Developing Test Programs Using TestStand™ Participant Guide

Course Software Version 2017 Service Pack 1 (SP1)
2019 Edition
Part Number 328189A-01

Trademarks

Refer to the *NI Trademarks and Logo Guidelines* at ni.com/trademarks for more information on National Instruments trademarks.

ARM, Keil, and µVision are trademarks or registered of ARM Ltd or its subsidiaries.

LEGO, the LEGO logo, WEDO, and MINDSTORMS are trademarks of the LEGO Group.

TETRIX by Pitsco is a trademark of Pitsco, Inc.

FIELDBUS FOUNDATION™ and FOUNDATION™ are trademarks of the Fieldbus Foundation.

EtherCAT® is a registered trademark of and licensed by Beckhoff Automation GmbH.

CANopen® is a registered Community Trademark of CAN in Automation e.V.

DeviceNet™ and EtherNet/IP™ are trademarks of ODVA.

Go!, SensorDAQ, and Vernier are registered trademarks of Vernier Software & Technology. Vernier Software & Technology and vernier.com are trademarks or trade dress.

Xilinx is the registered trademark of Xilinx, Inc.

Taptite and Trilobular are registered trademarks of Research Engineering & Manufacturing Inc.

FireWire® is the registered trademark of Apple Inc.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix®, Tek, and Tektronix, Enabling Technology are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

The ExpressCard™ word mark and logos are owned by PCMCIA and any use of such marks by National Instruments is under license.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your media, or the *National Instruments Patent Notice* at ni.com/patents.

Cover Photo

Photo courtesy: NASA

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

Visit ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

For further support information, refer to the *Additional Information and Resources* appendix. To comment on National Instruments documentation, refer to the National Instruments website at ni.com/info and enter the Info Code feedback.

Table of Contents

Student Guide

In this student guide, you will learn about the course description and the items you need to get started in this course.

Topics

+ Course Description
+ Course Goal
+ What You Need to Get Started
+ Installing the Course Software
+ Course Project Exercise Setup
+ Course Learning Map

A. Course Description

The Developing Test Programs Using TestStand course teaches you how to navigate the TestStand software design environment and use different design languages to quickly create test applications.

This course assumes that you are familiar with Windows and that you have experience writing algorithms in the form of flowcharts or block diagrams.

This course is also part of the NI Badge program. The NI Badge program helps you find learning resources and gain skills related to your projects. Track your knowledge growth with milestone badge assessments and professional certifications. Go to *ni.com* to test your understanding of engineering fundamentals and best practices using NI products with these free online assessments.

B. Course Goal

This course prepares you to use existing TestStand features and best practices to create and deploy test sequences as a complete test system.

This course does *not* describe the following:

- Every built-in function, or object; refer to the *TestStand Help* for more information about features not described in this course.
- Developing a complete application for any student in the class; refer to the examples and project templates in the TestStand directory you can use and incorporate into applications you create.

C. What You Need to Get Started

Before you use take this course, make sure you have all of the following items:

* Computer running Windows 7/8/10
* TestStand 2017 SP1 (32-bit)
* LabVIEW 2018 (32-bit) and/or LabWindows/CVI 2017
* *Developing Test Programs Using TestStand* course media, from which you install the following folders:

Directory	Table Head
`Exercises\Developing Test Programs`	Contains files used in the course
`Solutions\Developing Test Programs`	Contains completed course exercises

D. Installing the Course Software

Complete the following steps to install the course software.

1. Insert the course media in your computer. The **Developing Test Programs Using TestStand** dialog box appears.
2. Click **Install the course materials**.
3. Follow the on-screen instructions to complete installation and setup.

File Locations

Exercise files are located in the `C:\Exercises\Developing Test Programs` folder assuming that you installed the files on your root directory.

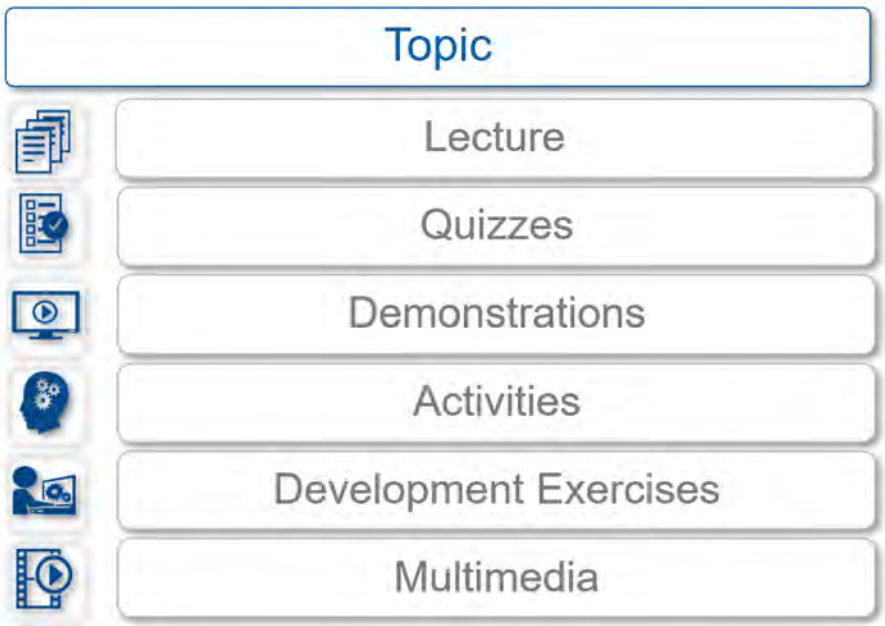
Solution files are located in the `C:\Solutions\Developing Test Programs` folder assuming that you installed the files on your root directory.

Instruction Methods

The Participant Guide is divided into lessons. Each lesson contains the following:

- An introduction with the lesson objective and a list of topics.
- Slide images with additional descriptions of topics, activities, demonstrations, and multimedia segments.
- Exercises to reinforce topics. Some lessons include optional and challenge exercises.
- A lesson review that tests and reinforces important concepts and skills taught in the lesson.

Note For Participant Guide updates and corrections, refer to `ni.com/info` and enter the Info Code `rdtce1`.



E. Course Project Exercise Setup

All the exercises in this course project depend on completion of the previous exercise. If you have not completed any of the exercises, please complete the following steps.

- Before you begin an exercise, copy the solution files for the previous exercise from the Solutions folder to the appropriate locations in Exercises directory (LabVIEW, CVI, or TestStand Only).
- Copy the Setup folder from the Solutions folder to the Exercises directory. Run the `setup` sequence file located in the `C:\Exercises\Developing Test Programs\Setup` directory to set up the search directories.
- To install the drivers, make sure that LabVIEW is not running and then navigate to `C:\Exercises\Developing Test Programs\Driver Simulation API` directory and run `Setup.bat` as administrator. The Setup.bat executable installs the necessary components in the LabVIEW and LabWindows/CVI directories.

Activity: Partner Interview

Find a partner and ask them the following questions. Fill in their answers below.

