



DyneSystems, Inc.

**Midwest & Dynamatic
Dynamometers**

**Cell Assistant
Dynamometer Test Automation Software**

USER MANUAL

No part of this manual may be reproduced or transmitted in any form or by any means, including photocopying, recording, or using information storage and retrieval systems, for any purpose other than the purchaser's own use, without the express written permission of Dyne Systems, Inc.

©2015 by Dyne Systems, Inc. All Rights Reserved.

Any other product names and services identified in this manual are trademarks or registered trademarks of their respective owners. No such uses, or the use of any trade name, is intended to convey endorsement or other affiliation with Dyne Systems, Inc.

Contact Information

Shipping Address:	W209 N17391 Industrial Drive Jackson, WI 53037
Mailing Address:	W209 N17391 Industrial Drive Jackson, WI 53037
Toll Free Phone:	(800) 657-0726
Fax:	(262) 677-9308
Web:	www.dynesystems.com
E-Mail:	sales@dynesystems.com

Document Revision History

Version	Date	Comment
01	03-MAR-2012	Initial Release.
02	01-NOV-2015	First Major Update.

Product Software Revisions

This document is up to date with respect to the following versions of product software.

Cell Assistant
Version 2.00 (Build 0019)

Table of Contents

Chapter 1 Introduction

Chapter 2 Installation

Operating System Requirements	1
Hardware Requirements	1
Installation of Cell Assistant	2
Licensing	3
Evaluating Cell Assistant	3

Chapter 3 Getting Started

Disable the Watchdog Feature	1
Creating a TestSystem.	2
Hardware Devices Tab	3
Channels Tab	3
TestPlan Tab	4
Limits Tab	4
Data Files Tab	4
Calibration	4
Adding Virtual Instrument Panels (VIP's)	5

Chapter 4 Hardware Devices

Adding a Device.	2
Deleting a Device.	2
Configuring a Device.	2
Serial Port Configuration	3
Network Configuration	4
Device Configuration	5
Device Diagnostics	6
Disabling a Device.	7

Chapter 5 Channels

Virtual vs. Physical Channels	2
Sorting and Filtering the List.	2
Exporting & Importing Channels.	3
Channel Types	3
Creating Virtual Channels	6
System Function Channels	8
TestPlan Calculation Channels	9
Average Channels	9
Modify / Configure Channels.	10
Delete / Rename Channels	11
Accessing Channels in a TestPlan.	12

Chapter 6 TestPlans

Top-Level TestPlans	2
Other TestPlans	2
GLOBAL Subroutines	2
CALCULATED Functions	3
LIMIT Handlers	3
Running a TestPlan	4
TestPlan Editor/Debugger	4
Adding Dialogs	5
Flight Recorder Files	5
Programming Topics	6
Language Extensions	6
Object References	6

Chapter 7 Limits

Adding a Limit	2
Configuring a Limit	2
Trigger Type	2
Trigger Range.	2
Limit Priority.	3
Limit Procedure	3
Exporting & Importing Limits	4
Limit Handler Procedure	4
Controlling Limits in a TestPlan.	5
Limit Sets.	7
“ThisLimit” Object.	9

Chapter 8 Data Files

Creating a Data File Format	2
Organizing Channels	2
Exporting & Importing Data File Formats	2
Data File Contents	3
Using Data File Formats in a TestPlan	4
Open	4
Close	4
IsOpen	5
StartLogging	5
StopLogging	6
LogRate	6
LogPeriod	6
IsLogging	7
SnapShot	7
ForceEntry	7
WriteChannelNames	8
WriteChannelUnits	8
WriteChannelHeader	8

Chapter 9 Virtual Instrument Panels

Microsoft Visual Studio Express 2012 (or similar)	2
Creating a VIP	2
VIPCONTROLS.ocx	4
Network VIP's	5
Using VIP's in a TestPlan	6
VIP Driver	7
VIP Relocator	7

Chapter 10 DSBasic Dialog Editor

Creating a dialog	1
Launching a dialog	2
Interacting with a dialog	2

Chapter 11

Calibration

Calibration Equation

Calibration Types.

Calibration Procedure

Calibration Database.

Exporting & Importing Calibration Database

1

2

2

4

4

Chapter 12

Preferences

Appearance

Features.

Watchdog.

1

1

1

Chapter 13

Miscellaneous

Data Folder Structure

Backing Up Data

Passwords

Status Bar

1

3

4

5

Chapter 14

System Channels

CHAPTER 1

Introduction

Cell Assistant is a very powerful data acquisition and control system providing all standard components of a full-featured data acquisition subsystem - synchronized data acquisition from multiple devices, limit detection, recorded data files, channel calibration, a flight recorder, etc. Device drivers are provided for connection to a variety of devices such as closed-loop dynamometer controllers, programmable logic controllers (PLC's), emissions analyzers, fuel meters, etc.

The control subsystem consists of running TestPlans written in the DSBASIC programming language. DSBASIC is a powerful scripting language similar to the Microsoft Visual Basic programming language. But it also incorporates many language extensions specifically designed for data acquisition and control. Using these language extensions, a running DSBASIC TestPlan can access channel data as well as control the operation of all connected devices. DSBASIC is an excellent programming language for writing TestPlans ranging from extremely small and simple tests (e.g. basic dynamometer speed and torque control, engine endurance tests, etc.) to extremely large and complex tests (e.g. EPA drive cycles, 40CFR Part 1065 Emission Testing, SAE Roadload and Coastdown tests, etc.)

Cell Assistant provides two methods for creating Graphical User Interfaces. First, the DSBASIC development environment includes a dialog editor. The dialog editor is an ideal tool to build simple dialogs for data entry at the start of a test. Secondly, Cell Assistant also includes an ActiveX Control that can be used to create high quality virtual instrument panels (VIP's). This control provides a rich set of user-interface controls such as analog meters, digital meters, strip charts, buttons, and pilot lights. Using Microsoft Visual Basic as a VIP builder, these controls can be used to build high quality user-interfaces for Cell Assistant. Absolutely no programming is required. Each control is designed to work interactively with Cell Assistant allowing the user to modify the appearance and operation of control merely by changing the control properties.

CHAPTER 2

Installation

1.0 Operating System Requirements

Cell Assistant runs under Microsoft Windows XP, Vista, and Windows 7.

2.0 Hardware Requirements

Cell Assistant will run on any personal computer running any of the previously listed operating systems. Additional hardware may also be required as described below.

A high-performance multi-port RS-232 serial port board typically needs to be installed as many devices still communicate via an RS-232 port. The high-performance serial port card provides two benefits. First of all, it provides 4 or 8 additional serial ports; whereas, a typical personal computer provides at most one (or none). Secondly, the high-performance serial ports utilize extremely large hardware buffers (4KBytes) guaranteeing no lost or dropped data bytes during data acquisition.

Many devices now provide a network interface. In this case, a 2nd Network Interface Card (NIC) must be installed. To support multiple network based devices, a network switch is also required.

If Dyne Systems is providing your computer, all required additional hardware will be pre-installed and configured; else, contact Dyne Systems for recommendations for all hardware.

3.0 Installation of Cell Assistant

A file similar to the following will be provided to you (via CD, e-mail, download, etc.).

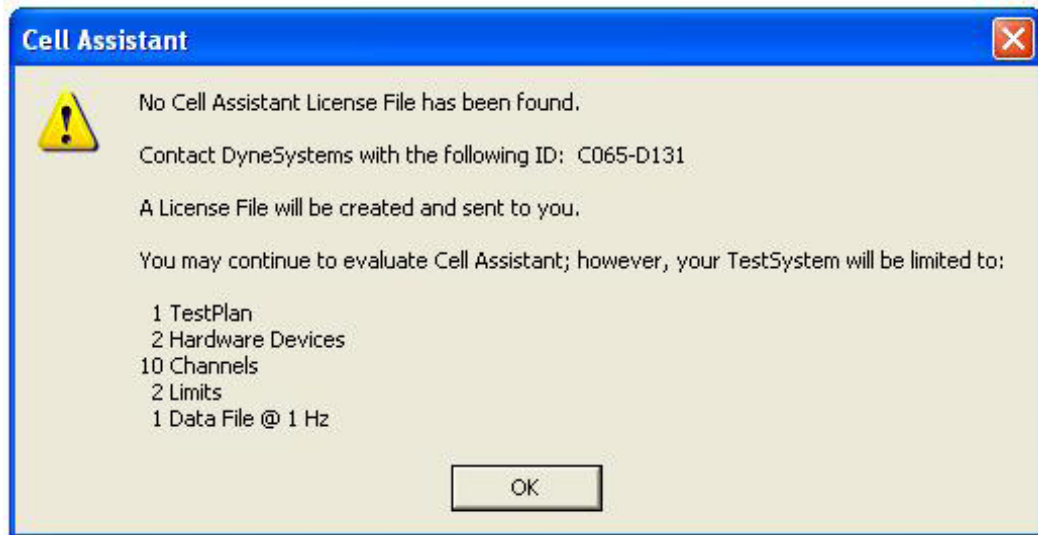
File Name = **Install-CA2012-2.00.0019.exe** (or similar)

In Windows Explorer, double-click this file to start the installer. Then follow the installation instructions as they are presented. The most important installation decision is the location of the installation data folder. For simplicity and ease of service, please keep the default installation data folder (i.e. C:\CA2012). Also, let the installer place a Cell Assistant icon on the desktop making it easier to launch this product.

Note: Cell Assistant uses older-style help files (i.e. *.hlp files). Unfortunately **WinHlp32.exe** is no longer included in Vista and Windows 7 distributions. Additionally, these OS's will interrupt the installer when they detect the installation of a *.hlp file. Fortunately, the interruption consists of a pop-up dialog directing you to a web site where **WinHlp32.exe** can be downloaded and installed. Follow the links and install **WinHlp32.exe** for the correct OS (i.e. Windows 7, 32-bit or 64-bit version).

4.0 Licensing

A full product license is required to use all of the features of Cell Assistant. A license consists of a license file that must be installed on the host computer. If no license file exists, the following dialog will appear whenever Cell Assistant is started.



As stated in the dialog, contact Dyne Systems service department and provide the listed ID (unique for each host computer). If you have purchased Cell Assistant, the ID will be used to generate a unique license file which will be e-mailed to you. The file (i.e. License.txt) must be placed in the Cell Assistant installation data folder (typically C:\CA2012).

If you have not purchased Cell Assistant, you may continue to use the product in "Evaluation" mode as described below.

5.0 Evaluating Cell Assistant

In evaluation mode, Cell Assistant operation is restricted as listed below; however, these restrictions will not prevent you from fully evaluating the power and versatility of Cell Assistant.

- 1 TestPlan
- 2 Hardware Devices
- 10 Channels
- 2 Limits
- 1 Data File Format @ 1 Hz logging rate

CHAPTER 3

Getting Started

This chapter provides an overview of Cell Assistant providing the minimum steps to quickly create a TestSystem and begin executing a test.

1.0 Disable the Watchdog Feature

Launch Cell Assistant by double-clicking the desktop icon.

When initially launched, an error dialog will appear regarding the initialization of the watchdog COM port. A watchdog device is an important safety mechanism in any test cell and is described elsewhere in this document. To initially proceed with Cell Assistant, the watchdog must be disabled as follows.

- 1 Click 'OK' to dismiss the warning dialog.
- 2 Select 'File' then 'Password...' from the main menu.
- 3 Enter the default configuration password (i.e. cfg) and click 'OK'.
- 4 Select 'File' then 'Preferences...' from the main menu.
- 5 Click the 'Watchdog' tab.
- 6 Select 'Disabled' in the 'Serial Port' drop-down list.
- 7 Click the 'Apply' button. Enter your initials and click 'OK' to log this change to the watchdog settings.
- 8 Click 'OK' to dismiss the 'Program Preferences' dialog.
- 9 Exit Cell Assistant.

2.0 Creating a TestSystem

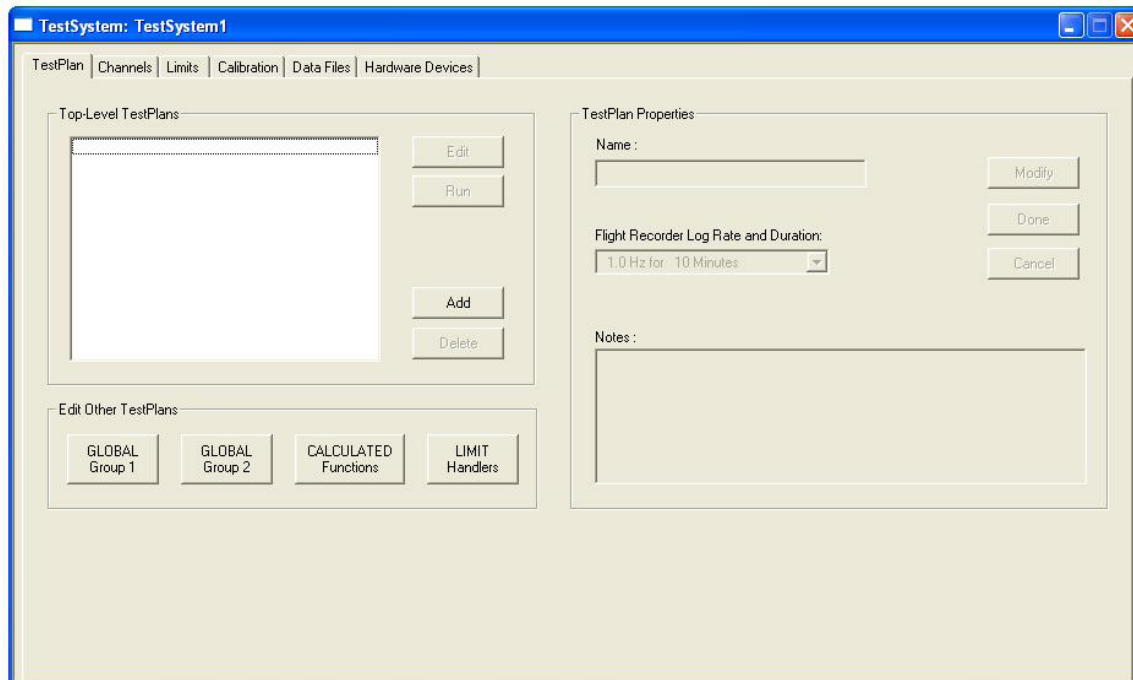
All information regarding your overall testing system are maintained in a single file. This information includes a list of installed devices, data channels, limits, etc. TestSystem files have a .ca2012 file extension. Unless the hardware configuration of a test cell changes, only one TestSystem file is typically required.

Launch Cell Assistant by double-clicking the desktop icon.

Create an empty TestSystem file as follows.

- 1 Select 'File' then 'Password...' from the main menu.
- 2 Enter the default configuration password (i.e. cfg) and click 'OK'.
- 3 Select 'File' then 'New' from the main menu. An empty TestSystem appears.
- 4 Select 'File' then 'Save As...' from the main menu.
- 5 Create a unique name for this TestSystem file and save it in the default 'TestSystems' folder.

The main TestSystem dialog (shown below) consists of six tabbed pages - TestPlan, Channels, Limits, Calibration, Data Files, and Hardware Devices. Each tabbed page manages a fundamental component of your TestSystem. To create your first TestSystem, visit the tabbed pages in the order described in this chapter. Each page is also described in greater detail in chapters that follow. The following list is merely a brief summary of the steps needed to create a TestSystem.



2.1 Hardware Devices Tab

The first step in creating a TestSystem is to install device drivers for all attached devices. All devices (serial port or network based) should be connected to the host computer and powered up. Add a driver instance for each connected device. Although rarely necessary, multiple devices of the same type (and hence multiple instances of the device driver) are allowed.

A set of device configuration dialogs are available for each device. These dialogs are used to configure the serial port and/or network communications settings as well as other device-specific configuration. Simple diagnostic dialogs are also available to test communications and overall functionality of each connected device.

Do not proceed to the next step until you have successfully established communications with each device.

2.2 Channels Tab

The second step is to create virtual channels that exchange data with the various installed devices. Some channels may be data channels (e.g. thermocouple readings, voltage readings, PLC I/O bits, etc.) while others may be device control channels (e.g. On/Off and Speed/Torque mode commands to a dynamometer controller). Other channel types such as internal timers, averages of other channels, computed channels, etc. can also be created.

If your system has been completely specified and fully designed, all channels can be created at this time. Unfortunately, this situation rarely occurs. Plus, it is also a good practice to build your design in smaller increments; therefore, initially create just a few analog input channels (e.g. temperatures, voltages, speed readings, torque readings, etc.) for each installed device.

Start data acquisition to test the channels created in the previous step as well as to further test communication with the connected devices.

Select 'Control' then 'Start Data Acquisition' from the main menu. A warning dialog may appear since the previously created channels have not yet been calibrated. Check the 'Do Not Warn Again' checkbox and click the 'Continue' button to start data acquisition. If all devices are properly connected and configured, data acquisition will start with no errors. The 'DAQ' indicator will appear in the status bar at the bottom of the Cell Assistant main application window. If any error dialogs appear, one or more devices are not properly connected and/or configured. Return to the 'Hardware Devices' tab to resolve these issues.

Use the Channel Viewer utility to view and validate all acquired data. Select 'View' then 'Channel Data' from the main menu. The Channel Viewer utility will appear. All channel data is displayed and actively updated.

Note: The Channel Viewer utility can be used to modify channel data. While this typically makes no sense for input channels (as they are continuously re-acquired and updated by the data acquisition subsystem), this is extremely helpful for debugging output channels. Later on (after various analog, digital, or device control output channels have been created), the Channel Viewer can be used to write data to these channels which ultimately send voltage commands, speed/torque setpoints, etc. to connected devices.

Close the Channel Viewer utility. Select 'Control' then 'Stop Data Acquisition' from the main menu to stop data acquisition.

2.3 TestPlan Tab

On the TestPlan page, create an empty Top-Level TestPlan by clicking the 'Add' button, entering a descriptive name for the TestPlan, and then clicking 'Done'. The DSBasic code for the empty TestPlan will appear in the TestPlan editor/debugger. The DSBasic code consists of a simple message box that (when run) pops up and waits for an operator to dismiss it. Click the 'Save' button and close the editor/debugger window.

Run this new Top-Level TestPlan by selecting it in the list and then clicking the 'Run' button. When a TestPlan is run, Cell Assistant first starts the data acquisition subsystem (as tested in the previous step). Next, the selected DSBasic TestPlan code is compiled and executed. For this empty TestPlan, click the 'OK' button on the pop-up message box to dismiss the message and ultimately end execution of the TestPlan. The data acquisition subsystem is automatically stopped as well.

2.4 Limits Tab

As your TestSystem grows in features and complexity, the Limits page is used to add limits and limit procedure handlers to various configured channels. Limits are fully described later in this document.

