialty controllers for systems such as Hydraulic fan drives and mobile generator control, as well as a comprehensive range of industry leading ruggedized user configurable, digital modules that can be combined and programmed to realize even the most difficult and expansive systems.

We initiate and manage both the hardware and software design with our in-house team of experienced engineering staff from the head office in Nevada City, CA. and have several industry experienced Field application engineers placed around the country able to support and work with you on projects from concept to fulfillment.

High Country Tek Inc. is known for product quality, pioneering technology and second to none customer service. Please visit our website (www.highcountrytek.com) to see our full product capabilities or contact us with your immediate or future control needs, we would be glad to work with you.

Thank you for using High Country Tek Inc. products.