1. Have you used TestStand before? If so, what kind of projects have you worked on?

2. What languages, environments, and frameworks have you used? Circle all that apply.

- C
- LabVIEW
- LabWindows/CVI
- .NET
- Other (please specify)

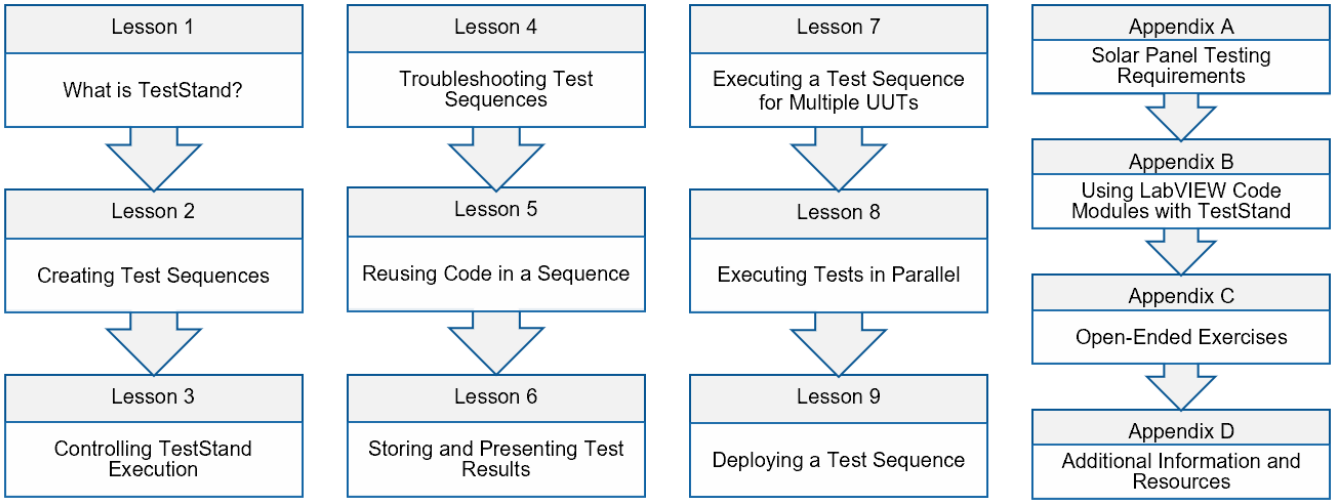
3. Have you used a test executive system like TestStand?

4. What projects are you currently working on?

5. What kind of projects do you hope to use TestStand for?

6. What else do you hope to get out of this training?

F. Course Learning Map



1 What is TestStand?

Describe the purpose of TestStand and the benefits of using test management software.

Topics

+ Introduction to TestStand
+ Benefits of Using TestStand

A. Introduction to TestStand

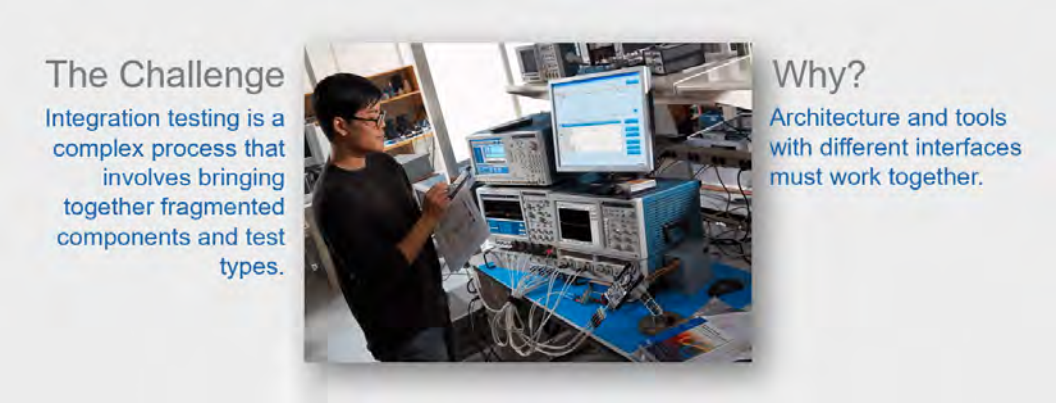
Objective: Explain the purpose of test management software.

Exploring Test System Terminology

Automated Test System A system that can reduce or eliminate the human effort needed to test a product by performing some or all tasks without human interaction.

Test Management Software Software that provides the infrastructure and the modular test framework required to repeat the operations for each product.

"I need to test compliance quickly… but"



Testing Is Critical for Cell Base Stations

What is a Test Executive?

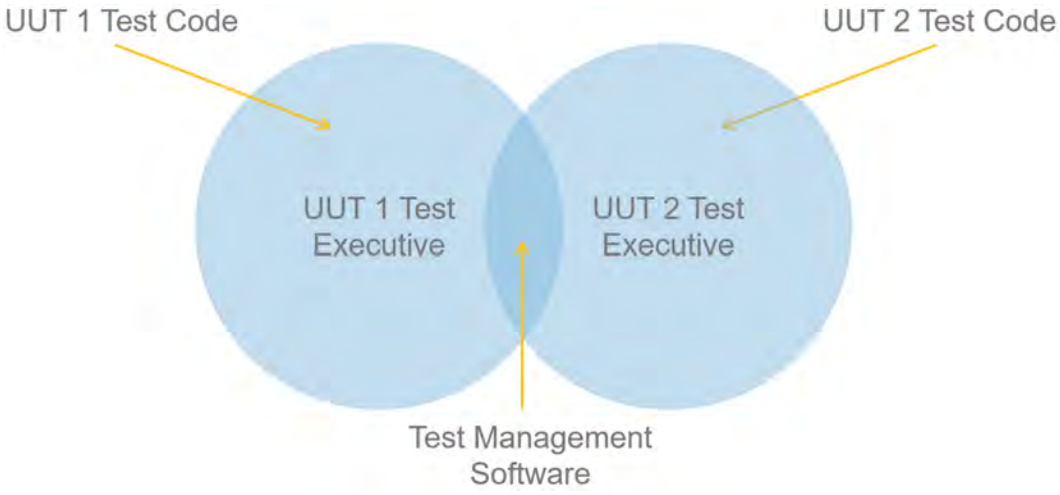
Test Executive A system that organizes and executes sequences of reusable code modules you can create in a variety of programming environments.

Activity: Test Executive Requirements Brainstorm

What needs should a test executive satisfy? Fill in your answers below.

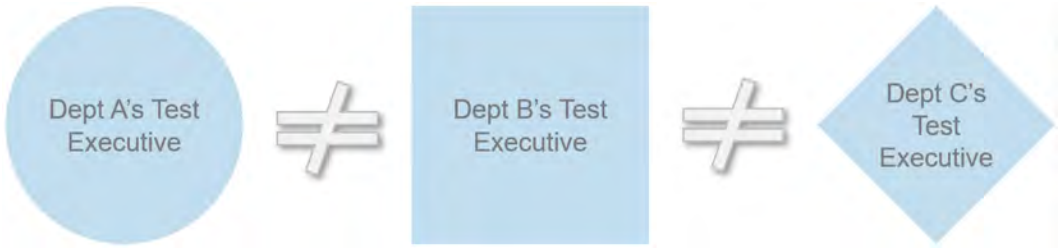
Common Test Executive Needs

Typical Automated Test System Operations



Why standardize on a Common Architecture?

Imagine that you are a test engineer at a major company that designs/produces a variety of products. You are responsible for testing solar panels and have developed your own test executive to test your solar panels on the production floor. Now imagine that an engineer in another group has created their own test executive to test their UUTs.

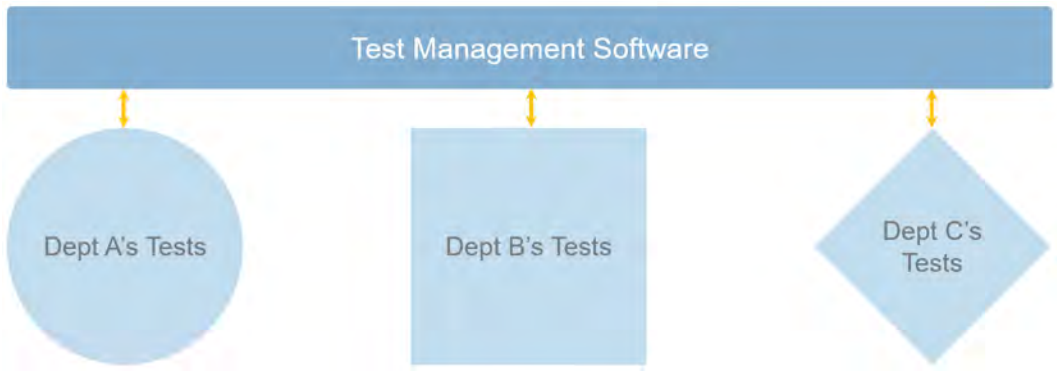


What problems could arise from having two independently-created test executives running on the production floor?

- Little code reuse or consistency
- Each department pays ongoing maintenance costs
- Developers and operators must learn different applications
- Often support only one language or development environment
- Do not scale for future needs.