2.5 Data Files Tab

Most all TestSystems require the logging of data. The Data Files page is used to create Data File formats. A format consists of a set of channels to be logged as well as the logging rate and file storage format. Just a few lines of DSBasic code are required to open a file and begin logging to a file using these defined formats. Data Files are fully described later in this document.

3.0 Calibration

The calibration page is used to manage the calibration of analog input and analog output channels. Calibration is fully described later in this document. Calibration is perhaps the last Cell Assistant feature to become familiar with. An entire TestSystem can be created, debugged, and finalized without the need of precise calibration of all analog channels.

4.0 Adding Virtual Instrument Panels (VIP's)

Visually monitoring and/or interacting with a test is an important component of nearly every TestPlan. Cell Assistant provides a set of graphical controls in the form of an ActiveX control. Using any version of Microsoft Visual Basic, these controls can be added to any form to create custom user interfaces. These graphical controls are designed to work exclusively with Cell Assistant and require absolutely no Visual Basic programming experience. As long as Cell Assistant is launched and data acquisition has been started, each control has full access to all configured data channels and can easily be connected to any channel.

Ultimately, each created form (i.e. VIP) is compiled into an executable (.exe) file. These VIP executable files can be launched and closed as needed by a running Cell Assistant TestPlan using just a few lines of DSBasic code.

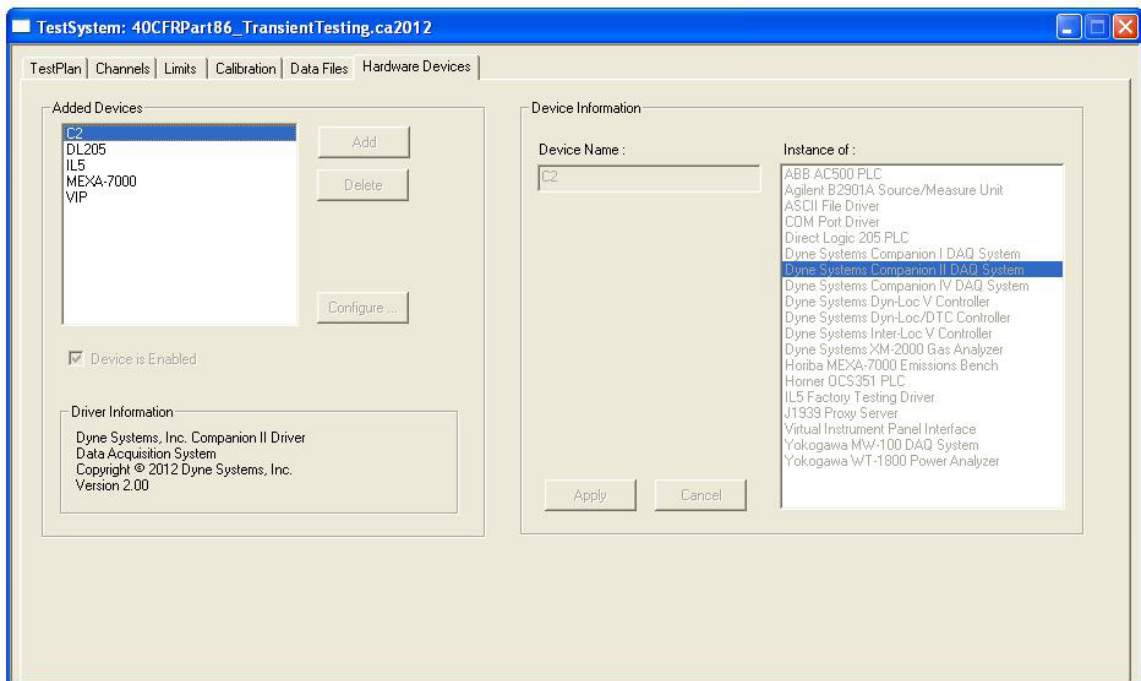
The details of creating VIP's using Microsoft Visual Basic are fully described later in this document.

CHAPTER 4

Hardware Devices

The Hardware Device page manages all devices in the TestSystem. Installing, connecting, and configuring each device is always the first step in creating a new TestSystem. For each installed device, verify that a corresponding device driver is listed on this page. If not, contact Dyne Systems immediately. The standard Cell Assistant installation contains a minimal set of drivers for the most commonly used modern devices; however, many other less popular drivers are available. Worst case, a new driver must be developed. Again, contact Dyne Systems early in your development process in order to resolve driver availability issues.

The Hardware Device page is shown below.



1.0 Adding a Device

A list of all currently 'Added Devices' is listed on the left side of the page. Click the 'Add' button to add a new instance of a device to the TestSystem. A list of available device drivers is listed on the right side of the page (i.e. the 'Instance of' list). When a new driver instance is selected, an abbreviated 'Device Name' is automatically created. These names must be unique among all installed devices. Typically, only a single instance of any device is added to a TestSystem; thus, the default name can be used. If a second instance of the same device driver is being added, modify the 'Device Name' as needed to make it unique among all installed devices. Click the 'Apply' button when done. The device configuration dialog will immediately be displayed. And because device communication has not yet been configured, an error dialog will most likely appear. Click 'OK' to dismiss this dialog.

Device configuration is discussed below.

2.0 Deleting a Device

Click the 'Delete' button to remove any previously added device from the system. Note that deleting a device should rarely be necessary and should be done with extreme caution. At most, deleting a device allows you to remove the device and then re-add the device using a new 'Device Name' (i.e. rename an installed device). This should only be done at the very beginning of TestSystem development. After channels, limits, data file formats, etc. have been created based on a device, it is NOT advisable to delete a device unless absolutely necessary.

3.0 Configuring a Device

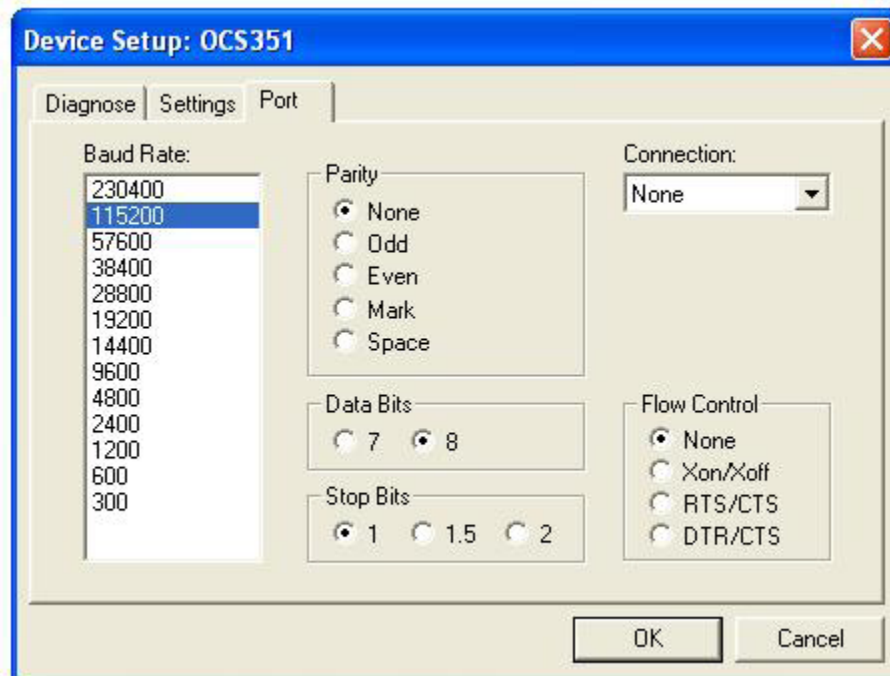
Each installed device must be properly configured by first selecting the device in the 'Added Devices' list and then clicking the 'Configure ...' button. A set of tabbed dialogs will appear providing access to communications setup, device configuration, and simple device diagnostics. The set of configuration dialogs are unique for each device. They are not documented here as operator familiarity with each device is assumed. A few sample dialogs are explained below.

3.1 Serial Port Configuration

Many devices connect to the host computer via a standard serial port (e.g. COM5, COM6, etc.). Default values for the Baud Rate, Parity, Data Bits, Stop Bits, and Flow Control are initially provided by the device driver and do not need to be changed. The only required new setting is the 'Connection' value. For each device, select the serial port (e.g. COM5, COM6, etc.) to which the device is connected.

Note: Always verify the communication settings in the device as well. They must match the settings in the device driver. Also, if possible, always configure the device to use the highest available Baud Rate.

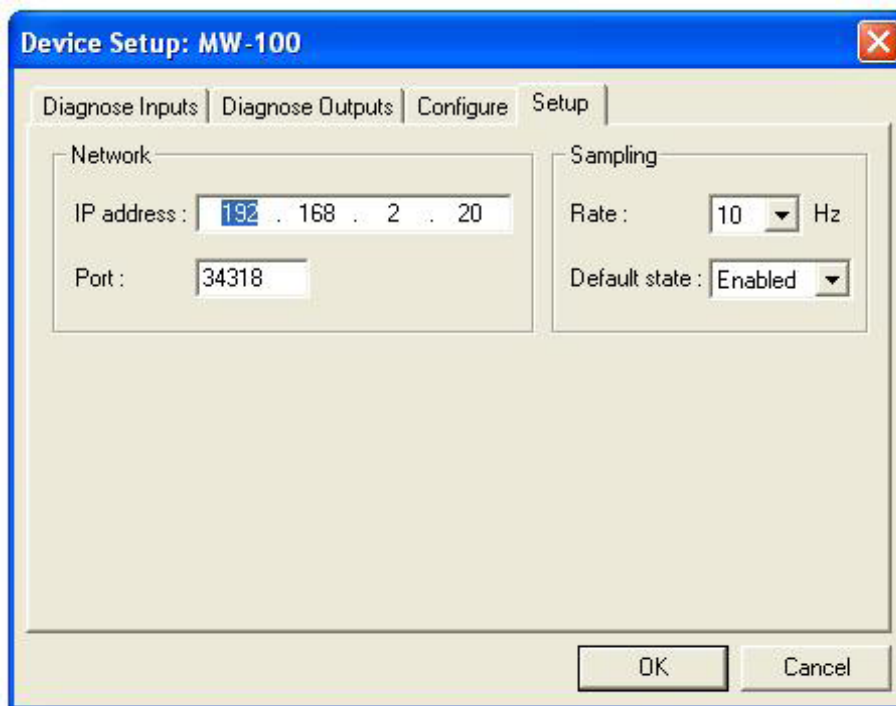
Note: Changes to communication settings are activated when the configuration dialog is closed; therefore, close then re-open the configuration dialogs whenever a communication setting is modified.



3.2 Network Configuration

Some devices connect to the host computer via a standard ethernet connection. If so, the device driver will provide a network configuration page similar to the following. Network devices are typically based on standard TCP/IP sockets; thus, the IP address and listening socket port number of the attached device must be configured in the driver. Consult the device manual for these values. They are typically configurable in the device as well; thus, you may need to contact your IT department for help in establishing a valid IP address that will not conflict with other devices on your Local Area Network. The default Port value specified by the device vendor can typically be used as is.

Note: Changes to communication settings are activated when the configuration dialog is closed; therefore, close then re-open the configuration dialogs whenever a communication setting is modified.



3.3 Device Configuration

Many devices are configurable. For example, the number, location, and type of installed modules in a data acquisition device must be configured in the device driver. Similarly, the I/O address of bits and registers in a Programmable Logic Controller (PLC) must be configured in the device driver. A sample configuration page for a PLC is shown below. The device configuration pages for all available device drivers are not documented here. Assuming operator familiarity with the device, the configuration fields presented in the respective device configuration pages should be obvious. Contact Dyne Systems with any questions.

Note: Based on the device configuration settings, the device driver creates a set of uniquely named 'physical' device channels. Ultimately, 'virtual' channels will be created from these physical channels. Thus, it is absolutely IMPERATIVE that the device configuration be correctly established at the start of your TestSystem design and not be changed afterward. This warning especially applies to PLC's. Changing the PLC configuration could cause a dangerous re-mapping of all I/O bit functionality between the PLC and your TestPlans.

Device Setup: OCS351

Diagnose Settings Port

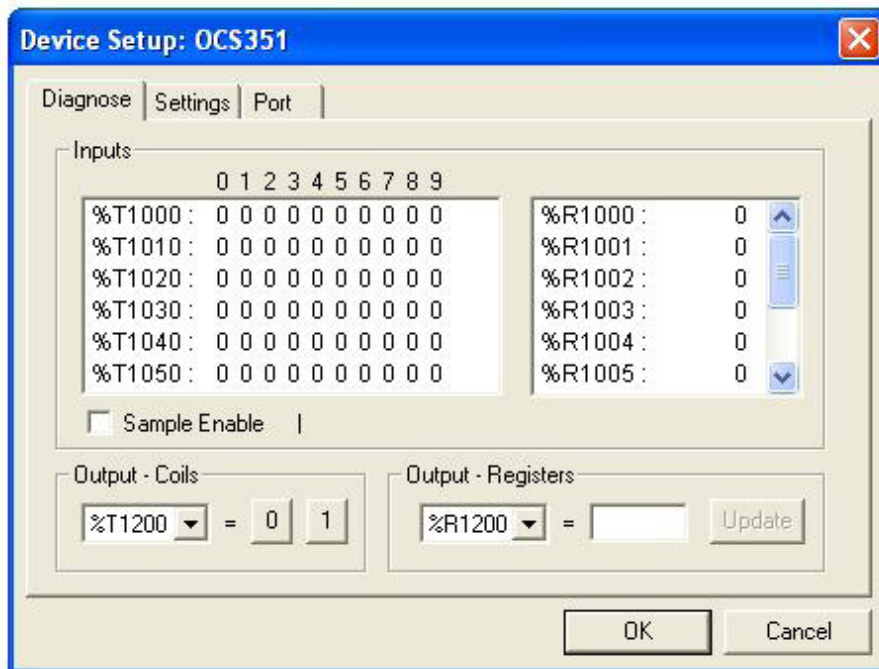
PLC Registers			
	Start	Qty.	Register Names
Digital Inputs :	1000	60	%T1000 to %T1059
Digital Outputs :	1200	60	%T1200 to %T1259
Analog Inputs :	1000	10	%R1000 to %R1009
Analog Outputs :	1200	10	%R1200 to %R1209

Slave Address : 1 Sample Rate : 10 Hz

OK Cancel

3.4 Device Diagnostics

The driver for most devices includes one or more diagnostic dialogs providing simplified access to data in the connected device. These diagnostics are independent of the Cell Assistant data acquisition subsystem and are extremely useful for basic device testing. A sample dialog for a PLC is shown below. For example, this dialog can be used to exercise all configured PLC I/O bits and subsequently verify external PLC wiring, safeties, etc. prior to actually using the channels in Cell Assistant. Every installed device should initially be tested in this manner prior to creating virtual channels and using them in Cell Assistant.



4.0 Disabling a Device

By default, when a device is added to a TestSystem, the device is fully enabled; i.e. the data acquisition subsystem will attempt to acquire data from the device whenever data acquisition is started and/or a TestPlan is run. There are several scenarios where a device must be disabled.

- A device is not yet available when TestSystem development as started.
- The device is temporarily unavailable due to repair and/or maintenance.
- The TestSystem is being created off-site where one or more devices are not available

Regardless of the reason, it occasionally becomes necessary to temporarily disable a device. When a device is disabled, the data acquisition subsystem will not communicate with the device. Similarly, any attempt by a running TestPlan to access the device (e.g. via a device control channel) will be blocked. This allows the developer to continue developing a TestSystem without the many communication error dialogs what would otherwise occur if the device were not disabled.

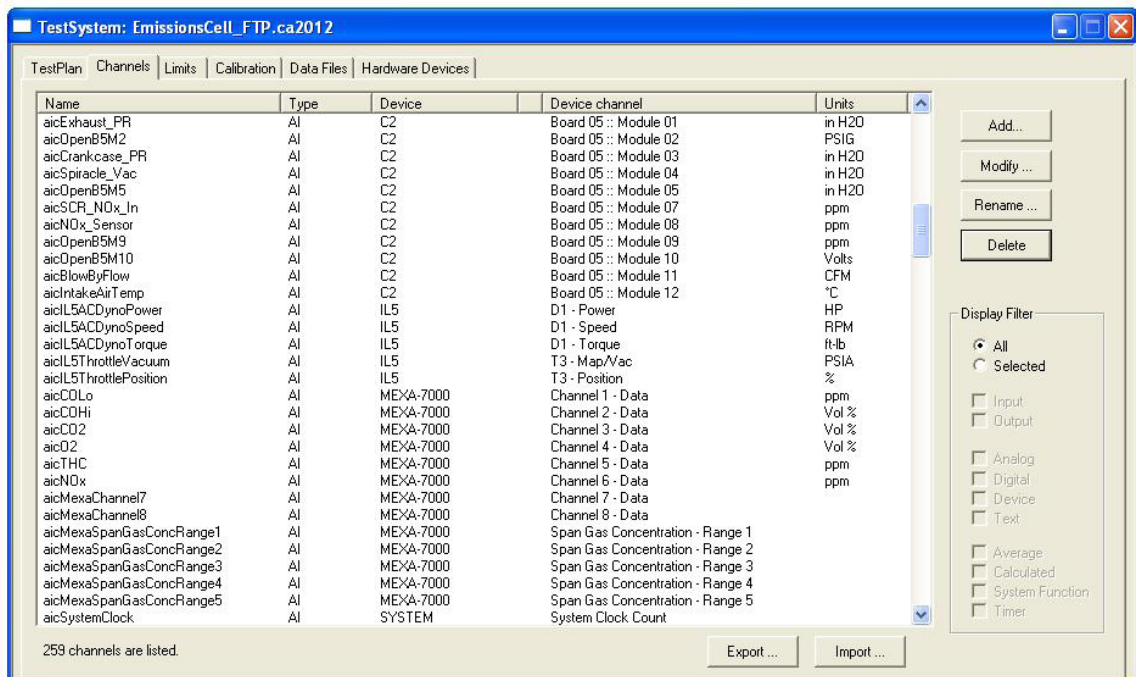
A device is disabled by selecting the device in the 'Added Devices' list and then un-checking the 'Device is Enabled' checkbox. Click the 'Save' button to make the change persist after the TestSystem is closed.

CHAPTER 5

Channels

Channels are used to exchange data and commands with all attached devices. Input channels generally are created to provide sampled readings for the data acquisition subsystem. Output channels, on the other hand, are generally used to issue commands to connected devices. All of these channels are created and configured on the Channels page.

The Channels page is shown below.