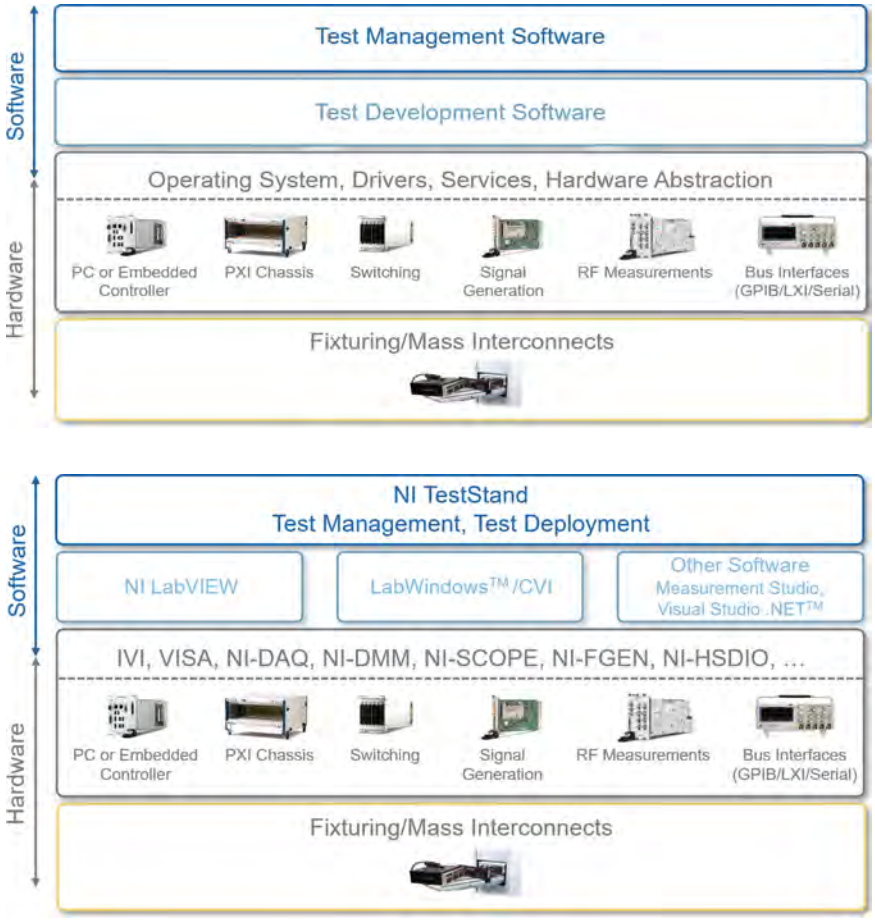
Automated Test System Operations

Using one application to handle the common functionality of the test system is an efficient way to implement your automated test system.



Components of an Automated Test System

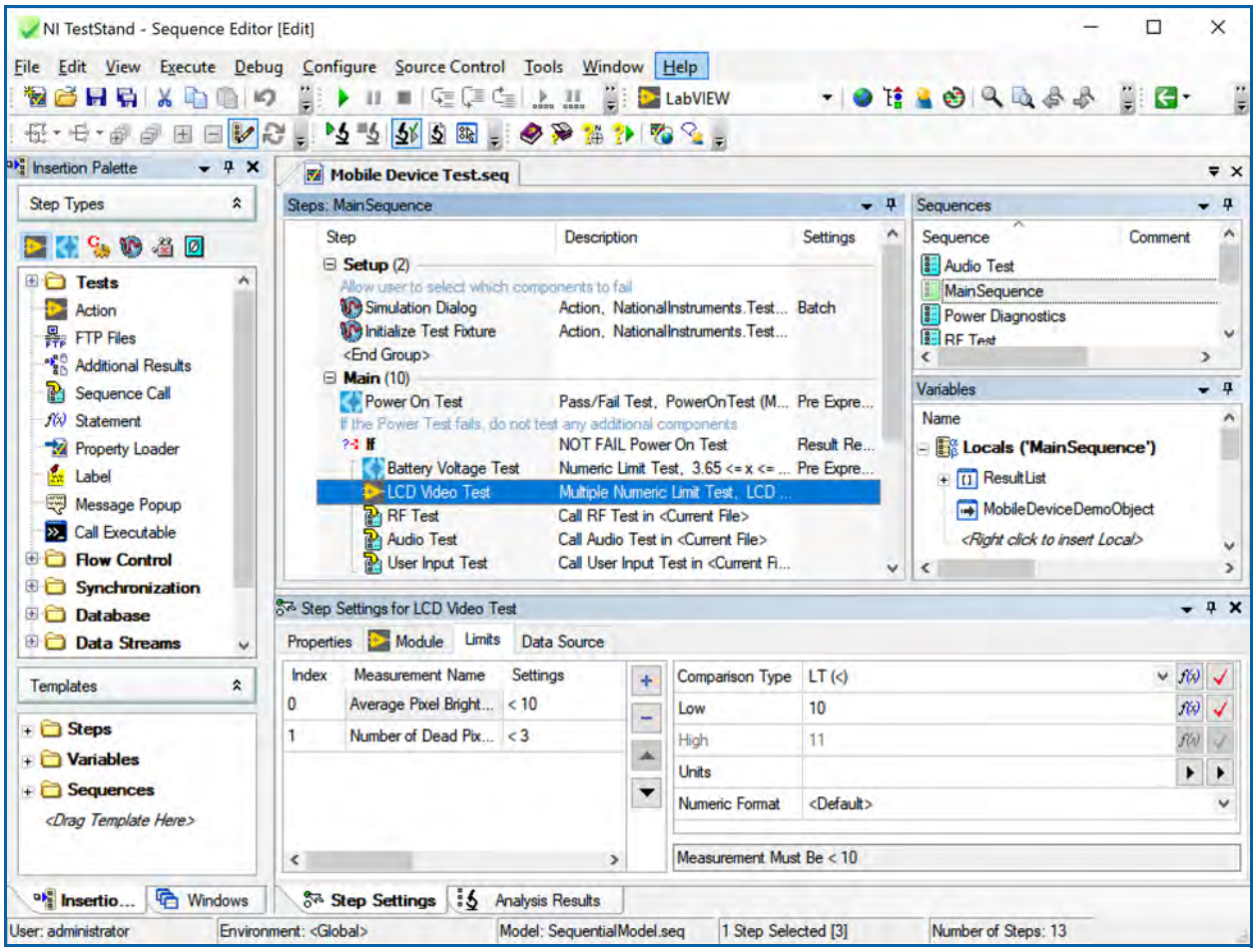
Modularity in test system design helps you increase the maintainability and reusability of your test system components. A modular test software architecture is divided into four layers.



Fixturing/Mass Interconnects	At the lowest level, the first component is well-designed mass interconnects or fixturing. Test operators must quickly and reliably connect and disconnect each device as they come off the assembly line.
OS, Drivers, Services	The OS/Drivers layer interfaces with your device under test (DUT). Layer components include your computer or embedded PXI controller, the PXI chassis, and any bench-top instruments that you have in your system.
Test Development Software	Tests are developed using a programming language. When developing software to control your tests, focus on modularity. Each test should be a piece of code that you can reuse and maintain independently.
Test Management Software	The top layer is the test management software that integrates the different pieces of a test system, controls the use of instrumentation resources, logs results, and integrates with other organization-wide systems.

Demonstration: Launch the TestStand Sequence Editor

Explore the TestStand development environment, known as TestStand Sequence Editor.



Activity: Interface Exploration Race

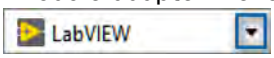
Be the first to find the answers to the following questions. Raise your hand for the trainer to check your answers. If any answers are incorrect, the trainer hands the page back for you to try again. All answers must be correct to place in the race. Good luck!

Instructions:

1. Type `%Teststandpublic%` into a Windows explorer window to open the TestStand public directory.
2. Navigate to `<TestStand Public>\Examples\Demos\Mobile Device Test`.
3. Double-click `Mobile Device Test.seq` to open the example sequence.
4. Click **OK** to close the Login dialog box.
5. Answer the questions below quickly and accurately.
   a. What is the first test under the Main category in the **Steps: Main Sequence** pane?

   b. Click on the answer you got for question a. Look at the **Step Settings for Power On Test** area on the bottom of the screen. What is the third category on the Properties tab?

   c. Click the following code module adapter menu in the toolbar:

   What is the fourth option from the top of the list?

   d. Select **Edit»Sequence File Properties** from the menu. On the Advanced tab, how many **Model Options** are there?

   e. In the **View** menu, what is the keyboard shortcut for **Types**?

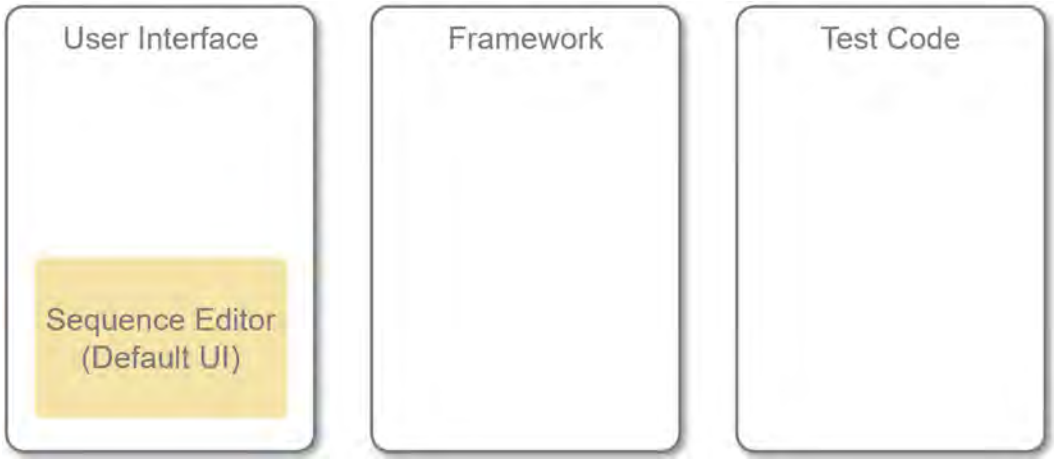
   f. In the **Configure»Result Processing** menu, what is the first Output Name listed?

   g. What does pressing <Ctrl-F1> do?

   h. Keep the dialog box from step g open and select the Index tab. Type in the word `sequence`. How many topics appear when you double click **sequence**?

TestStand Execution Architecture

As we go through the course we will add more detail to this basic execution architecture for TestStand.



B. Benefits of Using TestStand

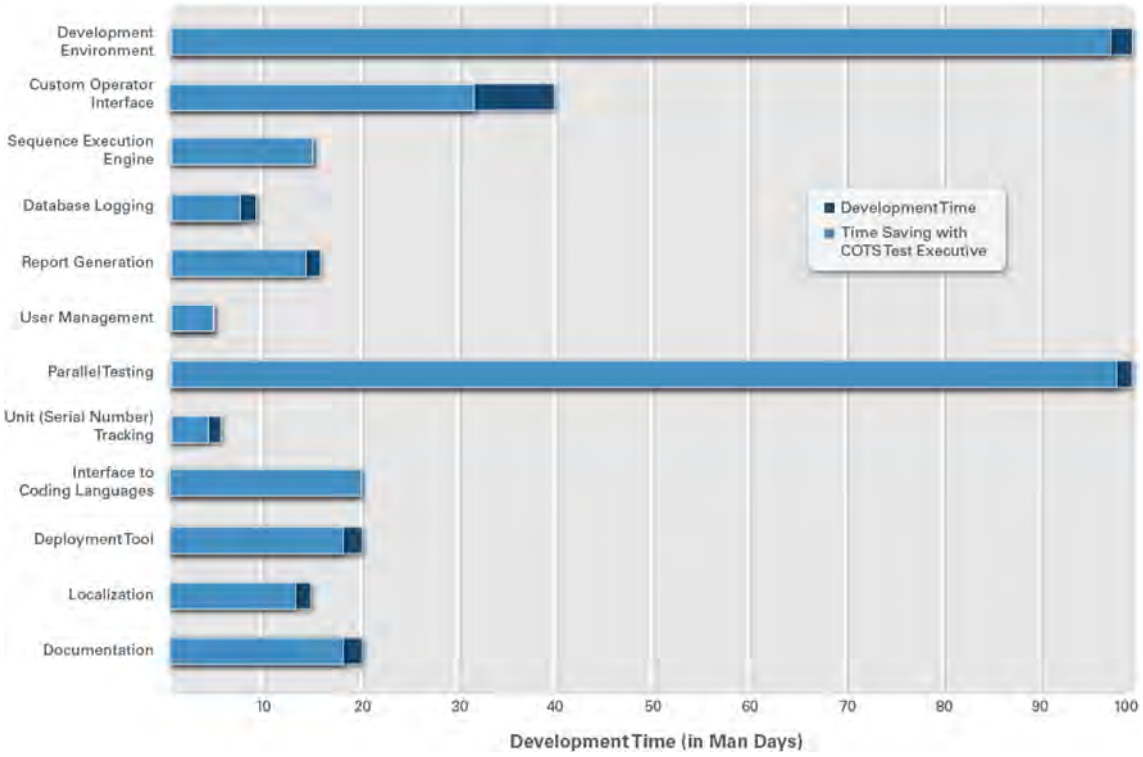
Objective: Describe the benefits of using TestStand.

Off-the-Shelf Test Management Environment

Streamlines Automated Test System Development

Some of the ways TestStand streamlines test system development include the following features.

- Increased reusability of test code because of a modularizaton of common tasks.
- Faster development of UUT-specific code because you can use application environments you are already familiar with such as LabVIEW and LabWindows/CVI.
- Decreased maintenance effort because test executive is standardized.



2 Creating Test Sequences

Create a new test sequence to implement a basic test.

Topics

+ Developing Test Code
+ Creating a New Test Sequence
+ Adding Steps to a Test Sequence
+ Creating and Calling Code Modules
+ Creating Test Steps
+ Executing a Test Sequence

Exercises

Exercise 2-1 Create a Test Sequence
Exercise 2-2 (LabVIEW) Call a Code Module
Exercise 2-2 (LabWindows/CVI) Call a Code Module
Exercise 2-2 (TestStand Only) Call a Code Module

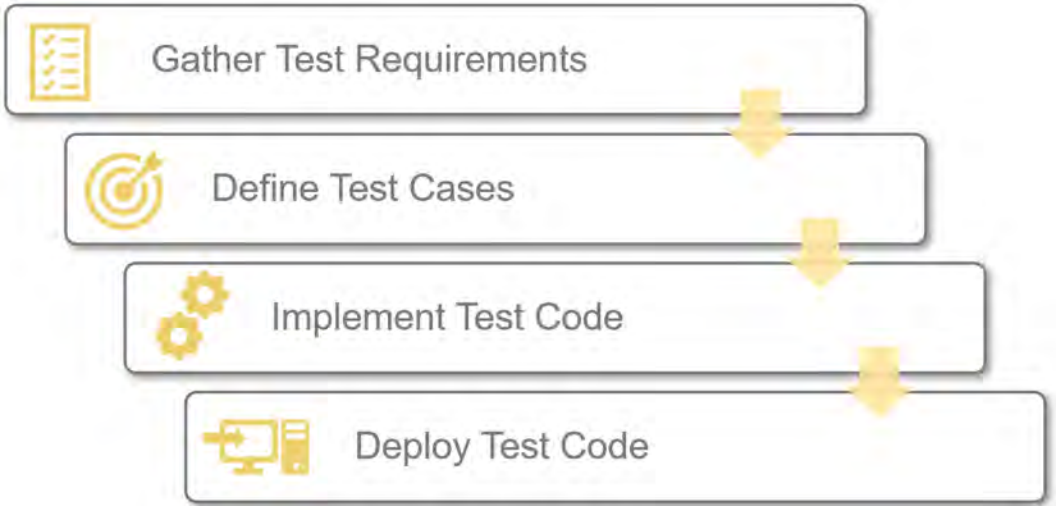
A. Developing Test Code

Objective: Describe an approach for developing test code.

Test Code Development Process

As a test code developer, your code must test individual aspects or components of a device. Your code combines and sequences these individual elements into a cohesive test program.