1.0 Virtual vs. Physical Channels

The Channels page manages all created 'virtual' channels'. A virtual channel is any channel created by the TestSystem developer. Each virtual channel is based on (i.e. bound to) one physical device channel. Each installed hardware device presents a set of physical channels. The name and type of each physical device channel is generally fixed; however, the TestSystem developer is free to create unique meaningful names for each virtual channel. In addition, each virtual channel can be separately scaled as needed if the scaling of the under-laying physical channel does not match the requirements of the application.

The main feature of the Channels page is the large list of virtual channels. The 'Name' column lists the name of each virtual channel as specified by the TestSystem developer. The 'Type' column shows an abbreviated type code for each channel. A list of type codes is shown below. A detailed discussion of the various channel types is presented later in this document. The 'Device' and 'Device channel' columns list the specific under-laying physical channel from the specific device that each virtual channel is based on.

Type Code	Channel Type
AI	Analog Input
AO	Analog Output
DI	Digital Input
DO	Digital Output
TI	Text Input
TO	Text Output
DSI	Device Status Input
DCO	Device Control Output
AVERAGE	Average
CALC	TestPlan Calculated
SYS FUNC	System Function
TIMER	Internal Timer

2.0 Sorting and Filtering the List

Several mechanisms are available to help manage the displayed list of virtual channels. These mechanisms are most useful for TestSystems with large channel counts. First of all, the list can be filtered using the checkboxes in the 'Display Filter' group box. The 'All' check box is the default setting where all created virtual channels are always displayed. Click the 'Selected' checkbox to enable filtering. The remaining checkboxes are then enabled (as needed) allowing the operator to filter the list based on channel type and direction.

Secondly, the displayed list of virtual channels can be sorted alphabetically (by column) by clicking the 'Name', 'Type', or 'Device' column header buttons respectively. Sorting the listed channels by

'Type' and 'Device' are the most useful - especially just prior to exporting the channel list to an external file.

3.0 Exporting & Importing Channels

The creation and configuration of virtual channels can be managed outside of Cell Assistant using popular tools such as Microsoft Excel. Using the channel Export feature, the entire displayed channel list can be exported to a comma-separated variable (*.csv) file for off-line storage, editing, and/or importing into another TestSystem. Importing standard sets of channels for various devices is an extremely quick and efficient way to set up a new TestSystem.

All currently displayed channels are exported as follows.

- 1 Select the appropriate 'Display Filter' settings (as only the currently displayed channels will be exported). Typically, check the 'All' checkbox to export all channels in the TestSystem.
- 2 Sort the displayed channel list (as the channels will be exported in the same order as they are displayed). Sorting by 'Type' is typically the most useful so click the 'Type' column header button.
- 3 Click the 'Export ...' button. An Export Channels pop-up dialog appears.
- 4 Modify the default filename as needed (e.g. change NNN to a valid number) and click the 'Save' button.

Previously exported channels can be re-imported by clicking the 'Import ...' button. A warning dialog appears explaining the dangers of importing channel data from an external file. If the imported file was previously exported from Cell Assistant (and not externally modified in any way), import errors are extremely unlikely. Dismiss the warning dialog, select the import file, and click the 'Open' button. All channels specified in the file will be merged into the existing channel list. If errors occur during the import process, it is extremely important to repair the file and re-import; else, the TestSystem may be left in an unusable state.

Note: When importing channels from a file, new channels will be added as needed and existing channels will be modified as needed. All other currently existing channels remain 'as is'. No channels will be deleted.

4.0 Channel Types

Virtual channels fall into one of two categories - (1) physical device based channels or (2) internal / computed channels. Physical device based channels are based on the physical device channels provided by the various devices via their associated device driver. Internal channels are created by Cell Assistant (e.g. timers and internal clocks). Computed channels are computed from other

channels using scripted mathematical equations and/or one of several pre-defined system functions. Computed functions and system functions are explained in greater detail later in this document.

The Cell Assistant data acquisition subsystem continuously reads all input channel values in a synchronized fashion based on the configured sampling rate of each device. It also continuously updates all internal and computed channels. Because physical device channels are often used as inputs to other computed channels, computed channels are always updated **AFTER** each data acquisition sample has completed. This maintains the synchronization of physical channels with the computed channels.

Device channels:

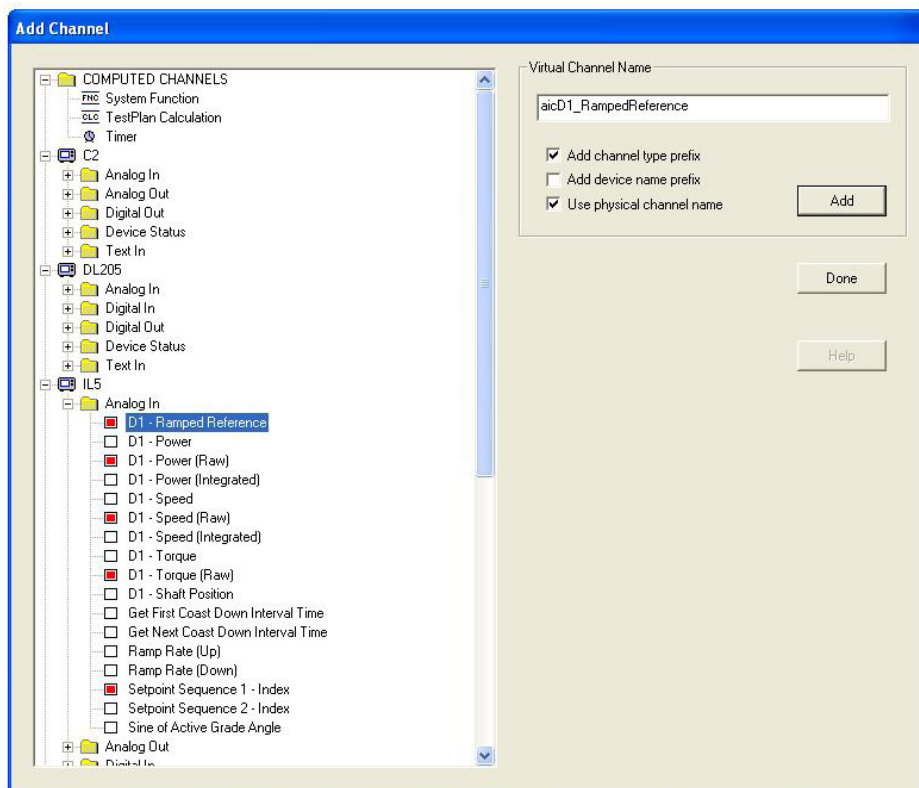
Channel Type	Representation	Typical Use
Analog Input	Double precision floating point	Typically represent readings from a data acquisition device (e.g. voltage and temperature readings, or speed and torque readings from a dynamometer controller).
Analog Output	Double precision floating point	Set output voltages on a DAQ device. Command setpoints to a dynamometer controller.
Digital Input	Boolean	Binary status values (e.g. PLC Input bits).
Digital Output	Boolean	Binary output values (e.g. PLC output relays). Simple On/Off commands sent to many devices.
Text Input	String	Retrieve string responses from simple ASCII based devices.
Text Output	String	Send commands to simple ASCII command based devices.
Device Status Input	Integer	Send multi-valued state commands to a device. (e.g. speed, torque, position mode control).
Device Control Output	Integer	Retrieving multi-valued status values from a device.

Other channels:

Channel Type	Representation	Description
Average	Double precision floating point	Computes the windowed average of another analog input channel. The width of the averaging window (in seconds) must be specified. Typically used to smooth/filter noisy analog readings.
TestPlan Calculated	Double precision floating point	These channels are implemented using the DSBASIC scripting language. The equations can be as simple or complex as needed and can use other channel values as inputs.
System Function	Double precision floating point	System functions are built-in functions, conversions, and other formulas frequently used in an engine test cell environment (e.g. HP to Watts conversion, corrected horsepower calculations, SAE and ISO corrections factors). System functions are defined in an Appendix at the end of this document.
Internal Timer	Integer	Timer channels count up (or down) in 1 second increments. They can be preset to start at any value and can be completely controlled by a running DSBASIC TestPlan.

5.0 Creating Virtual Channels

All virtual channels (except average channels) are created using the 'Add Channel' dialog which is displayed by clicking the 'Add ...' button on the Channels tabbed page. This dialog presents an Explorer-style list of all installed devices as well as all of the physical channels each device makes available for the creation of virtual channels. By expanding the various folders under each installed device, the TestSystem creator gets a full view of the channel capabilities of each device. The 'Add Channel' dialog is shown below.



Virtual channels can quickly be added as follows.

- 1 Select a physical channel from one of the listed devices. Or select one of the COMPUTED channel types from the top-most folder of special (i.e. non-device) channels.
- 2 Enter an appropriate virtual channel name (see discussion below).
- 3 Click the 'Add' button (see discussion below regarding System Function and TestPlan Calculation channels).
- 4 Repeat until all virtual channels have been created.

Virtual Channel Names:

Each virtual channel name must be unique in any TestSystem. In addition, since many virtual channels are referenced elsewhere in Cell Assistant (e.g. in data file formats, limits, and DSBasic TestPlan code), it is highly advisable to adhere to some type of channel naming convention. This allows the type, data direction, and function of any channel to be quickly determined based on its name.

The 'Add Channel' page provides a mechanism to automatically generate all or part of each virtual channel name. Checkboxes allow the TestSystem developer to enable/disable various components of virtual channel name generation. Enabling all of the mechanisms can greatly reduce the amount of time spent creating channels.

- Add channel type prefix
- Add device name prefix
- Use physical channel name

Channel type prefixes should always be used. The remaining features are optional. If enabled, the following channel name prefixes are automatically generated based on the type of the selected physical channel.

Channel Type	Channel Name Prefix
Analog Input	aic
Analog Output	aoc
Digital Input	dic
Digital Output	doc
Text Input	tic
Text Output	toc
Device Status Input	dsi
Device Control Output	dco
TestPlan Calculated	clc
System Function	sysfunc
Internal Timer	tmr

If selected, a device name prefix will be added immediately after the channel type prefix. The device name prefixes consist of the physical device name (exactly as listed in the channel list tree). This feature is rarely used. It is typically necessary when multiple instances of a device are installed in a

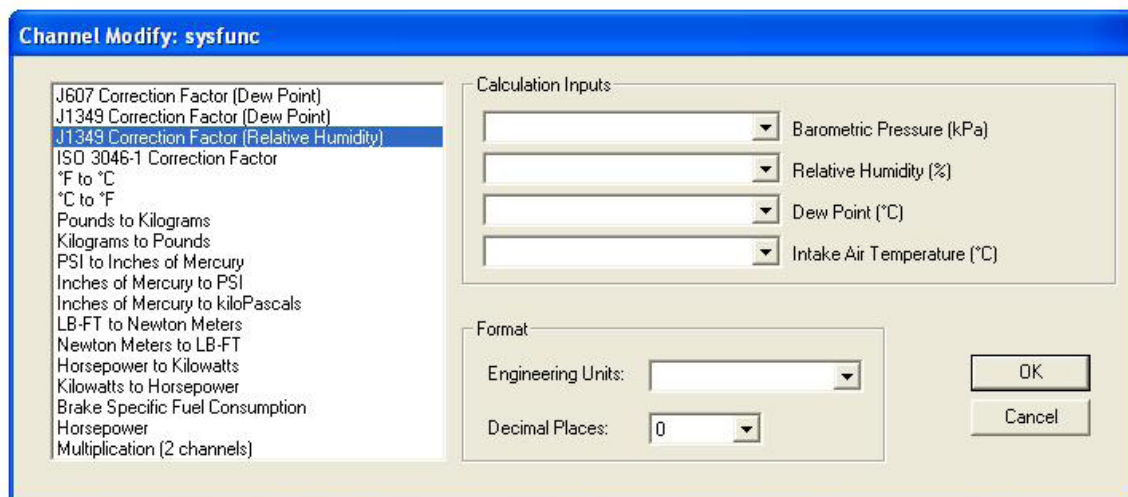
TestSystem. For example, adding the device name prefix to a channel name allows identical channels (e.g. a speed reading) from similar devices to be differentiated in the TestSystem.

Finally, the greatest time-saver during channel creation is to automatically use the physical device name (along with any other automatically generated prefixes) as the virtual channel name. This feature works well with devices that provide highly unique and well defined channel names (e.g. a dynamometer controller provides speed, torque, and power reading channels or an emissions bench provides CO, NOx, and HC reading channels). This feature does NOT work well with DAQ devices where the physical channel names are generic (e.g. slot 1 voltage, slot 2 temperature, etc.). For these devices, the TestSystem developer must generally manually enter a more meaningful name for each virtual channel.

Note: The 'Add Channel' page is designed to reduce the amount of time required for initial channel creation. Additional configuration of each created channel (if any) is done after all channels have been created by returning to the main Channels tabbed page and then using the various channel modify features.

6.0 System Function Channels

Unlike most of the other channel types, system function channels are configured immediately after they are created. When the 'Add' button is clicked, the following dialog appears.



The dialog box is titled "Channel Modify: sysfunc". It contains a list of conversion factors and functions on the left, and configuration options on the right.

Channel Modify: sysfunc

- J607 Correction Factor (Dew Point)
- J1349 Correction Factor (Dew Point)
- J1349 Correction Factor (Relative Humidity)**
- ISO 3046-1 Correction Factor
- *F to *C
- *C to *F
- Pounds to Kilograms
- Kilograms to Pounds
- PSI to Inches of Mercury
- Inches of Mercury to PSI
- Inches of Mercury to kiloPascals
- LB-FT to Newton Meters
- Newton Meters to LB-FT
- Horsepower to Kilowatts
- Kilowatts to Horsepower
- Brake Specific Fuel Consumption
- Horsepower
- Multiplication (2 channels)

Calculation Inputs

- Barometric Pressure (kPa)
- Relative Humidity (%)
- Dew Point (*C)
- Intake Air Temperature (*C)

Format

- Engineering Units: [Dropdown]
- Decimal Places: 0 [Dropdown]

OK Cancel

Use this dialog to select the conversion factor, equation, and/or function the newly created system function should implement. Based on this selection, drop-down lists for each of the appropriate inputs are displayed. Select the appropriate input channels as needed.

Note: If one or more of the required input channels have not yet been created, those inputs may remain un-configured at this time. They can be added and/or modified later on from the Channels tabbed page.

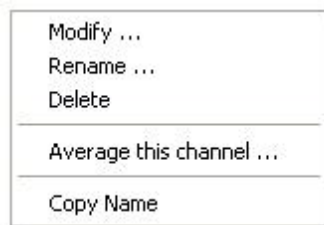
Note: All system functions are described in an Appendix at the end of this document.

7.0 TestPlan Calculation Channels

Like most of the other channel types, TestPlan Calculation channels require no immediate additional configuration when initially created; however, note that the `CALC_FUNCTIONS_TESTPLAN` is immediately opened in the background when a TestPlan Calculation channel is created. This occurs because Cell Assistant immediately generates a DSBASIC code “stub” for the newly created calculated channel. The stub initially just returns a value of 0.0. The DSBASIC implementation for the calculated channel may be entered now or at any time later on during TestSystem development. As is, the initial code stub will compile with no errors allowing TestSystem development to proceed without the final implementation of the calculated channel. Additional information regarding the implementation of TestPlan Calculation channels is presented in the TESTPLANS chapter of this document.

8.0 Average Channels

Average channels are NOT created from the ‘Add Channels’ page. Average channels are used to filter/smooth other analog input channels. Average channels are created from the Channels tabbed page. Right-click on any analog input (i.e. AI) channel and the following pop-up menu will appear.



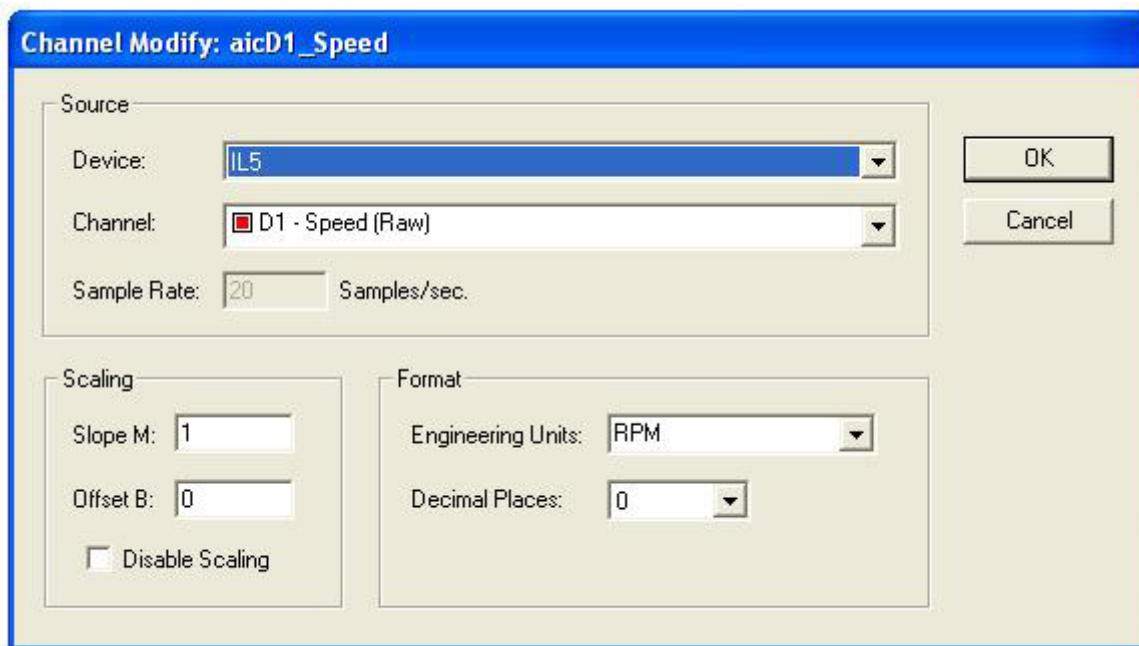
Select “Average this channel ...” and the respective ‘Channel Modify’ dialog will appear. Configure the channel as needed and click ‘OK’. A default virtual channel name is automatically created. It consists of the name of the selected analog input channel preceded by an ‘AVE_’ prefix. This default channel name can be renamed later on if needed.

9.0 Modify / Configure Channels

Additional channel configuration is available for most channel types. For example, additional scaling ($y = Mx + B$) and logic inversion can be applied. Also, incorrect device and physical channel selections can be corrected. As an example, channel configuration dialogs for analog input and digital input channels are shown below. Channel configuration/modification is available by selecting the appropriate channel on the Channels tabbed page and then clicking the 'Modify ...' button.