Test developers need enough understanding of the UUT to understand test requirements and define appropriate test cases. They must also understand the test framework well enough to write and deploy test code that uses that framework.



Multimedia: Solar Panel Tester Overview

In this course, you build a test system for testing a solar panel controller.

Complete the multimedia module, *Solar Panel Tester Overview*, available in the `C:\Exercises\Developing Test Programs\Multimedia` folder to learn more about this scenario.

Activity: Gathering Test Requirements

Use the information from the analysis of the UUT requirements to define a list of all the aspects of the UUT that you must test to verify the UUT.

For this activity, list only high-level requirements such as hardware, software, and types of tests.

B. Creating a New Test Sequence

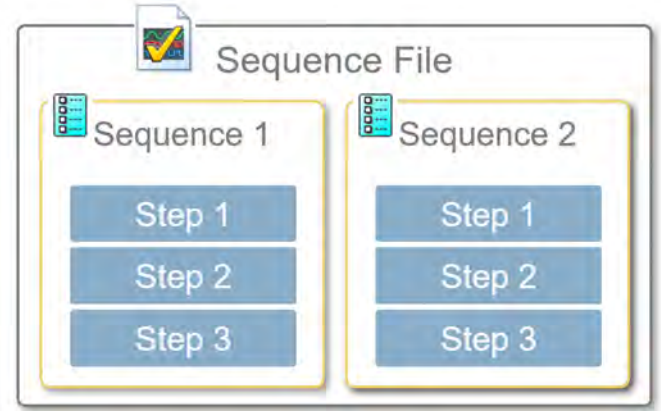
Objective: Describe the structure of a test sequence.

Exploring Test Sequence Terminology

Step A step is the fundamental building block of a TestStand test program. Each step represents one or more operations or tests.

Sequences A sequence is a series of steps that performs a test. To run a test, or test program, you execute a series of sequences.

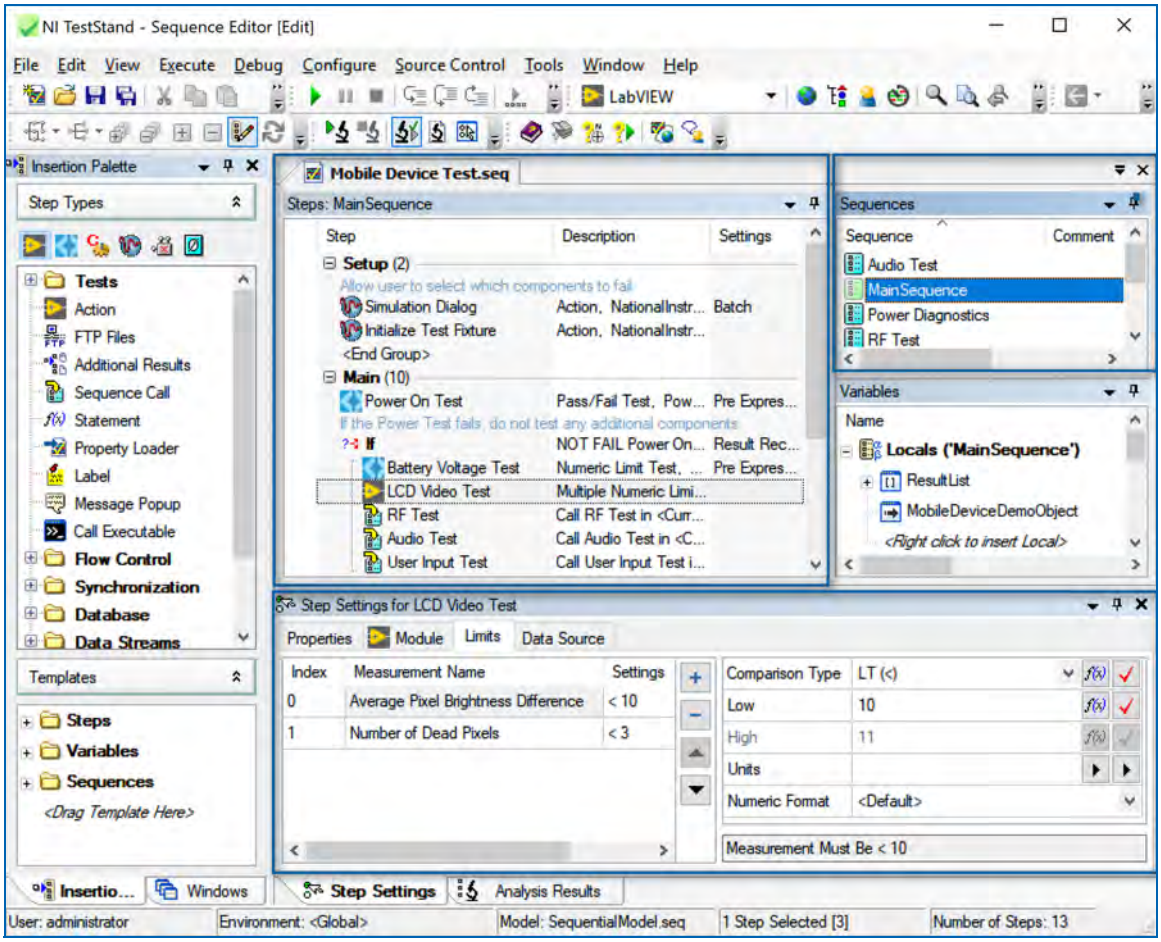
Sequence File A sequence file is a file that contains one or more sequences. All sequences must be in a sequence file.



Demonstration: Components of a Sequence File

Open the Mobile Device Test sequence file to review the components of a typical sequence file. The file is located in the following folder:

`<TestStand Public>\Examples\Demos\Mobile Device Test\Mobile Device Test.seq`



C. Adding Steps to a Test Sequence

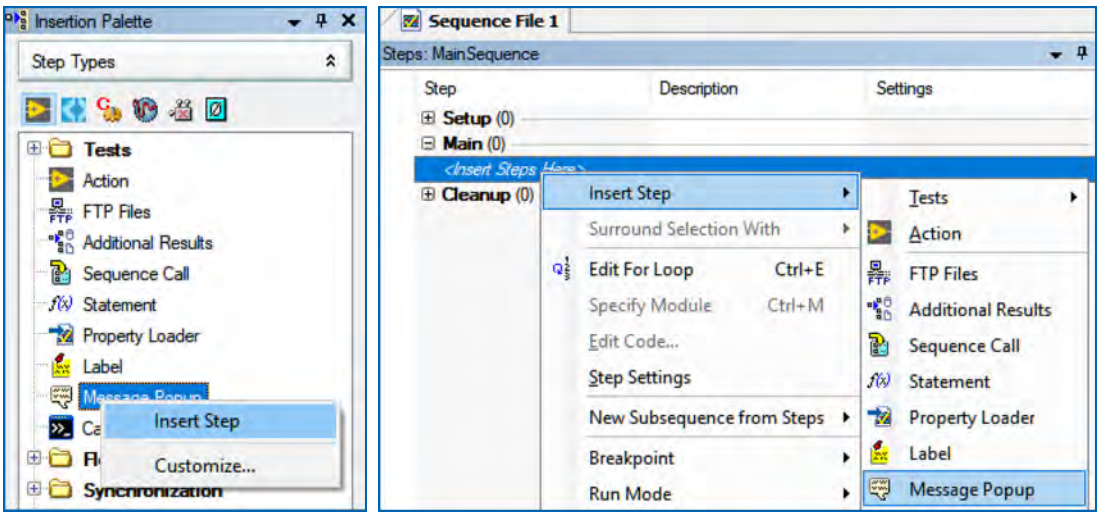
Objective: Add message popup steps and label steps to a test sequence.

Add a Step to a Sequence

When you add a step, you choose a step type from the Step Types palette. The step type defines the behavior and configuration options of the step. TestStand includes a variety of built-in step types.

You can add a step in various ways.