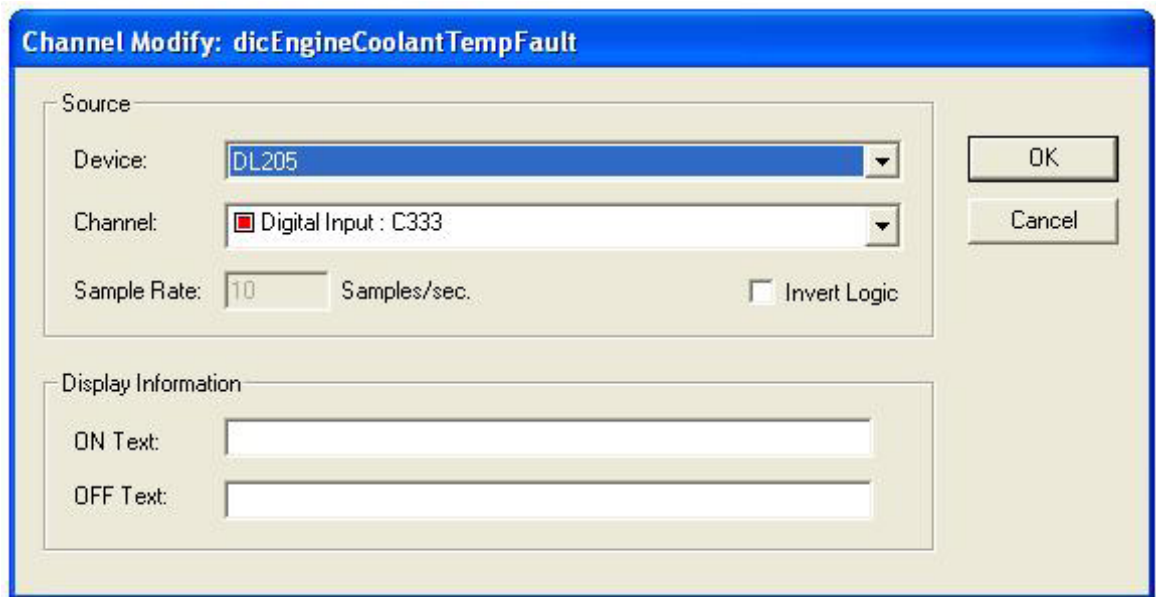
Note: Double-clicking the virtual channel name will also display the ChannelModify dialog.

Many of the configuration settings are cosmetic (e.g. Engineering Units, ON Text, OFF Text, etc.). They are typically of no use to a running DSBasic TestPlan. They do appear on other dialogs (for diagnostics) and can also be (optionally) displayed on Virtual Instrument Panels. Set these values only if necessary.



The image shows a 'Channel Modify' dialog box for the channel 'aicD1_Speed'. The dialog is divided into several sections. The 'Source' section contains a 'Device' dropdown menu set to 'IL5', a 'Channel' dropdown menu set to 'D1 - Speed (Raw)', and a 'Sample Rate' input field set to '20' with the unit 'Samples/sec.'. To the right of these fields are 'OK' and 'Cancel' buttons. Below the 'Source' section are two sub-sections: 'Scaling' and 'Format'. The 'Scaling' section has 'Slope M' set to '1', 'Offset B' set to '0', and a checkbox labeled 'Disable Scaling' which is currently unchecked. The 'Format' section has 'Engineering Units' set to 'RPM' and 'Decimal Places' set to '0'.

Channel Modify: aicD1_Speed	
Source	
Device:	IL5
Channel:	D1 - Speed (Raw)
Sample Rate:	20 Samples/sec.
Scaling	
Slope M:	1
Offset B:	0
<input type="checkbox"/> Disable Scaling	
Format	
Engineering Units:	RPM
Decimal Places:	0



10.0 Delete / Rename Channels

Virtual channels can be deleted from the TestSystem and/or renamed as needed by clicking the 'Delete' and 'Rename ...' buttons respectively. This should be done with great caution. It is generally safe to do so during the early stages of TestSystem development; however, it is a bit more troublesome later on after the channels have been bound to limits, included in data files, referenced in various TestPlans, etc.

Note: In general, Cell Assistant will attempt to repair/update the usage of a channel when it is deleted or renamed. Unfortunately, it can NOT repair/update all uses (e.g. references in DSBasic TestPlan code). Ultimately all unfixable uses will generate errors when data acquisition is started and/or a TestPlan is selected and run. At that time, the TestSystem developer must manually make all required repairs.

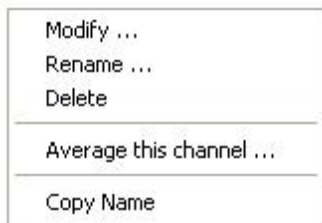
11.0 Accessing Channels in a TestPlan

The value of any channel may be accessed at any time by a running DSBasic TestPlan. As explained in other chapters, each virtual channel becomes an instantiated object to a running DSBasic TestPlan. For example, the value of a channel can be loaded into a variable using the following line of DSBasic code.

```
Variable# = aicIL5ACDynaSpeed
```

Where “aicIL5ACDynaSpeed” is a virtual channel based on a speed channel from a dynamometer controller. The value of the channel can now be used in other calculation, in decision elements, etc. Virtual channels are the simplest language extension to DSBasic; additional language extensions are explained in the TESTPLANS chapter.

Note: To properly reference a virtual channel in a TestPlan, it is extremely important to spell the virtual channel name correctly. If not, the DSBasic compiler will interpret the misspelled channel name as another variable resulting in an improperly executing TestPlan sequence. To prevent this common error, copy/paste functionality has been added to the channel list. To copy a virtual channel name into the global copy buffer, select the channel name using the right-mouse button and then select ‘Copy Name’ from the pop-up menu (shown below). Alternatively, press CTRL/C on the keyboard.

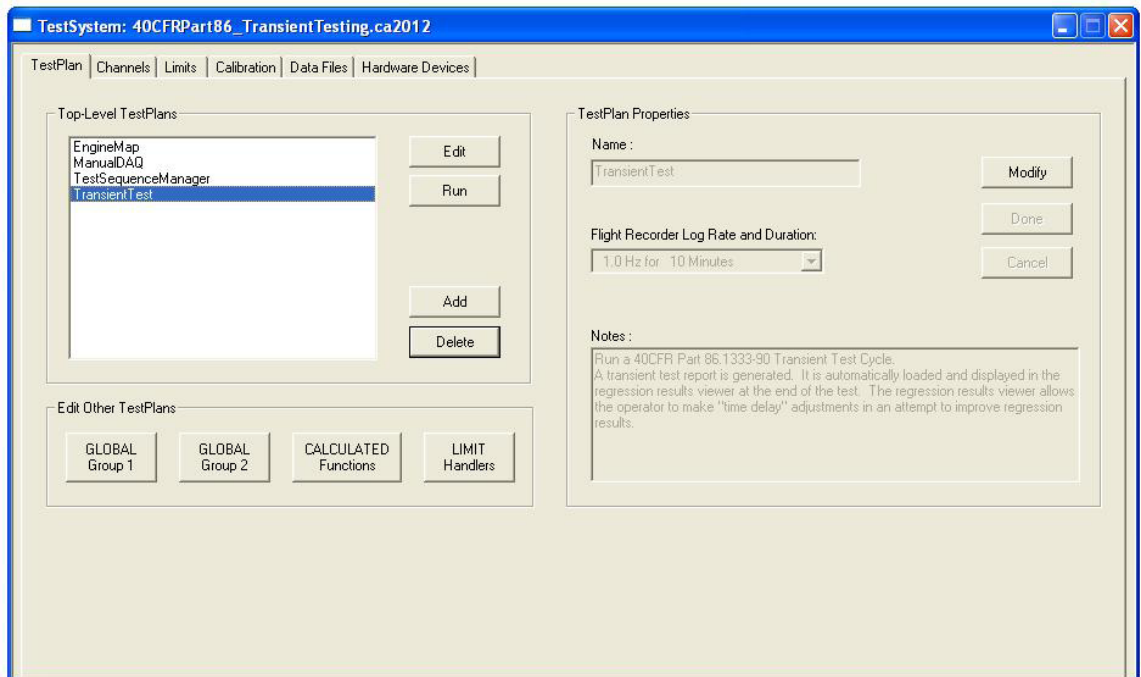


Paste the channel name into any DSBasic TestPlan in the usual manner (i.e. CTRL/V). The data value of the (correctly spelled) virtual channel is now accessible in the TestPlan.

CHAPTER 6

TestPlans

While the other pages (e.g. Channels, Calibration, Data Files, etc.) manage the “data acquisition” functionality of a TestSystem, the TestPlan page manages the “control” functionality. TestPlans control the entire flow of any test. TestPlans are written in the DyneSystems Basic (DSBasic) programming language. DSBasic is a powerful scripting language similar to the Microsoft Visual Basic programming language. But it also incorporates many language extensions specifically designed for data acquisition and control. Using these language extensions, a running DSBasic TestPlan can access channel data as well as control the operation of all connected devices.



1.0 Top-Level TestPlans

Any TestSystem may contain one or more TestPlans. TestPlans can be large (e.g. EPA emissions tests) or small (e.g. a simple report generating utility). Each TestPlan consists of one Top-Level TestPlan. A new Top-Level TestPlan can quickly be created by clicking the 'Add' button, entering a unique name for the TestPlan, and then clicking the 'Done' button. The following code stub is automatically generated. When compiled and run, it does nothing but pop up a message box and wait for the user to click 'OK' to dismiss the message box and end the test.

```

-----
` Dyne Systems, Inc. Cell Assistant TestPlan
`
` Created: 12/12/2015
-----

Rem {{DECLARATIONS}}

-----
`
Sub Main()

    Insert code here and remove next line.
    MsgBox "Test Over", 0, "SampleTestPlan"

End Sub

```

Every Top-Level TestPlan must contain exactly one "Main" subroutine. TestPlan execution begins at this point. Clearly, more subroutines must be created and saved into this TestPlan as your overall test is developed. The DSBASIC programming language is NOT documented here. The language reference is available on-line in Cell Assistant. Also, numerous textbooks are commercially available that describe the (similar) Visual Basic programming language. These can be a tremendous help to a novice programmer. This manual does contain descriptions of the various language extensions that have been added to DSBASIC. These extensions allow channel data, limits, data files, etc. to be accessed and controlled by a running DSBASIC TestPlan.

2.0 Other TestPlans

Whenever a Top-Level TestPlan is selected and run, four other TestPlan components are additionally compiled and executed. These other TestPlan components are contained in separate TestPlan files in order to better manage and organize all TestPlan code. Click the appropriate button in the 'Edit Other TestPlans' group to access these components.

2.1 GLOBAL Subroutines

To facilitate better TestPlan organization, two global TestPlans are available. All DSBASIC code in GLOBAL_TESTPLAN_1 and GLOBAL_TESTPLAN_2 are automatically compiled and executed along with every Top-Level TestPlan. These TestPlans can be used as a repository for DSBASIC

subroutines that may be used by more than one Top-Level TestPlan. They can also be used for any general subroutine in order to reduce the size of a Top-Level TestPlan that is getting large and perhaps difficult to manage.

2.2 CALCULATED Functions

The CALC_FUNCTIONS_TESTPLAN contains the DSBASIC code implementation of each TestPlan calculated function channel. See the Channels section of this manual for more information on creating TestPlan calculated functions. A code “stub” for each channel is created automatically when the channel is created. The DSBASIC code for these functions should be very short and concise (i.e. make the required channel calculation and exit the function).

Note: The CALC_FUNCTIONS_TESTPLAN should NOT contain any other DSBASIC code other than the TestPlan calculated functions.

2.3 LIMIT Handlers

The LIMITS_TESTPLAN contains all limit handling functions for each defined limit. See the Limits section of this manual for more information regarding the structure of limit handling functions. A simple limit handler code “stub” is automatically created when a limit is created. The DSBASIC code in a limit handler can be as simple or as detailed as needed in order to handler the limit trip condition.

Note: The LIMITS_TESTPLAN should NOT contain any other DSBASIC code other than limit handler code.

3.0 Running a TestPlan

This section describes (at a basic level) the sequence of internal events that occur whenever a Top-Level TestPlan is selected and run. Understanding these steps can often be useful when debugging a TestPlan.

Step	Description
1	All TestPlan components – the selected Top-Level TestPlan as well as the GLOBAL, CALCULATED Functions, and LIMIT Handler TestPlans – are compiled. These components must compile with no errors. NOTE: They are not run until a later step.
2	All language extensions are instantiated and registered. This includes an object for each virtual channel, each defined limit, each Data File Format, etc. Once registered, these objects can be referenced by all DSBasic TestPlan code.
3	The overall TestSystem is validated. For example, a DSBasic function must exist for each configured TestPlan calculated channel. Similarly, a DSBasic limit handler procedure must exist for each configured limit.
4	Data Acquisition is started. The 20 Hz Master Timing loop and device sampling loops are started and synchronized.
5	2 Second Delay. This delay allows the data acquisition system to begin retrieving valid data samples prior to beginning execution of the TestPlan code, monitoring limits, recording data files, etc.
6	The TestPlan Flight Recorder is started. The Flight Recorder is explained below.
7	The “Main” routine in the selected Top-Level TestPlan is started. Your TestPlan is now officially running.
8	The High and Low priority limit threads are started.

Note: The data acquisition system can be started by itself (i.e. no TestPlan is selected and run) by selecting ‘Control’ then ‘Start Data Acquisition’ from the main menu. In this case, only the first four steps listed above occur.

4.0 TestPlan Editor/Debugger

All TestPlan creation is done with the DSBasic Editor/Debugger. It is fully documented in the Cell Assistant on-line documentation.

5.0 Adding Dialogs

Most lines of DSBasic code are used to supervise/control/execute a test; however, DSBasic also provides a programming framework for creating simple dialogs. Simple dialogs can be dynamically created and launched from within a TestPlan. Simple instructions regarding the use of the DSBasic dialog editor as well as how to write DSBasic code to interact with a user via a popup dialog is explained elsewhere in this manual.

6.0 Flight Recorder Files

The only unique TestPlan property (at this time) is the Flight Recorder configuration.

The Flight Recorder is a background task in Cell Assistant that automatically logs test data (e.g. all channel data, limit trips, etc.) to a circular data buffer while a TestPlan is running. This data buffer is referred to as a Flight Recorder. It is mainly used to analyze TestPlans that have shutdown and/or terminated prematurely due to errors. When a TestPlan stops (for any reason), the Flight Recorder will contain a history of all channel data values and other events that occurred for a specified time interval prior to the stopping of the TestPlan. All data in the circular buffer is written to an external file which can be viewed with a standard spreadsheet program.

The 'Flight Recorder Log Rate and Duration' property requires a tradeoff decision between the duration of the data history retained prior to a TestPlan stop versus the rate at which all channel data is saved. The default value for all TestPlans is a 1.0Hz recording of all data for 10 minutes which should be adequate for nearly all TestPlans.

All Flight Recorder files are saved in the "FlightRecorders" subfolder of the main Cell Assistant installation folder (typically C:\CA2012\FlightRecorders). The Flight Recorder filenames are of the form "FlightRecXX.csv" where XX ranges from 00 to 99. When you run your first TestPlan, the file FlightRec00.csv will be created. When the second TestPlan is run, the file FlightRec01.csv is created. When the 100th TestPlan is run, FlightRec99.csv is created. Finally, when the 101th TestPlan is run, a new FlightRec00.csv is created which will replace the previous one. This sequence continues forever. When viewing the files in the "FlightRecorder" subdirectory, always sort the file list by "creation date" or "last modified date" in order to determine which flight recorder file was created last (and hence is the one from the most recent TestPlan run).

Flight Recorder files employ the same format as Data Files (which are described later in this manual). But unlike Data Files, the contents of Flight Recorder files cannot be configured. A Flight Recorder file contains snapshots of ALL TestSystem channels taken at the configured Flight Recorder log rate. Also, whereas Data Files can be opened, closed, started, and stopped under the complete control of a running TestPlan, Flight Recorder data logging starts when a TestPlan starts and stops when the TestPlan stops. Note that in almost all debugging situations, a regular TestPlan Data File is a better choice for data logging because it can be configured to log only the channels of interest and logging can be selectively started and stopped under complete control of the running TestPlan. The Flight Recorder task is designed to be a "backup mechanism" in situations where Data Files are not (or cannot be) used.

7.0 Programming Topics

7.1 Language Extensions

Other chapters in this manual describe how to configure channels, limits, data file formats, etc. These chapters also describe how to access and/or reference channels, limits, data file formats, etc. in a running TestPlan (which is implemented in the DSBasic programming language). The ability to access these items in the DSBasic language is accomplished via language extensions.

DSBasic is an object oriented language. You can programmatically create objects of various kinds and manipulate any objects properties and/or invoke any of an object's methods. Each time a TestPlan is run, Cell Assistant automatically creates a large set of global objects that can be accessed by the TestPlan. One channel object is created for every virtual channel that has been created and configured in the Channels page. Similarly, one limit object and one data file object is created for every limit and data file format that has been created and configured on the respective page.

These global objects act like any other object in the DSBasic language except these objects provide an interface between the running TestPlan and the underlying Cell Assistant data acquisition subsystem. Setting an object to a value (i.e. `object.value = N`) typically just sets a value in some memory location in the computer. Setting the value of a channel object, on the other hand, does more than that. Setting the value of a channel object that happens to be an output channel will cause a command to be sent to a device. Similarly, reading the value of a channel object actually returns a data value that was sampled from a device by the data acquisition subsystem.

This concept needs to be clearly understood. When accessing Cell Assistant objects in a TestPlan, you may actually be communicating with an external device.

7.2 Object References

There are two methods by which any object can be accessed by a running TestPlan. The first method is discussed in the respective chapters on channels, limits, and data file formats. This is the direct method where an object is accessed using its name. For example, every channel object has the same name as the channel it represents. For example, a channel named **"aocDynoSetpoint"** will be represented in the TestPlan by a channel object named **"aocDynoSetpoint"**. You can then send a setpoint value to the Dynamometer controller in the TestPlan using the following line of DSBasic code.

```
aocDynoSetpoint = 100
```

Cell Assistant objects can also be accessed via a "reference" to the object. In DSBasic, you can create an object that can be used to reference other objects of the same type. The reference object is NOT an object itself. It must reference another object in order to be used. An example is shown below

```
Dim aocReferenceToAnyAnalogChannel As AnalogChannel
Set aocReferenceToAnyAnalogChannel = aocDynoSetpoint
aocReferenceToAnyAnalogChannel = 100
```

This example accomplishes the same thing as the previous example. The first line creates (i.e. “Dim”) the reference object. The second line establishes the channel object that will be referenced. Note that the reference object can only reference objects of similar type. In this example, “**aocDynoSetpoint**” must be an analog channel. Finally, in the third line, the value of the reference object is set which actually sets the value of the “**aocDynoSetpoint**” channel. Similarly, any other method/property of the original object can be invoked/modified using the reference object.

This example really does not demonstrate the true usefulness of using object references. Object references are most useful when the same set of operations needs to be performed on several objects. It is easier to write a subroutine that performs these operations on an object reference. Then the “Set” command can be used to set the reference object to the appropriate object prior to calling your subroutine that operates on the reference object.

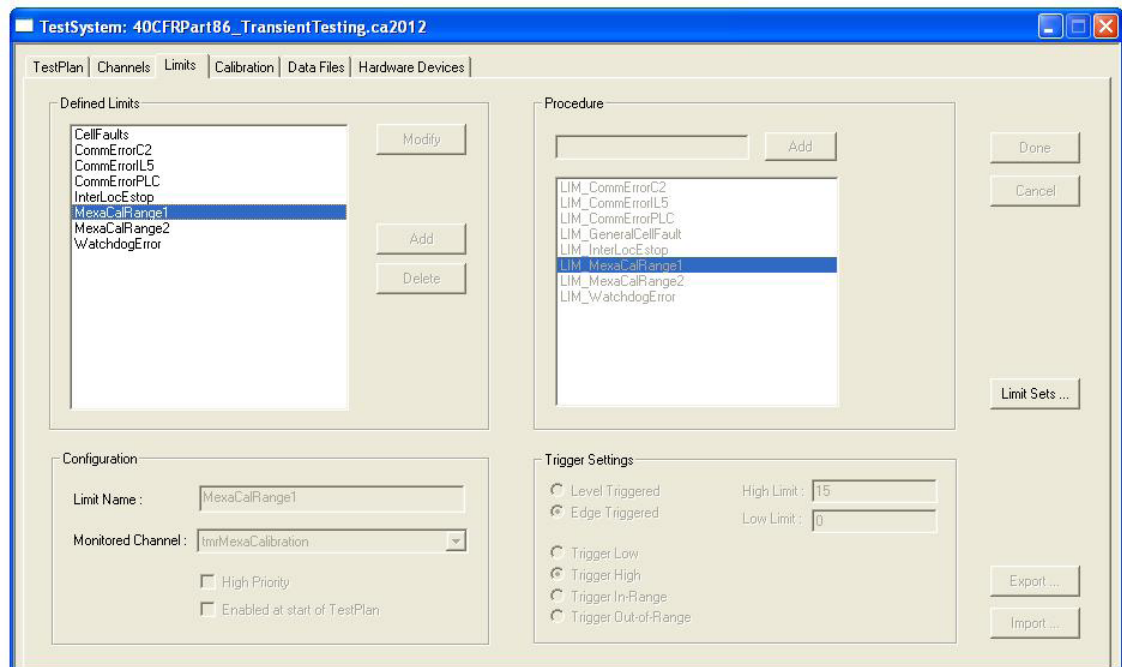
Note: Object references are more likely to be used with limit objects and data file format objects than channel objects.

CHAPTER 7

Limits

Limits are used to provide safety mechanisms in any TestSystem. Limits are used to detect when physical channel values are too high, too low, out of range, etc. When a limit condition is detected, the limit is referred to as being “tripped”. When a limit trips, dedicated DSBasic limit handling functions can be executed to handle the limit as needed (e.g. take corrective action, shutdown the test, etc.)

The Limits page is shown below.



1.0 Adding a Limit

Click the 'Add' button to create a new limit. Enter a unique name for the new limit, select the channel to be monitored, and then click the 'Done' button to save the new limit. The next section describes how to fully configure the new limit.

2.0 Configuring a Limit

Click the 'Modify' button to configure a limit. All limit configuration fields are now enabled.

2.1 Trigger Type

Each limit is either an "Edge Triggered" limit or a "Level Triggered" limit. The Trigger Type determines when and how often the DSBasic limit handler routine is executed when a limit trips.

For an "Edge Triggered" limit, the DSBasic limit handler will only be called at the onset and offset of the trip conditions. The limit handler will be called once when the trip condition is first detected. It will NOT be called again until the trip condition clears itself. Thus, the DSBasic limit handler will be called exactly two times when an "Edge Triggered" limit event occurs. The DSBasic limit handler can easily distinguish if it is being called at the onset or at the offset of a trip condition and, thus, execute a unique set of instructions for each condition. This is explained in a later section.

For a "Level Triggered" limit, the DSBasic limit handler will be repeatedly called as long as trip conditions persist for a given limit. When the limit task determines that trip conditions exist for a limit, the DSBasic limit handler is called. If this handler fails to correct the cause of the trip and/or doesn't disable the limit, the limit task will evaluate all of the other limits and, eventually, will re-evaluate the tripped limit and see that trip conditions still exist. Thus, the limit handler will be called again. This series of events will continue indefinitely until the trip condition is cleared or the limit handler disables this limit (which is typically done immediately).

Note: Most limits are configured to be "Edge Triggered" limits with a limit handler procedure that only processes the onset of a trip.

2.2 Trigger Range

The "Trigger Range" property determines how the limit task evaluates the current value of the "Monitored Channel". The following table describes the various "Trigger Range" configurations. Depending on the specified type, one or both of the limit threshold values must be specified.

Note: The "Low Limit" and "High Limit" threshold values can also be dynamically modified by a running TestPlan as explained in a later section.

Trigger Range	Limit will trip if ...	Low Limit	High Limit
Low	(Monitored Channel value) < (Low Limit value)	Y	-
High	(Monitored Channel value) > (High Limit value)	-	Y
In-Range	(Low Limit value) ≤ (Monitored Channel value) ≤ (High Limit value)	Y	Y
Out-of-Range	(Monitored Channel value) < (Low Limit value) ... OR ... (Monitored Channel value) > (High Limit value)	Y	Y

2.3 Limit Priority

Cell Assistant provides two limit monitoring tasks – a high priority task and a low priority task. Check the “High Priority” checkbox to have the selected limit monitored by the high priority limit task. If not checked, the selected limit will be monitored by the low priority limit task.

High priority limits and low priority limits are continuously evaluated when a TestPlan is executing. If a low priority limit is triggered, the limit procedure (described below) is executed. When this happens, high priority limits continue to be evaluated; however, if another low priority limit is triggered, the corresponding procedure will not be executed until the currently executing low priority limit procedure has finished executing. In other words, the limit handlers for multiple tripped low priority limits are processed sequentially – one low priority limit handler cannot interrupt another low priority limit handler.

On the other hand, when a high priority limit is triggered, execution of the low priority limit procedure WILL BE suspended and the high priority limit procedure will be executed. Similarly, if another high priority limit is triggered, it will be ignored until the currently executing high priority limit procedure has finished executing – high priority limit handlers are processed sequentially as well.

2.4 Limit Procedure

When a limit is tripped, Cell Assistant executes DSBASIC code contained in a limit handler procedure. All limit handler functions reside in a special TestPlan called the LIMITS_TESTPLAN. All currently available limit handler procedures (in the LIMITS_TESTPLAN) will be shown in the “Procedure” list.

Note: It is generally NOT necessary to create a unique limit handler procedure for each defined limit; in most TestSystems the same limit handler procedure can be used to handle many different limit conditions. If so, configure the selected limit by selecting one of the currently listed limit “Procedures”. The selected limit procedure will now be “shared” by multiple limits.

If a new limit procedure is required, enter the procedure name and click the 'Add' button. A default limit handler procedure will automatically be created and added to the end of the LIMITS_TESTPLAN. Note that all limit handler procedure names contain the prefix "LIM_". This prefix is added automatically and does not need to be entered by the TestSystem programmer.

Adding DSBasic code to the actual limit handler procedure is described in a later section.

3.0 Exporting & Importing Limits

The creation and configuration of limits can be managed outside of Cell Assistant using popular tools such as Microsoft Excel. Using the limit Export feature, the entire displayed list of defined limits can be exported to a comma-separated variable (*.csv) file for off-line storage, editing, and/or importing into another TestSystem. Importing standard sets of limits for frequently monitored data channels is an extremely quick and efficient way to set up a new TestSystem.

All currently defined limits are exported as follows.

- 1 Click the 'Export ...' button. An Export Limits pop-up dialog appears.
- 2 Modify the default filename as needed (e.g. change NNN to a valid number) and click the 'Save' button.

Previously exported limits can be re-imported by clicking the 'Import ...' button. A warning dialog appears explaining the dangers of importing limit data from an external file. If the imported file was previously exported from Cell Assistant (and not externally modified in any way), import errors are extremely unlikely. Dismiss the warning dialog, select the import file, and click the 'Open' button. All limits specified in the file will be merged into the existing list of defined limits. If errors occur during the import process, it is extremely important to repair the file and re-import; else, the TestSystem may be left in an unusable state.

Note: When importing limits from a file, limits will be added as needed and existing limits will be modified as needed. All other currently existing limits remain 'as is'. No limits will be deleted.

4.0 Limit Handler Procedure

Each limit has a limit handler procedure (written in DSBasic code) that is called whenever a limit trips. This handler can be very simple (e.g. merely disable the limit and/or or stop the running TestPlan) or it can be complex (e.g. attempt to determine the cause of the trip condition and try to rectify the cause). When and how often the limit handler procedure is called was explained in a previous section.

Shown below is the default limit handler procedure that is automatically created when the ‘Add’ button is clicked on the Limits tabbed page.

```

-----
\
\
Function LIM_MyLimit() As Integer
    If ThisLimit.IsTripped Then
        \ TODO: Add code to handle the onset of an edge-triggered
        \ limit or a level-triggered limit here.
    Else
        \ TODO: Add code to handle the offset of an
        \ edge-triggered limit here.
    End If
    LIM_MyLimit = 0
End Function

```

The code comment fields clearly show where the TestSystem programmer must add additional code. There are two general sections where code should be added – one section to process the onset of a limit trip and another to process the offset (i.e. clearing) of a limit trip condition. As stated previously, typically most limit handler DSBASIC code will be added to the “onset” processing section.

Return Values:

A limit handler function MUST return a value before exiting. This return value determines what the limit task will do when the limit handler has finished execution. The following table lists all possible return code values.

Return Value	Action
0	TestPlan continues to execute.
255	Stop the currently running TestPlan.
All other values	Ignored (reserved for future use).

A limit handler should never attempt to stop a running TestPlan by executing the “End” statement; instead, it should return a value of 255 and let the limit task stop the TestPlan. If a TestPlan is stopped by executing the “End” statement within the context of a limit handler (or TestPlan calculated function also), the results are unpredictable.

5.0 Controlling Limits in a TestPlan

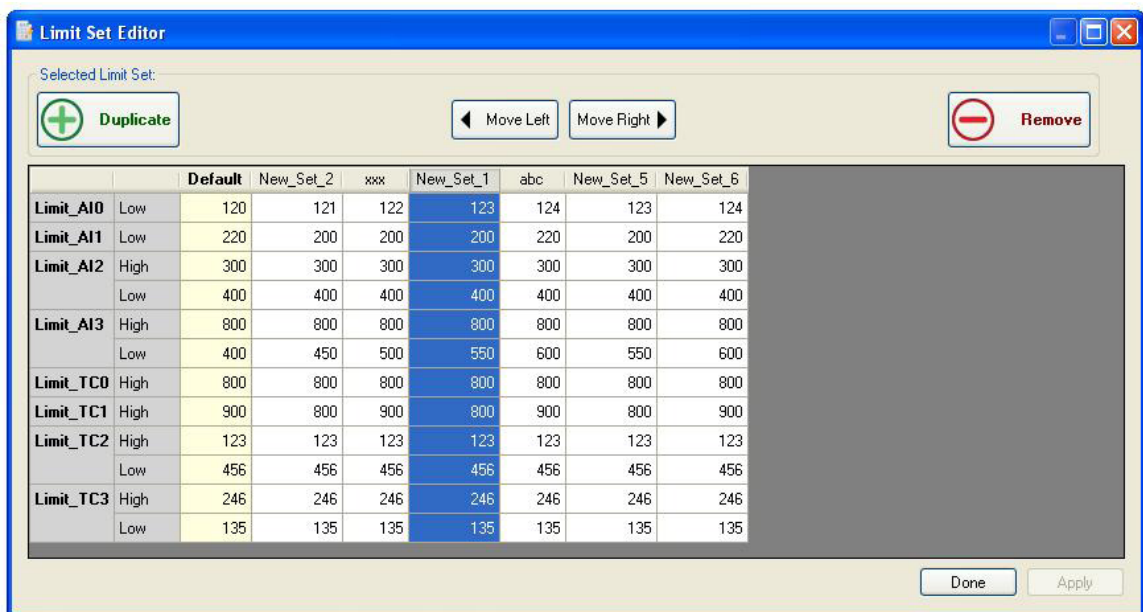
When a TestPlan is compiled and executed, the DSBASIC compiler instantiates a limit object for each defined limit. Using various methods defined for the limit object, a TestPlan can enable/disable limits, change the high and low trip values, and monitor the trip status of any limit. The syntax of these methods is listed below.

Method	Description	Data Type	Example
Enable	Enable the limit. Note the initial enable/disable state of a limit when a TestPlan is initially started is configurable from the Limits tabbed page.	N/A	<code>VoltageLimit.Enable</code>
Disable	Disable the limit. It will no longer be evaluated by the limit tasks. This is often used inside the limit handler to prevent additional trips until the limit condition is corrected and/or cleared.	N/A	<code>VoltageLimit.Disable</code>
Low	Read or set the value of the Low Limit trip point. A running TestPlan can modify the Low Limit trip point using this method; however, each time the TestPlan is started, the Low Limit trip point is always reset to the configured value (on the Limits page).	Double	<pre> TripLevel# = VoltageLimit.Low VoltageLimit.Low = 2.5 </pre>
High	Read or set the value of the High Limit trip point. A running TestPlan can modify the High Limit trip point using this method; however, each time the TestPlan is started, the High Limit trip point is always reset to the configured value (on the Limits page).	Double	<pre> TripLevel# = VoltageLimit.High VoltageLimit.High = 8.5 </pre>
Status	Determines if trip conditions exist (i.e. the value of the monitored channel is too low, too high, out of range, etc.). This method is typically used when a limit has been disabled allowing the TestPlan to leisurely check if trip conditions have cleared before re-enabling the limit.	Boolean	<pre> If VoltageLimit.Status Then ' Trip Conditions Exist. ' Do something here. End If </pre>
IsTripped	Returns the trip state of the limit. This property is subtly different from the Status property. The state of this property will persist until the limit task re-evaluates the limit conditions (even though the actual limit trip conditions may have changed).	Boolean	<pre> If VoltageLimit.Status Then ' Limit has tripped. ' Do something here. End If </pre>

6.0 Limit Sets

When configuring the High and Low limit thresholds for analog input channels, the thresholds are set to values that will protect the Device-Under-Test (DUT) and/or other equipment in the test cell. Unfortunately, the initial set of limit thresholds may not be appropriate for every DUT to be tested. To accommodate the frequent need to adjust the limit thresholds, a “Limit Sets” feature has been implemented. Multiple sets of limit thresholds for analog input channels can be created. The appropriate set of thresholds should be selected at the very beginning of the TestPlan.

Limit sets are created by clicking the “Limit Sets ...” button on the Limits page. The Limit Set Editor will then be launched as shown below.



The High and Low thresholds for all analog input channels are displayed. The “Default” column lists the thresholds that were defined on the main Limits page. Using this editor, any column of thresholds (i.e. a limit set) can be selected, duplicated, and subsequently modified and saved. Each set must be given a unique name as the name is used to set the desired Limit Set at the start of any TestPlan.

The DSBasic compiler creates a unique DSBasic object for controlling the use of Limit Sets. The object name is “LimitSet”. The various methods associated with object are listed below.

Note: The use of Limits Sets is an advanced programming topic. This feature is typically programmed by DyneSystems.

i **ActiveSet**

This method allows a TestPlan to query the name of the active Limit set or change to a new active Limit Set. Several uses of the “**ActiveSet**” method are shown below.

```
activeSet$ = LimitSet.ActiveSet           ` Get name of currently active limit set.
LimitSet.ActiveSet = "NameOfNewLimitSet" ` Change to a new active limit set.
LimitSet.ActiveSet = "Default"           ` Restore use of the "Default" limit set.
LimitSet.ActiveSet = ""                  ` Restore use of the "Default" limit set.
```

Note: When any TestPlan is initially started, the currently active Limit Set is always the “Default” set.

ii **Count**

This method allows a TestPlan to determine the number of configured Limits Sets. It is used along with the “GetNames” method to construct a list of available Limit Sets for display to an operator. The syntax of the “Count” method is shown below.

```
NumSets% = LimitSet.Count
```

iii **GetNames**

This method allows a TestPlan to retrieve the names of all available Limit Sets. The syntax of the “GetNames” method is shown below.

```
Dim LimitSetNames() As String
LimitSet.GetNames LimitSetNames' Retrieve names of all available limit sets.
```

7.0 “ThisLimit” Object

Prior to executing a TestPlan, the DSBASIC compiler instantiates a limit object for each defined limit. The name of each DSBASIC limit object is the same as the configured limit name; thus, all executing DSBASIC code (TestPlan code, calculated function code, limit handler code, etc.) can access and control any specific limit using the respective limit object name.

A problem arises when a generic limit handler is used for more than one limit (which is typical). This “shared” limit handler may need to disable the limit, alter the high or low limit values, log a message, etc. just as any other limit handler would typically do; thus, from within the context of the executing limit handler procedure, the executing code requires a means to determine exactly which limit object has caused the shared handler to execute.

The ability to use a DSBASIC limit handler for more than one limit is provided by the “**ThisLimit**” object as explained below.

Note: Once a limit handler contains references to a specific limit object, it can no longer be used as a shared limit handler for any other limits.