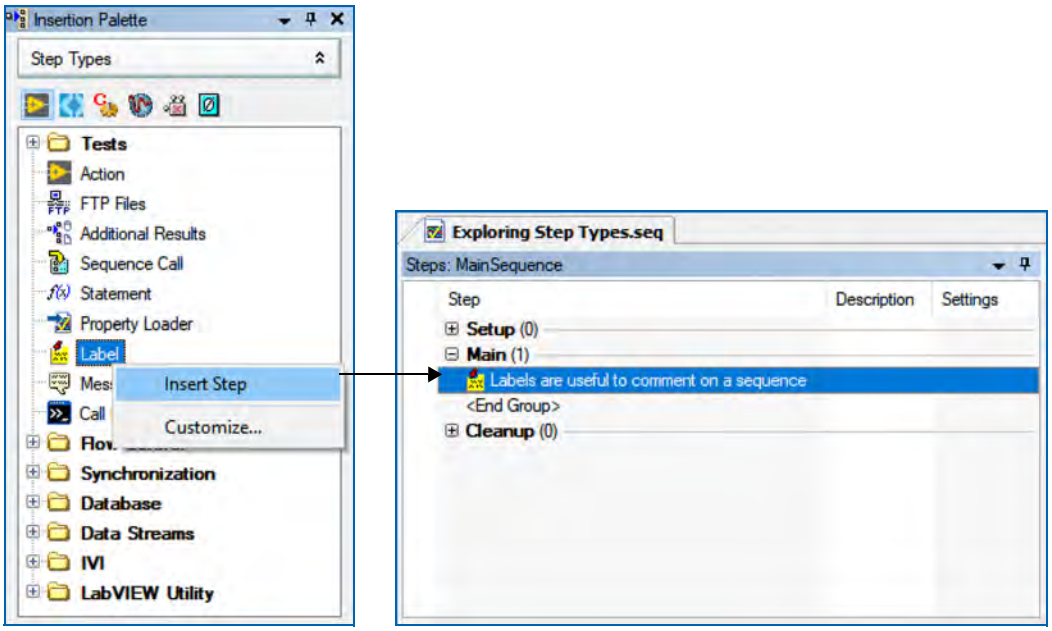
- In the Insertion palette, double-click the type of step you want to add.
- In the Insertion palette, right-click on the step type and select **Insert Step** from the menu.
- Right-click in the Steps pane and select **Insert Step** from the menu.



Exploring Step Types - Label

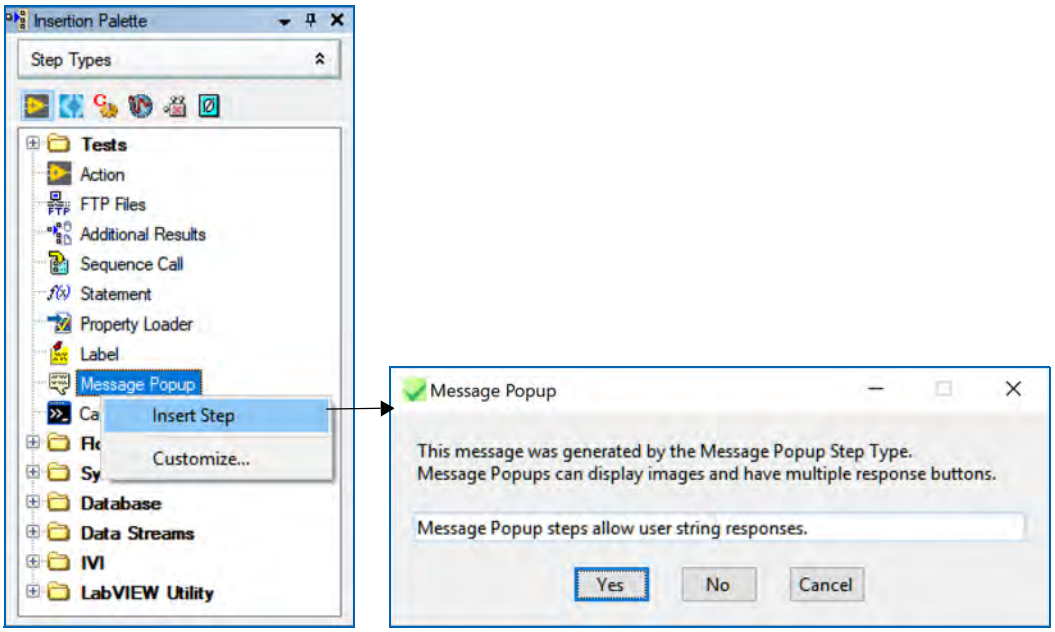
Use Label step types to add comments to sequences and to create reminders or placeholders for other steps.

Label steps do not pass or fail. After a Label step executes, the TestStand Engine sets the step status to Done or Error.



Exploring Step Types - Message Popup

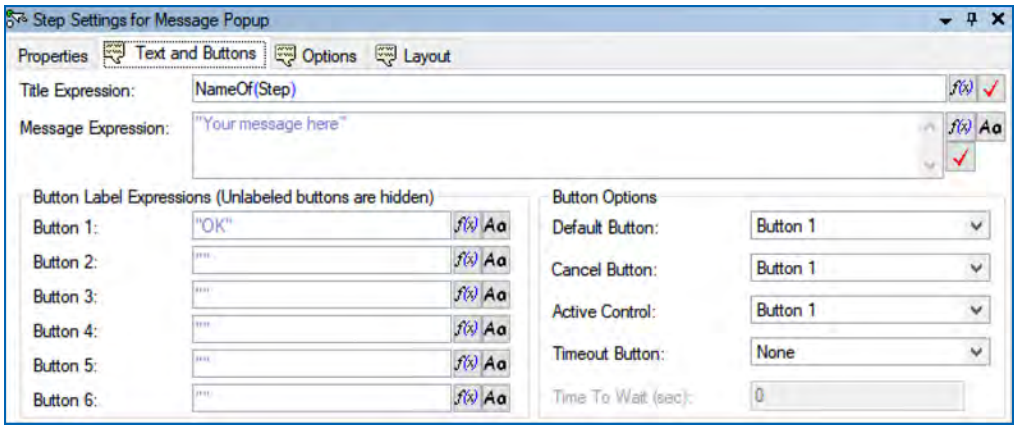
Use the Message Popup step type to display a message to the user or obtain feedback.



Configure a Step

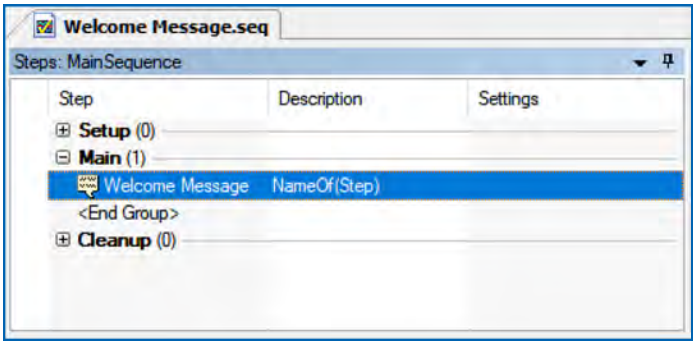
Use the Sequence Editor Step Settings pane to edit the settings of the selected steps.

- Modify the values of built-in properties.
- Edit the step type-specific properties.
- Specify the module call that a step performs.



Demonstration: Create a Message Popup Step

In this demonstration, you create a new sequence file and modify MainSequence to contain a Message Popup step.



Activity: Define Test Cases

Use the information from the *Test Requirements* section in Appendix A of the participant guide to define a list of test cases for the UUT.

Exercise 2-1: Create a Test Sequence

Goal

Create a new sequence file to test a solar panel and controller.

Scenario

The first step in testing a solar panel and controller is to visually inspect the UUT for defects.

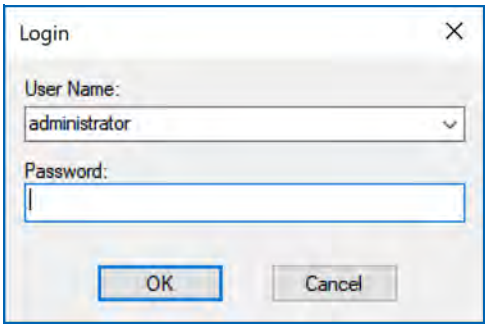
In this exercise, create a test sequence that begins by displaying a message box to prompt the operator to inspect the UUT.