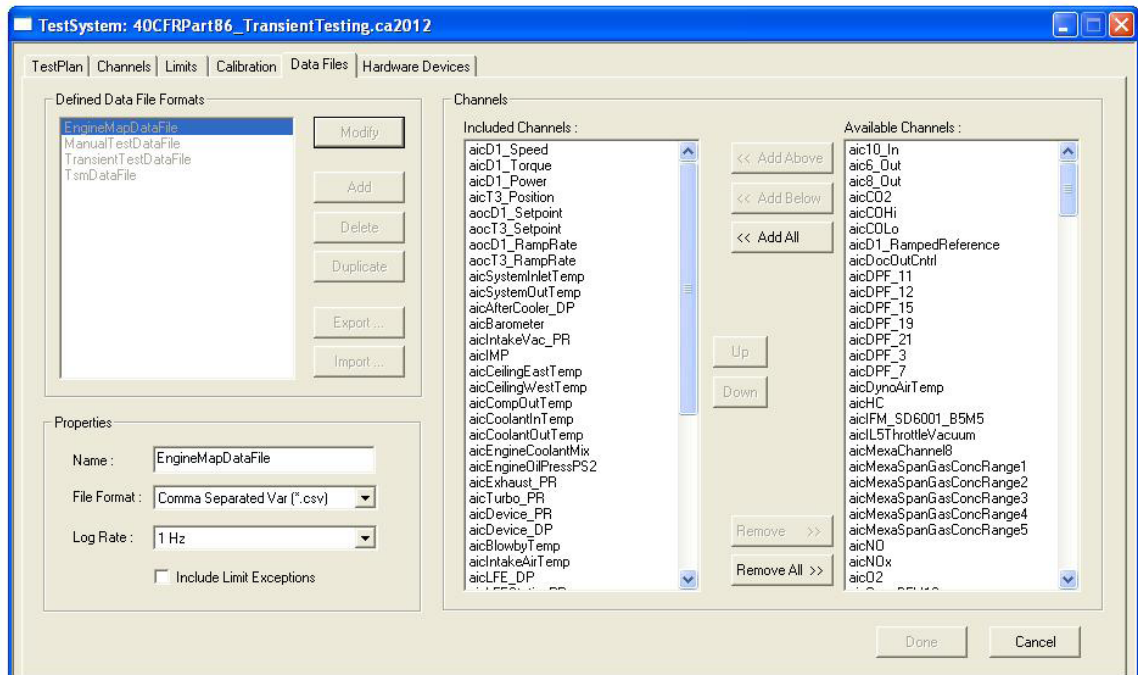
The “**ThisLimit**” object is identical to a limit object. Most (but not all) of the methods that are available for a limit object are also available for the “**ThisLimit**” object. This object only exists during the context of an executing limit handler. The limit task creates this object and sets it to reference the current limit prior to running the limit handler. This object allows methods of the currently active limit to be referenced in the limit handler without specifically referencing any particular limit. The following methods are available.

ThisLimit.Enable
ThisLimit.Disable
ThisLimit.High
ThisLimit.Low
ThisLimit.IsTripped
ThisLimit.Status

CHAPTER 8

Data Files

Recording data is an important function of any data acquisition system. A running TestPlan may open and log data to multiple files simultaneously. This allows low speed and high speed data to be logged at different rates and be stored in different file for post analysis. Asynchronous entries such as limit exceptions, random data snapshots, error messages, etc. can also be inserted into any data file at any time. The Data Files page is used to create data file formats. A file format consists of an ordered list of channels to be logged, the file format used to store the data, and a default logging rate (which can be changed at any time by a running TestPlan). The Data Files tabbed page is shown below.



1.0 Creating a Data File Format

Click the 'Add' button to create a new data file format. Then set all of the properties of the data file format. Note that the 'Name' property is the name of the data file format – NOT the name of the file where the data will ultimately be saved. The actual file name will be specified by the DSBasic script when the data file format is opened and activated. Check the 'Include Limit Exceptions' checkbox if you want the limit tasks to automatically record limit exceptions in the open data file. Note that your limit handler code can also log custom limit handling exception messages.

Click the 'Done' button to save the new data file format.

Click the 'Modify' button to re-open the data file format and begin adding and organizing all channels to be logged by this data file format. Organizing channels is explained below.

Note: A new data file format can also be created by duplicating an existing format. Select the data file format to be duplicated and click the "Duplicate" button. Modify the format as needed, enter a new format name, and click the "Done" button.

2.0 Organizing Channels

When a data file format is enabled for modification, all available channels are alphabetically listed on the right-most list box. This list will ONLY contain channels not already included in the data file format. The various 'Add' and 'Remove' buttons are used to transfer channels back and forth between the 'Included' list and the 'Available' list. Note the 'Add Above' and 'Add Below' buttons require an insertion point to be defined in the 'Included' list. These buttons are only enabled when a single selection exists in the 'Included' list.

Multiple-selection is supported in both channel lists. Simultaneously hold down the CTRL or SHIFT keys to select multiple channels and/or contiguous blocks of channels.

Finally, all channels will be logged in the exact order shown in the 'Included' channel list. Modifications and corrections to the channel logging order can be made using the 'Up' and 'Down' buttons.

3.0 Exporting & Importing Data File Formats

The creation and configuration of data file formats can be managed outside of Cell Assistant using popular tools such as Microsoft Excel. Using the Export feature, a selected data file format can be exported to a comma-separated variable (*.csv) file for off-line storage, editing, and/or importing into another TestSystem. Importing standard data file formats is an extremely quick and efficient way to set up a new TestSystem.

A single data file format can be exported as follows.

- 1 Select a Defined Data File format.
- 2 Click the 'Export ...' button. An 'Export a Data File Format' pop-up dialog appears.
- 3 Modify the default filename as needed (e.g. change NNN to a valid number) and click the 'Save' button.
- 4 Repeat the previous steps to export other data file formats.

Previously exported data file formats can be re-imported by clicking the 'Import ...' button. A warning dialog appears explaining the dangers of importing data file formats from an external file. If the imported file was previously exported from Cell Assistant (and not externally modified in any way), import errors are extremely unlikely. Dismiss the warning dialog, select the import file, and click the 'Open' button. The new data file format specified in the imported file will be added to the TestSystem and should appear in the list of Defined Data File formats. If errors occur during the import process, it is extremely important to repair the file and re-import; else, the TestSystem may be left in an unusable state.

4.0 Data File Contents

A data file contains many types of records (i.e. rows of data). Clearly, the majority of the records are the synchronously logged channel data; however, other types of data records such as file header information, limit exceptions, etc. may also be recorded. The first column of each data row contains a data record "type" field indicating the type (and hence the format) of each data record. This field is necessary for correct parsing of the data file contents. The following table lists the various "type" codes that may exist in the first column of each data row.

Type Code	Description
HD	Header line. Various header lines are automatically added as data files are opened and closed, logging is started and stopped, log rates are changed, etc. Other header lines provide column headers listing the channel names and channel units of each data column. If needed, header lines can be suppressed resulting in a "data only" log file.
TL	Timed logged data line. This line is a synchronously logged snapshot of all channel data. It was automatically recorded by the data acquisition subsystem.
SL	Snapshot logged data line. This line is an asynchronously logged snapshot of all channel data. This snapshot was initiated using the .Snapshot method in a running TestPlan. Snapshot log entries are typically made (as needed) when normal synchronous logging by the data acquisition subsystem is disabled.
LM	Limit Exception data. If enabled as part of the data file format, the limit processing subsystem will automatically add these entries whenever a limit trip condition occurs.
FE	Forced Entry. A running TestPlan can asynchronously log any text string into a data file at any time.

5.0 Using Data File Formats in a TestPlan

Logging data to a data file is completely controlled by a running TestPlan. When a TestPlan is run, the DSBASIC compiler instantiates a unique DSBASIC object for each configured data file format. Using various object methods, new data log files can be created and/or existing data log files can be opened. Data logging is started and eventually stopped. Finally the data log file is closed causing all data to be permanently stored on the local hard disk drive. This section describes the syntax for these object methods. They are also described in the on-line help files.

5.1 Open

A data file is initially created (or appended to) using the “**Open**” method. Multiple data files can be opened at any time as long as each uses a different configured data file format. In other words, for any single data file format, only a single data file can be open and active at any time. The syntax of the “**Open**” method is shown below.

```
FormatName.Open "Filename" [, Mode, [NoHeaderFlag]]
```

FormatName is the name of the configured data file format. “**Filename**” (quotes required) is the actual name of the file being created. Note that “**Filename**” is a string and must NOT contain a path or extension. The file extension (.csv or .txt) will be added based on the configured File Format type while the data file will be logged to the DataFiles subfolder. **Mode** determines whether the file will be added (i.e. appended) to or overwritten. If added to, data will be added to the end of the file. The allowed **Mode** values are “**Add**” or “**Overwrite**”. **Mode** is an optional parameter. If omitted, the default mode will be to add to the file. The following are all valid uses of the “**Open**” method.

```
Format1.Open "MyDataFile", Add
Format1.Open "MyDataFile", Overwrite
Format1.Open "MyDataFile" (defaults to Add mode)
Format1.Open "MyDataFile", Add, NH
```

When the “**Open**” method is executed, Cell Assistant automatically writes three lines of header data (identified by “HD” in the first column). The first line documents when the file was opened and which format was used. The second line lists the channel names. The third line lists the units for each channel.

An optional **NoHeaderFlag** parameter allows a data file to be opened without adding the three header lines to the data file. This allows the creation of “numbers only” data files that may be easier to analyze later on. To prevent the logging of the header lines, set the **NoHeaderFlag** to “**NoHeader**” or “**NH**”. If this parameter is omitted, the default action is to always log the header lines.

5.2 Close

This method closes a currently open data file. The syntax of the “**Close**” method is shown below.

```
FormatName.Close [NoHeaderFlag]
```

The following are valid uses of the “**Close**” method.

```
Format1.Close
Format1.Close NH
```

When the “**Close**” method executes, Cell Assistant automatically writes one line of header data (identified by “HD” in the first column). This line records the format of the data file being closed as well as the time and date.

An optional **NoHeaderFlag** parameter allows the data file to be closed without adding the header line to the data file. This allows the creation of “numbers only” data files that may be easier to analyze later on. To prevent the logging of the header line, set the **NoHeaderFlag** to “**NoHeader**” or “**NH**”. If this parameter is omitted, the default action is to always log the header line.

5.3 IsOpen

This method allows a TestPlan to check if a data file is currently open using the specified data file format. If so, this method returns TRUE. If not, this method returns FALSE. The syntax of the “**IsOpen**” method is shown below.

```
FormatName.IsOpen
```

The following is a valid usage of this method. It is typically used in shutdown and error handling routines that perform cleanup after an error. The TestPlan can check if a data file is actually open and, if so, close it.

```
If Format1.IsOpen Then
    Format1.Close
End If
```

5.4 StartLogging

This method initiates synchronous data logging to the currently open data file (i.e. the “**Open**” method must have been previously called). The syntax of the “**StartLogging**” method is shown below.

```
FormatName.StartLogging [Rate[, NoHeaderFlag]]
```

Rate is an optional parameter. If present, it specifies the logging rate in Samples/sec (i.e. Hz) at which the data acquisition subsystem will add data entries to the currently open log file. If **Rate** is not specified, the default logging rate (specified in the data file format) will be used. Valid **Rate** values are 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, and 20.0 Hz. The following examples are valid uses of the “**StartLogging**” method.

```
Format1.StartLogging
Format1.StartLogging 0.01
Format1.StartLogging 10
Format1.StartLogging 10, NH
```

When the “**StartLogging**” method is executed, Cell Assistant automatically writes a line of header data (identified by “HD” in the first column). This entry documents exactly when logging was started and at what rate. Afterwards (and at the specified logging rate), Cell Assistant automatically writes lines of data which contain the values of the specified channels. These data records are identified by “TL” in the first column.

The optional **NoHeaderFlag** parameter allows data logging to be started without adding the header line to the data file. This allows the creation of “numbers only” data files that may be easier to

analyze later. To prevent the logging of the header line, set the **NoHeaderFlag** to “**NoHeader**” or “**NH**”. If this parameter is omitted, the default action is to always log the header line.

5.5 StopLogging

This method causes the data acquisition subsystem to stop logging data to the currently open file. The syntax of the “**StopLogging**” method is shown below.

```
FormatName.StopLogging [NoHeaderFlag]
```

Valid uses of the “**StopLogging**” method are shown below.

```
Format1.StopLogging  
Format1.SopLogging, NH
```

When the “**StopLogging**” method is executed, Cell Assistant automatically writes a line of header data (identified by “**HD**” in the first column). This data entry documents exactly when logging was stopped.

The optional **NoHeaderFlag** parameter allows data logging to be stopped without adding the header line to the data file. This allows the creation of “numbers only” data files that may be easier to analyze later on. To prevent the logging of the header line, set the **NoHeaderFlag** to “**NoHeader**” or “**NH**”. If this parameter is omitted, the default action is to always log the header line.

5.6 LogRate

This method changes the rate at which the synchronous data sample records are written to the currently open data file. The syntax of the “**LogRate**” method is shown below.

```
Format1.LogRate = Rate
```

Rate specifies the logging rate in Samples/sec (i.e. Hz) at which the data acquisition subsystem will add data entries to the currently open log file. Valid **Rate** values are 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, and 20.0 Hz. The following examples are valid uses of the “**LogRate**” method.

```
Format1.LogRate = 0.02  
Format1.LogRate = 5.0
```

5.7 LogPeriod

This method changes the rate at which the synchronous data sample records are written to the currently open data file. Functionally, this command is identical to the “**LogRate**” method except the data logging rate is specified as a period (i.e. seconds between successive samples). This method is generally more useful when specifying very slow sampling rates (i.e. below 1 Hz). The syntax of the “**LogPeriod**” method is shown below.

```
Format1.LogPeriod = Period
```

Period specifies the logging period (in Sec) between successive data samples in the currently open log file. Valid **Period** values are 0.05, 0.1, 0.2, 0.5 seconds or any value between 1 and 600 seconds. The following examples are valid uses of the “**LogPeriod**” method.


```
Format1.LogPeriod = 0.1
Format1.LogPeriod = 5.0
Format1.LogPeriod = 60.0
```

5.8 IsLogging

This method allows a TestPlan to check if a data file is currently logging data. If so, this method returns TRUE. If not, this method returns FALSE. The syntax of the “IsLogging” method is shown below.

```
FormatName.IsLogging
```

The following is a valid usage of this method. It is typically used in shutdown and error handling routines that perform cleanup after an error. The TestPlan can check if a data file format object is currently logging data to a file and, if so, disable logging before closing the file.

```
If Format1.IsLogging Then
    Format1.StopLogging
End If
```

5.9 SnapShot

This method writes a single snapshot of all configured channel values into the currently open data file. It is typically called when synchronous data logging is currently not active. The syntax of this method is shown below.

```
Format1.SnapShot
```

5.10 ForceEntry

This method allows any text string to be added to the currently open data file. The syntax of this method is shown below.

```
Format1.ForceEntry "Your string goes here."
```

The following examples are valid uses of the “ForceEntry” method.

```
Format1.ForceEntry "Operator Name = Tom Smith"
Format1.ForceEntry "Engine Serial Number is XY01234"

Format1.ForceEntry "Test aborted due to coolant over-temperature".
```

5.11 WriteChannelNames

This method adds a header record listing the names of all channels included in the currently open data file format. This header line is exactly the same as the header line recorded by the “**Open**” method when a data file is initially opened or created. This method is typically used to periodically re-write all channel names in order to improve the readability of the data file. It can also be used to add channel name information that may have been previously suppressed in the “**Open**” method with the “**NoHeader**” flag. The syntax of the “**WriteChannelNames**” method is shown below.

```
Format1.WriteChannelNames
```

5.12 WriteChannelUnits

This method adds a header record listing the configured units of all channels included in the currently open data file format. This header line is exactly the same as the header line recorded by the “**Open**” method when a data file is initially opened or created. This method is typically used to periodically re-write all channel units in order to improve the readability of the data file. It can also be used to add channel unit information that may have been previously suppressed in the “**Open**” method with the “**NoHeader**” flag. The syntax of the “**WriteChannelUnits**” method is shown below.

```
Format1.WriteChannelUnits
```

5.13 WriteChannelHeader

This method is equivalent to successively calling the “**WriteChannelNames**” and “**WriteChannelUnits**” methods except this method guarantees the channel names header record and the channel units header record are logged in consecutive rows in the data file. The syntax of the “**WriteChannelHeader**” method is shown below.

```
Format1.WriteChannelHeader
```

CHAPTER 9

Virtual Instrument Panels

Virtual Instrument Panels (i.e. VIP's) are on-screen displays of real-time data as it is acquired by Cell Assistant. Each VIP consists of a large Windows form containing graphical display objects such as analog meters, gages, buttons, pilot lights, etc. allowing a user to interact with a running TestPlan in Cell Assistant. Each VIP should be designed to optimally display data for a specific TestPlan. To improve readability and viewing of all live data, multiple VIP's can be created and launched by a single TestPlan if needed.

VIP's are executable programs developed in Visual Basic using standard development tools such as Microsoft Visual Basic Express or Microsoft Visual Studio. Once created, they are compiled into a standard executable file format (*.exe file) and placed into the 'VIPs' sub-folder of the main Cell Assistant installation folder. From that folder, a running TestPlan can launch and close any VIP as needed.

All of the graphical display objects (e.g. meters, gages, etc.) are provided in the form of an ActiveX control. This control (VIPCONTROLS.ocx) is automatically installed and registered during Cell Assistant product installation.

Note: Visual basic programming and the general use of Microsoft Visual Basic (in general) is not discussed here. Please familiarize yourself with this product via practice as well as reading all on-line documentation. **NO ACTUAL PROGRAMMING IS REQUIRED.** It is only necessary to place controls on a form, modify their visual appearance, and connect them to a Cell Assistant virtual data channel. This can all be done graphically.

1.0 Microsoft Visual Studio Express 2012 (or similar)

Microsoft Visual Studio Express 2012 should be installed on the Cell Assistant computer. Follow the product installation instructions. All default folder settings should be used - this includes a default folder structure to store and manage all Visual Basic projects.

Before a VIP can be created, the DyneSystem's VIP Controls must be added to the Toolbox palette. Perform the following additional steps when creating your first VIP form.

- 1 Display the Toolbox palette (if not already displayed).
- 2 Add a new TAB to the palette and rename this tab to 'DyneSystems'.
- 3 Locate the menu item to add components to this new TAB.
- 4 Select "COM Components".
- 5 Browse for the file "VIPCONTROLS.ocx" (usually located in the C:\WINDOWS\SysWOW64 folder).
- 6 Click the 'Open' button.
- 7 All graphical controls will now be added to the Toolbox palette.

All graphical controls contained in VIPCONTROLS.ocx will now appear on the Toolbox palette and are available for adding to any new VIP.

2.0 Creating a VIP

Launch Microsoft Visual Studio Express 2012 (or similar) and create a new "Windows Forms Application" project. A single blank form will be displayed. If this is your first project, follow the setup steps listed in the previous section in order to install all DyneSystem's VIP Controls on to the Toolbox palette.