Requirements

The solar panel and controller shall have no visible defects as determined by a test operator.

Implementation

1. Launch the TestStand Sequence Editor and click **OK** to dismiss the Login dialog box.



Note If Compatibility dialog box appears enable the **Don't Show this Dialog Box Again** option and click **OK** to dismiss it.

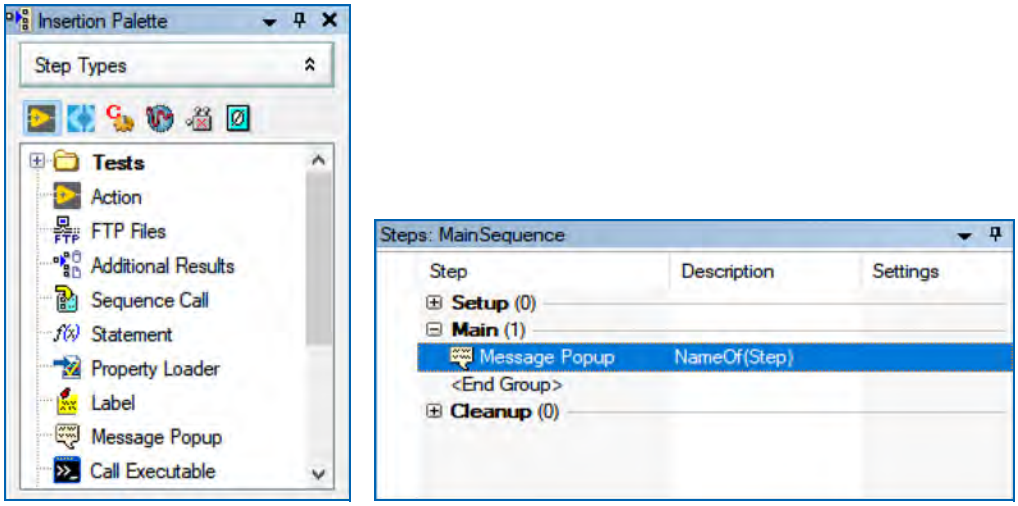
2. Select **File»Save Sequence File** and save the sequence file as `Solar Panel Test.seq` in the:

   a. `C:\Exercises\Developing Test Programs\LabVIEW` directory if you will perform exercises for the LabVIEW version.

   b. `C:\Exercises\Developing Test Programs\CVI` directory if you will perform exercises for the CVI version.

   c. `C:\Exercises\Developing Test Programs\TestStand Only` directory if you will perform exercises for the TestStand Only version.

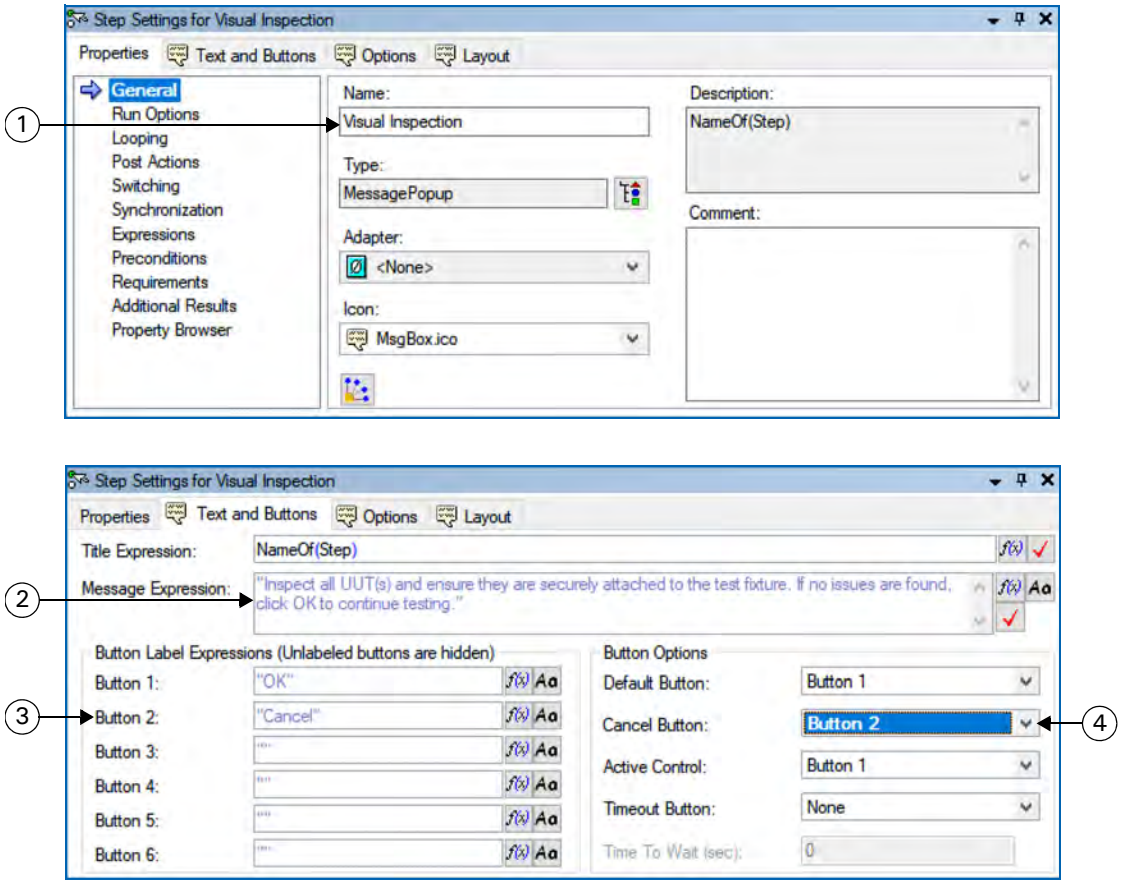
3. Double-click **Message Popup** in the Insertion Palette to add a new message popup step to the Main step group, as shown below.



1 Select **View»Insertion Palette** to display the palette, if necessary.

4. Configure the Message Popup step in the Step Settings pane.

Tip Select **View»Step Settings** to display the Step Settings pane, if necessary.

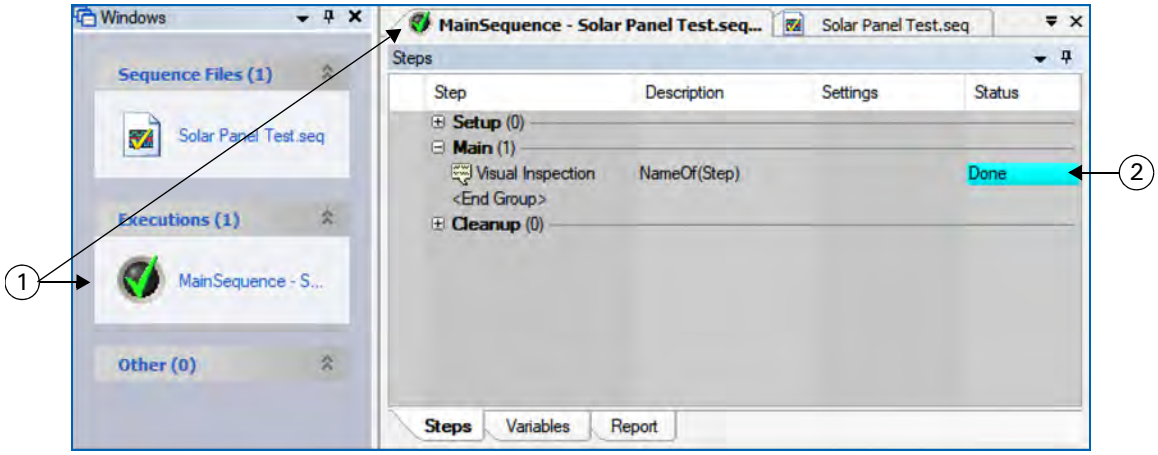


1 Rename the step `Visual Inspection`.
2 Enter the following into the **Message Expression** field of the **Text and Buttons** tab:
`"Inspect all UUT(s) and ensure they are securely attached to the test fixture. If no issues are found, click OK to continue testing."`
3 In the **Button 2** field, enter `"Cancel"`.
4 Select **Button 2** from the **Cancel Button** drop-down menu. The Cancel Button refers to the red X in the upper right-side of the dialog box.