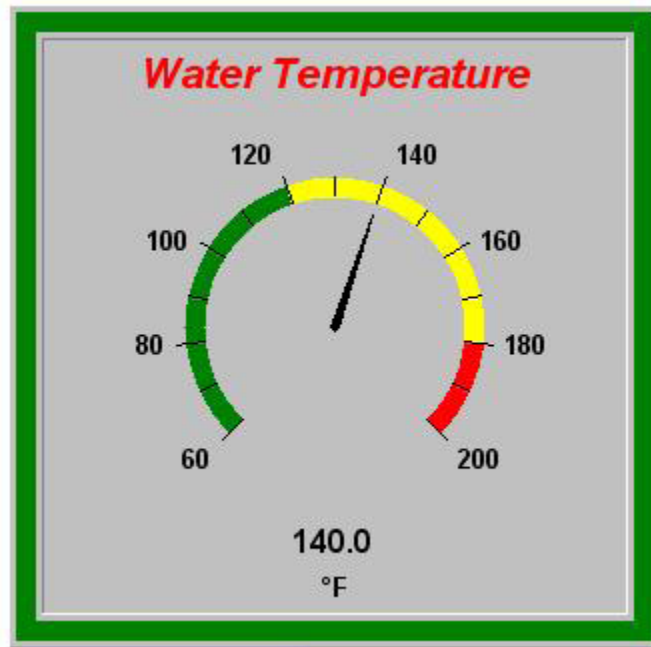
It is much easier to create VIP's when data acquisition is currently running in Cell Assistant. If data acquisition is running, all graphical control objects can access TestSystem virtual channel and limit information in real-time; therefore, before creating your VIP, launch Cell Assistant, load the TestSystem file that will use this VIP, and start data acquisition.

Now add various VIP controls to the form as needed. All controls are discussed briefly in the next section. Building a VIP is a simple process of adding various controls to a form, connecting them to a virtual channel in Cell Assistant, and then adjusting all of the various control properties in order to obtain the desired size and appearance of the control. By default, many of the appearance properties of each control are initially determined by the channel they are connected to. For example, captions, units, numerical precision of readouts and scale numbering, On/Off text, etc. are initially taken from the Cell Assistant defaults for the connected channel. For each property, however, you can uncheck the "Use channel default" box and enter your own value.

When done, close the project and specify reasonable names for all forms files, project files, solution files, project folders, etc.

Limit Information – Scale Colors:

Many of the meters can also make use of limits configured in Cell Assistant. The limits which are available for any control depend on the selected channel. Only limits that are monitoring the selected channel can be used. These limits affect how the meter scales are drawn. Meter scales are always the color GREEN by default. Up to four of the available limits can be selected. Once selected, each limit must be assigned a color of RED or YELLOW. The control will then repaint part of the scale based on the assigned color and High and Low limit values of each selected limit. An example is shown below.



In this example, two limits that were monitoring on the selected channel were also selected to be used for this control – both were High limits (that overlap). The first High limit has a trip point of 180 with the color RED assigned. The other High limit has a trip point of 120 with the color YELLOW assigned.

To this point, the only advantage of selecting limits is to provide an operator with visual feedback about limit information for a channel by painting limit ranges on the meter scale. The real power of using limits with a meter is explained in the next section which describes how the meter border color changes when one of the selected limits trips in Cell Assistant.

Limit Information – Border Colors:






The border color of a meter is always colored GREEN just like the meter scale; however, if limits are selected, the border color can be used as a highly visual indicator of limit trip conditions. If one of the selected limits trips, the border will change to the same color that has been assigned to that particular limit. The border will stay that color until the limit un-trips (i.e. the trip condition is cleared and reset).




Note that the border color has NOTHING to do with where the needle is pointing. For example, the needle could be pointing in the RED range, yet Cell Assistant might not have that particular limit

currently enabled. Thus, the limit will not trip and the border will not change color. Similarly, a limit can trip in Cell Assistant which causes the border color of all affected meters to change color. The actual value of the channel may then change which causes the needle to now point to a GREEN portion of the scale. But the Cell Assistant limit handler (in DSBasic code) may still be processing the limit and has not yet allowed the limit trip state to return to normal. Thus, the border colors remain in their tripped state.

3.0 VIPCONTROLS.ocx

The VIPCONTROLS ActiveX control provides a set of meters, gages, etc ... allowing high-quality user-interfaces to be created with absolutely NO programming whatsoever. The controls can be placed on a Visual Basic form and then totally configured using standard property pages. Each of these controls is briefly described below. The details of each control property are not discussed since most have an obvious affect on the controls operation and/or appearance.

Control	Icon	Description
Analog Meter		A simple needle-type meter that continuously displays the value of any analog channel.
Digital Meter		A simple numerical readout of any analog channel.
Bar Meter		Another popular meter style used to continuously display the value of any analog channel.
Text Meter		The Text Meter has three modes of operation depending on the configured Input Source. (1) If connected to a text channel, the meter simply displays the value of the text channel. (2) If connected to a digital channel, the meter displays the ON text or OFF text depending on the current state of the digital channel. (3) If the meter is connected to a limit, the meter displays the configured "tripped" or "not tripped" text based on the current tripped state of the connected limit.
Pilot Light		The Pilot Light has three modes of operation depending on the configured Input Source. (1) If connected to a digital channel, the color of the lamp is determined by the state of the digital channel. (2) If connected to a device channel, the color of the map is determined by whether (or not) the value of the device channel exactly matches the configured "Match value". (3) If the lamp is connected to a limit, the color of the lamp is determined by the trip state of the connected limit.

Control	Icon	Description
Push Button		<p>The Push Button control is simultaneously an input and output device. When clicked, data is written to the connected output channel. On the other hand, at all times, the appearance of the button is determined by the state of the connected input channel. Using a separate input channel in this way allows the creation of closed-loop button operations. If this feature is not needed, a checkbox exists allowing the configured output channel to be used as the input channel as well.</p> <p>The connected output channel can be a digital or device channel. When connected to a device channel, the configured “Output value” will be written to the connected channel whenever the button is clicked.</p> <p>When connected to a digital channel, the output value written in response to a button click is configurable. (1) A TRUE constant can be written to the channel for every button click. (2) A FALSE constant can be written to the channel for every button click. (3) The last written 0/1 value can be toggled and written to the output channel. (4) The current 0/1 value of the connected input channel can be read, toggled, and then written to the output channel.</p> <p>Note: Always be aware this button will drive the state of the output channel just as a running TestPlan in Cell Assistant drives the state of the channel programmatically using DSBasic commands. Thus, if the output channel is an output channel for a hardware device, clicking the button will most likely cause commands to be sent to an external device.</p>
Strip Chart		<p>The StripChart control allows up to 6 Analog channels to be continuously plotted in real time. The Y-axis scale information for only one channel is displayed at a time. Click the Channel button to cycle through the Y-axis information for the other channels. The color of this information will match the color of the appropriate channel on the stripchart.</p>
Computer Select		<p>The ComputerSelect control is used to create a VIP that runs over a network. The absence/presence of this control determines whether this particular VIP tries to access data on the local computer or on a remote computer on the network. Network VIP's are explained below.</p>

4.0 Network VIP's

By default, all VIP's make a real-time connection to Cell Assistant running on the same computer. This is done internally via shared system memory. Building a VIP that runs on another computer is a very simple task of forcing the VIP to connect to Cell Assistant via a network connection. This is done by adding the ComputerSelect control to any VIP.

The absence/presence of this control determines whether this particular VIP tries to access data on the local computer or on a remote computer on a network. In the absence of a ComputerSelect control, all controls on a VIP attempt to connect to a Cell Assistant virtual channel that exists on the

local computer. As soon as a ComputerSelect control is added to a Visual Basic form, all controls rely on the ComputerSelect control for their data.

The ComputerSelect control has two properties: The **Computer** name and the **TCP/IP Port** of the remote computer that this control should try to connect to. You can enter either the name of the computer or the IP address of the computer. If the remote computer exists on a local network, you can also click the **Browse** button to get a list of available computers and then select the desired computer graphically. The **TCP/IP Port** must match the TCP/IP port that is currently being used by the “Network VIP Data Server” that is running on the remote Cell Assistant computer. Typically, the default value should be specified.

If Cell Assistant is running on the specified computer (and it is configured to support network connections), this control will immediately connect to the remote computer. When the ComputerSelect control is first placed on a form, it will read “Not Connected”. Once a connection is established, the controls appearance will change to a progress bar providing visual feedback of network activity when the VIP is run.

At this point, you can continue to develop and/or run VIP’s in the usual manner. Keep in mind you are now connected to a remote Cell Assistant computer via a network connection that may have long network latencies. These delays are usually quite tolerable if running entirely in a local Intranet; however, these delays can be longer if the remote computer is farther away (in a network sense).

Note: The most efficient process to develop a network VIP is to fully develop the VIP on the Cell Assistant local computer. When done, simply add and configure the ComputerSelect control and the VIP can now be moved to another computer.

5.0 Using VIP’s in a TestPlan

Only a few lines of DSBasic code are needed to launch a Virtual Instrument Panel as shown below. The full path to the VIP executable file as well as the VIP main window title must be known. The window title is needed in order to check if the VIP is already launched (e.g. it may have been externally launched by an operator using Windows Explorer and/or it was not properly closed during a previous TestPlan execution).

```
Sub Main()
    ' Title and path to my VIP.
    MyVipTitle$ = "Main VIP"
    MyVipPath$ = "C:\CA2012\VIPs\MainVIP.exe"

    ' Launch VIP (if not already running).
    If AppFind$( MyVipTitle$ ) = "" Then
        id = Shell( MyVipPath$, vbNormalNoFocus )
    End If
End Sub
```


Similarly, the previously launched VIP can be closed using the following lines of DSBASIC code.

```
Sub Main()
    ` Title of my VIP.
    MyVipTitle$ = "Main VIP"

    ` Close VIP (if needed) .
    If AppFind$( MyVipTitle$ ) <> "" Then
        AppClose MyVipTitle$
    End If
End Sub
```

Note: All of the DSBASIC function calls used in the above examples (i.e. AppFind\$, AppClose, and Shell) are fully described in the on-line DSBASIC reference manual.

6.0 VIP Driver

Each meter, gauge, button, etc. in a VIP is connected to (i.e. bound to) a virtual channel in a Cell Assistant TestSystem. And virtual channels are created from the various physical device channels provided by the installed device drivers. There are many programming situations where a VIP and the associated TestPlan need to exchange data that is NOT provided by a physical device (e.g. not a measured voltage or thermocouple reading). For these situations, a pseudo-device driver called the "Virtual Instrument Panel Interface" driver is available.

This driver can be added to a TestSystem (using the Hardware page) just as any other hardware device driver is added. This driver presents large arrays of generic pseudo-channels that can be used to create non-hardware virtual channels. These non-hardware virtual channels (obviously) are not connected to any device and are not updated by the data acquisition sub-system; therefore, a running TestPlan and an associated VIP are free to read/write/exchange data using these channels as needed.

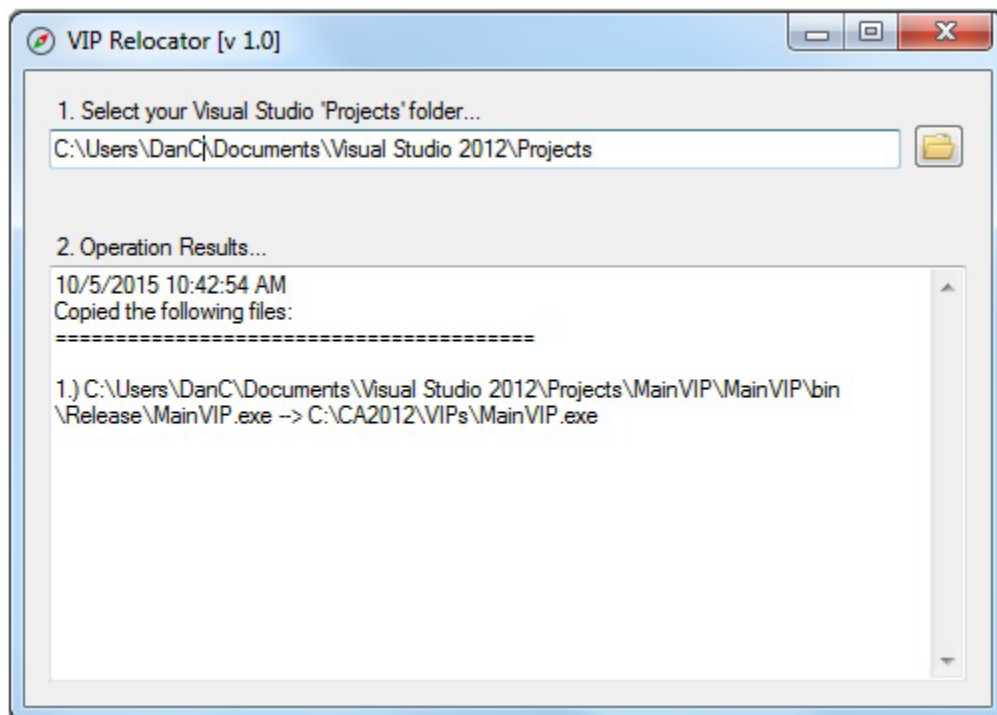
7.0 VIP Relocator

As explained in other sections of this chapter, Virtual Instrument Panels are created by building Windows "Forms" applications using the latest version of Microsoft Visual Studio Express. The resulting VIP is an executable file that must be copied to the Cell Assistant VIPs folder. Visual Studio also creates two dynamic link libraries (i.e. DLL's) that must be copied along with the VIP executable. Because this is a tedious and mistake prone process, a special application was created to facilitate the coping of all files. It is called the "VIP Relocator". This program should be executed every time any change is made to a VIP and the VIP executable is subsequently rebuilt.

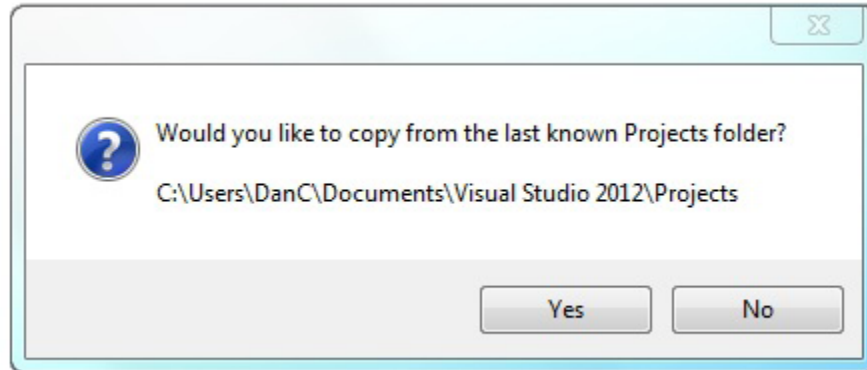
During installation, a desktop shortcut to the VIP Relocator program is installed as shown below. Double-click the icon to launch the program.



The following window will appear. On the first launch of the application, all fields will be blank. Click the “folder” icon to browse for the path to the main “Projects” folder where all VIP projects reside under. Once selected, the relocater will then scan all subfolders for VIP project executables and copy them the Cell Assistant VIPs folder. All copied files are displayed in the “Results” window.



On subsequent launchings of the VIP Relocator, the projects path has already been established. In this case, the following confirmation dialog is displayed. Click the Yes button to copy all files.



CHAPTER 10

DSBasic Dialog Editor

While the previous chapter describes how to create rich looking Virtual Instrument Panels capable of displaying acquired data in real time, the DSBasic framework provides another graphical user interface mechanism – the Dialog Editor. The Dialog Editor is a perfect choice for creating simple user interfaces that display relatively static data and/or for gathering user input prior to a test. Simple dialogs consisting of buttons, lists, text entry, check boxes, etc. can be created. These dialogs can (if needed) be used to display real-time data; however, they are cosmetically limited and somewhat difficult to program. It is best to create VIP executables for real-time data.

A simple dialog created with the DSBasic Dialog editor requires three coding tasks. These tasks are discussed briefly below.

- 1 Define and create the dialog using the Dialog Editor.
- 2 Add a few lines of DSBasic code to a TestPlan to launch the dialog.
- 3 Create a DSBasic handler function to interact with the launched dialog.

Note: Please consult the DSBasic on-line reference manual for more information regarding the editor as well as the following DSBasic functions.

- **Begin Dialog** (statement)
- **Dialog** (function)
- **DlgProc** (function)

1.0 Creating a dialog

The Dialog Editor creates a dialog template that fully describes the appearance and functionality of the created dialog. So before launching the dialog editor, click the cursor in the TestPlan editor to mark the location where the Dialog Editor will place the dialog

template when done. From the main menu, select 'TestPlan' then 'Dialog Editor ...' to launch the Dialog Editor.

Proceed to build the user interface dialog as needed. When complete, select 'File' then 'Exit and Return' in the Dialog Editor. A dialog template that fully describes the created dialog will be placed in the currently open TestPlan. A sample template containing two buttons and a text box is shown below.

```
Begin Dialog TestInputDialogTemplate , ,178,160,"Test Parameters"
    OKButton 132,8,40,14
    CancelButton 132,28,40,14
    TextBox 20,16,36,12,.EnterUserName
End Dialog
```

Note: This template can be modified/updated by fully selecting the entire template (i.e. select all text from the **Begin Dialog** to the **End Dialog** statements, inclusive) and then re-launching the Dialog Editor.

2.0 Launching a dialog

Launching the dialog is a two step process – (1) create an instance of the dialog template and then (2) launch the newly created instance. This is accomplished using the following lines of DSBasic code.

```
Dim InputDialog As TestInputDialogTemplate
result% = Dialog (InputDialog)
```

The first line of DSBasic code creates an instance of the dialog template and the second line of DSBasic code launches the instance. Note that the '**Dialog**' statement does not return until the dialog is closed. A result code is returned indicating the exact dialog button that was clicked to dismiss the dialog. See the on-line help for the '**Dialog**' function for more information regarding the returned result code.

3.0 Interacting with a dialog

Many dialogs interact with an operator in order to gather various types of user input, validate parameters, etc. To build more powerful user dialogs, a "dialog proc" function can be added to any dialog template. This function is simply a DSBasic function with a pre-defined format that interacts with a running dialog. The DSBasic run-time engine calls this function as needed in order to allow various events to be handled. For example, the DSBasic run-time engine will call the "dialog proc" function when the dialog is initially displayed allowing your code to initialize all controls and data fields. Similarly, the function will be called whenever a button is clicked, a list selection is changed, a

checkbox is checked/unchecked, etc. A “dialog proc” function can be added to the previous dialog template example by modifying the template as follows.

```
Begin Dialog TestInputDialogTemplate , ,178,160,"Test Parameters",.TestInput
    OKButton 132,8,40,14
    CancelButton 132,28,40,14
    TextBox 20,16,36,12,.EnterUserName
End Dialog
```

Note: The first line contains an additional parameter – the name of the “dialog proc” function. Please see the on-line documentation for the **DlgProc** (function) for information regarding the format, calling parameters, and implementation of this function. This is an advanced programming topic and will not be further explained here.

CHAPTER 11

Calibration

Calibration allows the TestSystem operator to reduce/eliminate measurement errors in many analog input and analog output signals. Calibration of analog signals is a required component of many standardized testing protocols (e.g. SAE, 40CFR, ISO, etc.). Cell Assistant provides several methods of calibration; the selected method is usually based on the availability of calibrated reference sources and/or is dictated by a mandated testing standard.

The Calibration tabbed page manages the calibration of all calibratable signals. This page is used to review the previous calibration results of each signal as well as to backup and restore the calibration database.

1.0 Calibration Equation

Each signal is calibrated by applying an OFFSET and a GAIN correction according to the following equation.

$$(\text{Calibrated Value}) = (\text{GAIN correction}) * (\text{Un-calibrated Value}) + (\text{OFFSET correction})$$

The Un-calibrated value represents the raw channel reading from the connected hardware device. The GAIN correction and OFFSET correction values are determined by the most recent calibration procedure performed on the respective channel. Finally, the Calibrated Value is the corrected value that is used throughout Cell Assistant when a TestSystem is executing.

2.0 Calibration Types

Cell Assistant provides three calibration procedures – 1 Point, 2 Point, and Manual. The method used to calibrate any channel is often dictated by a mandated testing standard and/or is restricted due to the availability of calibrated references sources. 1 Point calibration requires exactly one calibrated reference source. The channel reading can then be “zeroed” only at that reference value. Since a second reference point is not used, only the OFFSET correction can be computed.

2 Point calibration requires two calibrated references sources allowing the channel reading to be “zeroed” at each reference value. When a 2 Point calibration is performed, both the OFFSET and GAIN correction values can be computed.

Finally, Manual calibration requires no calibrated references sources. The GAIN and/or OFFSET correction values are known. For example, they are available on a calibration sheet provided with a particular transducer. In this case, the GAIN and OFFSET correction values are simply entered into the calibration database. No actual calibration procedure is performed.

3.0 Calibration Procedure

Any channel can be calibrated (or re-calibrated) by selecting the channel on the Calibration tabbed page and then clicking the ‘Calibrate’ button. The following calibration page is then displayed.

First select the calibration 'Method' to be used.

If Manual calibration is selected, the Gain Correction and Offset Correction fields will be enabled allowing the calibration operator to simply enter the desired values.

If 1 Point calibration is selected, a reference source (e.g. a calibrated temperature, pressure, or voltage) should be applied to the respective transducer. Click the 'Low Point' Start button to start the data acquisition subsystem. The un-calibrated reading of the channel will be displayed. When the reading is stable, enter the known calibrated value of this reading. The 'Accept' button will become enabled. The operator can now click the 'Accept' button to accept the calibration at this value. Or the 'STOP' button can be pressed to stop data acquisition and perform no calibration at this time

Note: The 'Start' button text changes to 'STOP' when data acquisition is started.

If 2 Point calibration is selected, the previous procedure must be performed two times – once at a Low Point calibrated reference value and again at a High Point calibrated reference value.

Note: For better calibration accuracy, the calibrated reference points should be as far apart as possible (with respect to the available span of the respective transducer).

4.0 Calibration Database

The calibration database is a single file containing the results of every calibration performed on this local computer. The calibration database records the exact device and physical channel name of each calibrated channel; thus, if the same device channel is used in multiple TestSystems, the channel need only be calibrated by one of the running TestSystems. Afterwards, the calibration is applied to every use of the channel by all other TestSystems.

The calibration database file is located in the Calibrations sub-folder of the installation folder (typically C:\CA2012\Calibrations\CalibrationDatabase.db). The contents of this file should NOT be manually edited. Corruption of the contents of this file may render the entire database useless. The calibration database should be backed up as needed (as explained in the next section).

5.0 Exporting & Importing Calibration Database

The entire calibration database can be backed up/exported to a comma-separated variable (*.csv) file for off-line storage as follows.

- 1 Click the 'Export ...' button. A Save Calibrations pop-up dialog appears.
- 2 Modify the default filename only if needed (e.g. change NNN to a valid number) and click the 'Save' button.

A previously exported calibration database can be re-imported by clicking the 'Import ...' button and selecting the calibration file to import.

Note: When importing a saved calibration file, new calibrations will be added/merged to the existing calibration database as needed. If a calibration entry already exists for a given channel, the calibration data will be updated. All other currently existing calibration entries remain 'as is'. No calibration entries will be deleted.

Each row of an exported calibration file contains the calibration data for one channel as listed below.

Column	Data
A	Physical Channel Name
B	Physical Device Name
C	Name of Virtual channel used during calibration
D	Name of TestSystem that performed the calibration
E	Date/Time stamp
F	Method of calibration
G	Actual Low Point (un-calibrated reading of channel)
H	Desired Low Point (desired reading of channel)
I	Actual High Point (un-calibrated reading of channel)
J	Desired High Point (desired reading of channel)
K	Gain Correction Factor
L	Offset Correction Factor
M	Comments

CHAPTER 12

Preferences

Several aspects Cell Assistant can be configured by a TestSystem developer. These application settings and/or features are grouped into several tabbed preferences pages. Each preference tab is discussed below. The application preferences are accessed by selecting 'File' then 'Preferences ...' from the main system menu.

1.0 Appearance

These settings only affect the appearance of the application. Each should be obvious and are not discussed here.

2.0 Features

The "Network VIP Data Server" can be enabled and configured here. See previous section on "Network VIP's" for more information. When enabled, Cell Assistant will launch a background task that provides real-time channel data to VIP's running on other computers via a network. Network VIP's are rarely used so the data server is typically disabled.

3.0 Watchdog

The watchdog feature in Cell Assistant provides a method of shutting the system down in the event of a computer failure (hardware or software). To accomplish this, many systems include a small external watchdog timer module which is connected to a serial port. When

Cell Assistant is running properly, the DTR bit (on the configured serial port) is toggled approximately once each second. Each time the watchdog timer module senses the DTR bit transition from low to high, it resets its internal timer. If this internal timer is not reset approximately every two seconds, the timer will expire and cause the output contacts of the module to open. These contacts should be wired up to the system such that the system shuts down when the contacts are open.

If the watchdog circuit is properly installed, the system cannot be run unless the computer is on and Cell Assistant is running; however, in many cases it is still necessary to run without the computer on or without Cell Assistant running. This could be accomplished by using a switch to short the output contacts of the watchdog module, effectively bypassing the watchdog circuit. If this is done, however, there is a danger that a test can be inadvertently run with the watchdog bypassed. This will result in inadequate protection of the equipment. A preferred method is to implement an interlocking contact on the bypass switch (closed when the bypass switch contact is open and open when the bypass switch contact is closed). This interlocking contact should be wired between the RTS and CTS lines of the serial port. Cell Assistant will not allow the system to run a test if the contact from the RTS line to the CTS line of the serial port is open, which indicated that the watchdog is bypassed.

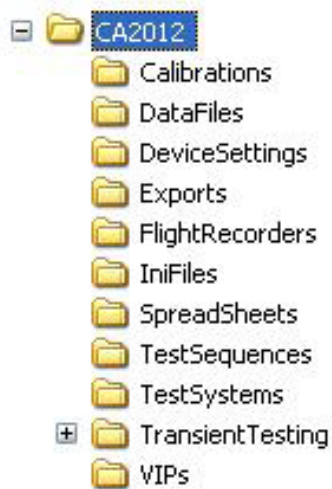
If there is no watchdog timer module installed, the “Serial Port” setting should be set to “Disabled”.

CHAPTER 13

Miscellaneous

1.0 Data Folder Structure

Assuming Cell Assistant was installed using the default installation folder (i.e. C:\CA2012), the following folder structure is created by the installer on the C: drive of the host computer. These sub-folders are created in order to organize all data that is part of a TestSystem. Some data is a static component of the overall TestSystem implementation (e.g. configuration files). Other data (e.g. test data) is generated “on the fly” by running TestPlans (e.g. data logging files).



Folder	Use
Calibrations	Contains the main calibration database file – CalibrationDatabase.db This is also the default folder where all other exported snapshot files of the calibration database are saved (e.g. cal_0001.csv, cal_0002.csv, etc.)
DataFiles	This is the folder where running TestPlans will store all data log files. A running TestPlan can specify any destination folder to save a data log file; however, this is the default folder when no destination folder is specified (which is typical for most TestPlans).
DeviceSettings	Contains several *.INI files containing all device settings. Communications.INI contains all serial and network settings for all configured devices. GeneralSettings.INI contains all other settings for all configured devices. Other device-specific *.INI files may also exist.
Exports	This is the default folder when exporting channels, limits, and data file formats to a *.csv file.
FlightRecorders	All flight recorder *.csv files are stored here. The TestSystem developer cannot change this folder.
IniFiles	Any DSBasic TestPlan can create application specific *.INI files to store various test parameters, test results, etc. These files can reside anywhere on the local computer; however, this folder is provided as a convenient storage location.
SpreadSheets	This folder is provided as a convenient storage location for Microsoft Excel spreadsheet files. Several standard Cell Assistant applications provided by DyneSystems rely on the use of spreadsheets. They will be stored in this folder.
TestSequences	The “Test Sequence Manager” (TSM) is a standard Cell Assistant application developed by DyneSystems. The TSM relies on the use of test sequence files (i.e. *.tsf files). They will be stored in this folder.
TestSystems	All TestSystem files (i.e. *.ca2012 files) should be saved here.
TransientTesting	DyneSystems has developed a set of TestPlans to perform EPA 40CFR Part 86 Engine Mapping and Transient Cycle tests. This folder contains many other tools to support these applications as well as provide data storage for the various test results.

Folder	Use
VIPs	<p data-bbox="634 352 805 380">TransientTesting</p> <p data-bbox="634 409 1414 594">Virtual Instrument Panel (VIP) executable files should be stored here. VIP's are created using the latest version of Microsoft Visual Basic and/or Microsoft Visual Studio. All project files are maintained elsewhere - typically in default project folders created by the specified visual development tools. As VIP's are compiled, the final executable files should be placed into this folder.</p> <hr data-bbox="732 632 1414 636"/> <p data-bbox="732 648 1370 768">Note: A "VIP Relocator" tool is available to simplify the copying of all required files as well as to eliminate file copying errors. See the "Virtual Instrument Panel" section for information regarding the VIP Relocator tool.</p> <hr data-bbox="732 789 1414 793"/>

2.0 Backing Up Data

Backing up your TestSystem is a trivial task – simply archive all data in the installation folder (e.g. C:\CA2012) as well as all sub-folders. All Cell Assistant product files are stored elsewhere and can easily be re-installed using the product installer; thus, those files require no backup. As explained previously, the installation folder contains all generated data as well as all static configuration data and licensing files. Backing up everything in this folder will facilitate a full restore of the system later (if needed).

Note: A few configuration data items are stored in the system registry (e.g. passwords, appearance settings, etc.). It is not necessary to backup these items as they typically can be re-established should Cell Assistant require a complete restore/re-installation.

3.0 Passwords

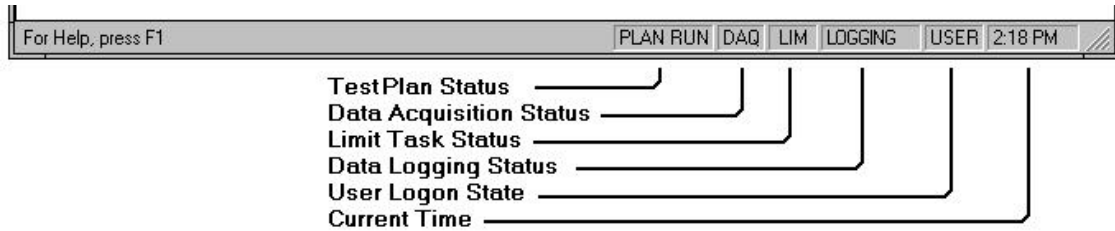
Cell Assistant provides three levels of access to all features and data – two of the levels are password protected.

Access Level	Description
User	This is the most basic access level. This is the default access level when Cell Assistant is initially started. No password is required. This level allows the user to run a test (i.e. select and execute a TestPlan) and to view various screens. But no editing is allowed. The operator cannot edit any DSBASIC scripts, add or modify data channels, change report formats, etc.
Calibration	This level requires a password. The initial default password is 'cal'. This level is similar to the 'User' level but adds access to all calibration features. The operator can run TestPlans but with the additional capability to calibration data channels as often as needed.
Configuration	This is the highest access level. The initial default password is 'cfg'. This level provides access to all features of Cell Assistant and is mainly intended for used by TestSystem developers.

To change the operator access level, select 'File' then 'Password ...' from the main menu (or click the corresponding toolbar icon). The Password dialog then appears allowing the operator to enter passwords (to obtain a higher access level), logout (to lower your access level), and to change the password values.

4.0 Status Bar

A status bar is located at the bottom of the main Cell Assistant window. An example is shown below.



Indicator	Description
Test Plan Status	This indicator displays PLAN RUN whenever a TestPlan is running. If a TestPlan has been terminated by a limit exception handler, the indicator displays ABORTED . It is blank at all other times.
Data Acquisition Status	This indicator displays DAQ when data acquisition is running; otherwise, it is blank.
Limit Task Status	This indicator displays LIM when the limit tasks are running (i.e. whenever a TestPlan is running). If a limit exception is detected, it displays TRIP . The indicator will again display LIM when all limits return to normal. It is blank at all other times.
Data Logging Status	This indicator displays LOG OPEN whenever a running TestPlan opens a data file for logging. The indicator changes to LOGGING when data logging commences. It is blank at all other times.
User Logon State	Displays the current access level: USER , CAL , or CFG .

APPENDIX 1

System Channels

The following system function channels are available in Cell Assistant (see Channels chapter).

J607 Correction Factor (Dew Point)	
This function calculates the SAE J607 correction factor for torque and power using dew point as the measure of humidity.	
Where:	
$Correction\ Factor = \frac{101}{B - E} \times \sqrt{\frac{t + 273}{288.6}}$	
$E = 0.61078 \cdot e^{17.2694 \left(\frac{t_{dp}}{t_{dp} + 238.3} \right)}$	
Input	Description
B	Barometric Pressure (kPa)
t_{dp}	Dew Point (°C)
t	Intake Air Temperature (°C)

J1349 Correction Factor (Dew Point)	
This function calculates the SAE J1349 correction factor for torque and power using dew point as the measure of humidity.	
Where:	
$Correction\ Factor = 1.18 \times \left(\frac{99}{B - E} \right) \times \sqrt{\frac{t + 273}{298}} - 0.18$	
$E = 0.61078 \cdot e^{17.2694 \left(\frac{t_{dp}}{t_{dp} + 238.3} \right)}$	
Input	Description
B	Barometric Pressure (kPa)
t_{dp}	Dew Point (°C)
t	Intake Air Temperature (°C)

J1349 Correction Factor (Relative Humidity)	
This function calculates the SAE J1349 correction factor for torque and power using relative humidity as the measure of humidity.	
$\text{Correction Factor} = 1.18 \times \left(\frac{99}{B - E} \right) \times \sqrt{\frac{t + 273}{298}} - 0.18$ $E = \frac{\% RH}{100} \times 0.61078 \cdot e^{17.2694 \left(\frac{t_a}{t_a + 238.3} \right)}$	
Input	Description
B	Barometric Pressure (kPa)
RH	Relative Humidity (%)
t_a	Ambient Air Temperature (°C)
t	Intake Air Temperature (°C)

ISO 3046-1 Correction Factor	
This function calculates the ISO 3046-1 correction factor for torque and power.	
Where:	
$\text{Correction Factor} = \left(\frac{99}{P_b} \right)^{1.2} \times \left(\frac{t + 273}{298} \right)^{0.6}$ $P_b = B - RH * [T_1 + (T_2 * T_h) + (T_3 * T_h^2) + (T_4 * T_h^3) + (T_5 * T_h^4)]$ $T_1 = 0.620854$ $T_2 = 0.0390123$ $T_3 = 0.00188944$ $T_4 = 0.00000663369$ $T_5 = 0.000000657956$	
Input	Description
B	Barometric Pressure (kPa)
RH	Relative Humidity (%)
t	Intake Air Temperature (°C)
T_h	Temp of Relative Humidity Sample (°C)

°F to °C	
This function converts a temperature in degrees Fahrenheit to a temperature in degrees Celsius.	
Temperature (°C) = $(T_f - 32) / 1.8$	
Input	Description
T_f	Temperature (°F)

°C to °F	
This function converts a temperature in degrees Celsius to a temperature in degrees Fahrenheit.	
Temperature (°F) = $(T_c * 1.8) + 32$	
Input	Description
T_c	Temperature (°C)

Pounds to Kilograms	
This function converts a weight in pounds (lb) to a mass in kilograms (kg).	
Mass = $W * 0.45359237$	
Input	Description
W	Weight (lb)

Kilograms to Pounds	
This function converts a mass in kilograms (kg) to a weight in pounds (lb).	
Weight = $M * 2.2046226$	
Input	Description
M	Mass (kg)

PSI to Inches of Mercury	
This function converts a pressure in pounds per square inch (PSI) to a pressure in inches of mercury (in. Hg) at 0° C.	
Inches Hg = $P * 2.03602$	
Input	Description
P	Pressure (psi)

Inches of Mercury to PSI	
This function converts a pressure in inches of mercury (in. Hg) at 0° C to a pressure in pounds per square inch (PSI).	
$PSI = INHG * 0.0491154311$	
Input	Description
INHG	Pressure (in Hg)

Inches of Mercury to kiloPascals	
This function converts a pressure in inches of mercury (in. Hg) at 0° C to a pressure in kiloPascals (kPa).	
$KPA = INHG * 3.38638962$	
Input	Description
INHG	Pressure (in Hg)

LB-FT to Newton Meters	
This function converts a torque in pound feet (lb ft) to a torque in Newton meters (Nm).	
$NM = T * 1.355817947$	
Input	Description
T	Torque (lb-ft)

Newton Meters to LB-FT	
This function converts a torque in Newton meters (Nm) to a torque in pound feet (lb ft).	
$LB-FT = T * 0.73756215$	
Input	Description
T	Torque (Nm)

Horsepower to Kilowatts	
This function converts power in horsepower (HP) to power in kilowatts (kW).	
$KW = HP * 0.7457$	
Input	Description
HP	Power (HP)

Kilowatts to Horsepower	
This function converts power in kilowatts (kW) to power in horsepower (HP).	
$HP = KW / 0.7457$	
Input	Description
KW	Power (kW)

Brake Specific Fuel Consumption	
This function calculates Brake Specific Fuel Consumption (BSFC) from fuel consumption and power. The units depend on the units used for power and fuel consumption.	
$BSFC = F/P$	
Input	Description
F	Fuel consumption
P	Power

Horsepower	
This function calculates horsepower (HP) from torque and speed.	
$HP = (S \times T) / 5252$	
Input	Description
T	Torque (lb-ft)
S	Speed (RPM)

Multiplication (2 channels)	
This function multiplies any two channels together creating a product that is a new channel.	
$PRODUCT = CH1 \times CH2$	
Input	Description
CH1	Channel 1
CH2	Channel 2