5. Save the sequence file.

Test

1. Select **Execute»Run MainSequence** to execute the sequence.
2. Confirm that the message popup contains the following items.
   - Correct message text
   - **OK** and **Cancel** buttons.
3. Click either the **OK** or **Cancel** button to dismiss the dialog box.
4. Verify that the execution passed.



1	The list bar control and the **MainSequence** tab display a passing icon with a green checkmark.
2	The Status column displays **Done** after the execution has finished.

5. Press <Ctrl-D> to close the completed execution window.

End of Exercise 2-1

D. Creating and Calling Code Modules

Objective: Explain the role of a code module and identify the steps necessary to create and call a code module in a test sequence.

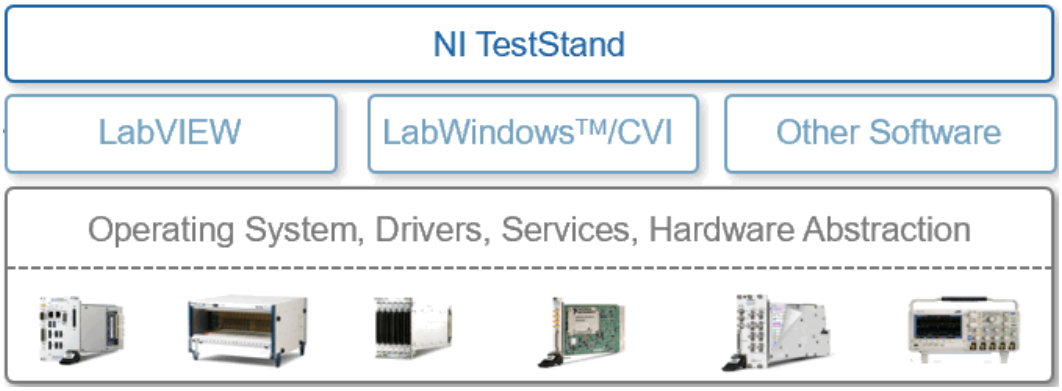
What is a Code Module?

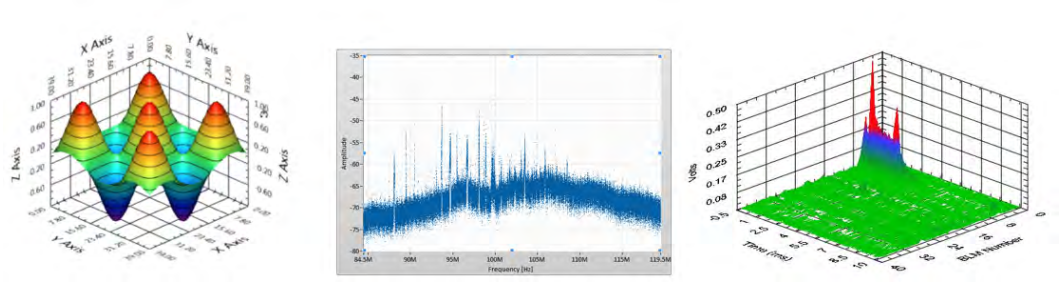
Code Module External code called from a TestStand step to perform a specific test or other action.

Module Adapters Interface used by TestStand steps to call a specific type of code module.

When to Use a Code Module?

Create a code module from a supported application development environment to interface with the hardware driver and acquire data.





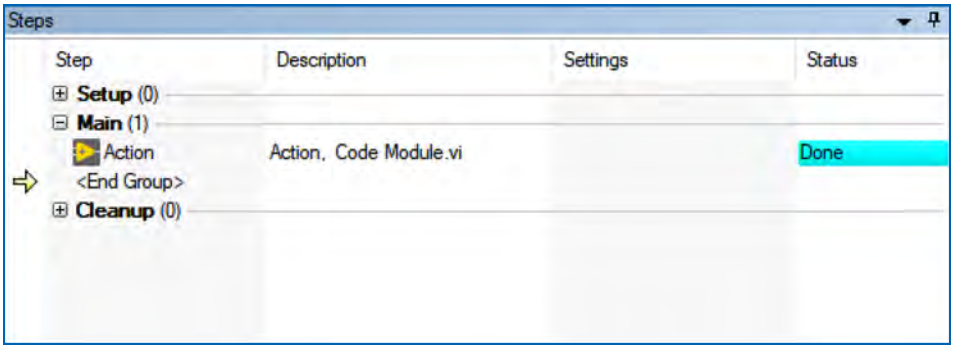
Programming Tasks for Code Modules	Programming Tasks for TestStand
Setting up stimuli	Evaluating test measurements against limits
Making measurements or serving as an interface to hardware	Sequencing or determining the execution order of tests or subtests
Performing routing tasks	Reporting and logging results
Manipulating complex arrays	
Iterating over large amounts of data or performing large numbers of calculations	

Calling a Code Module

1. Create a new action or test step.
2. Select a module adapter.
3. Create the code module.
4. Configure step to call the code module.

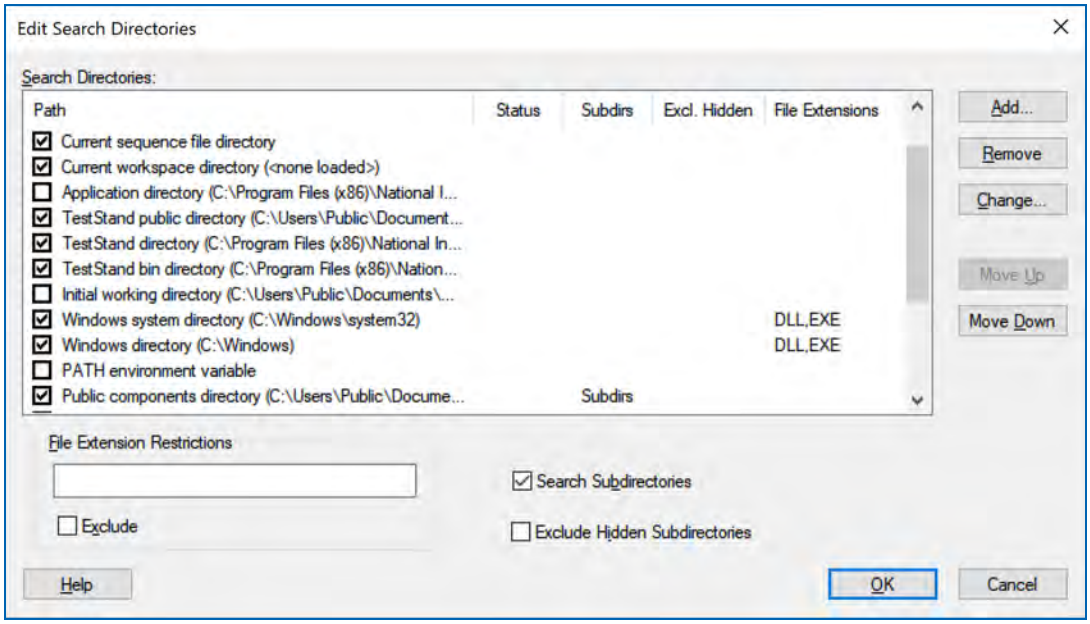
Demonstration: Create and Configure an Action Step

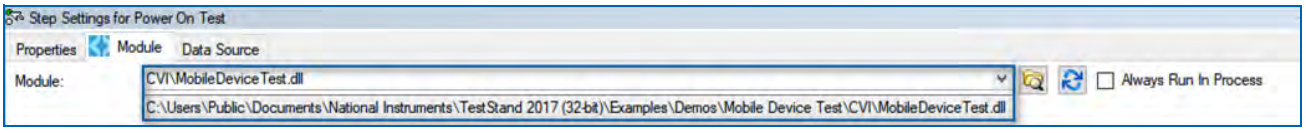
In this demonstration, create an action step that calls a code module.



Specify the Code Module File Path

TestStand maintains a list of search directories to search when you use a relative path to specify a file or code module. To modify the Search Directories, select **Configure»Search Directories**.



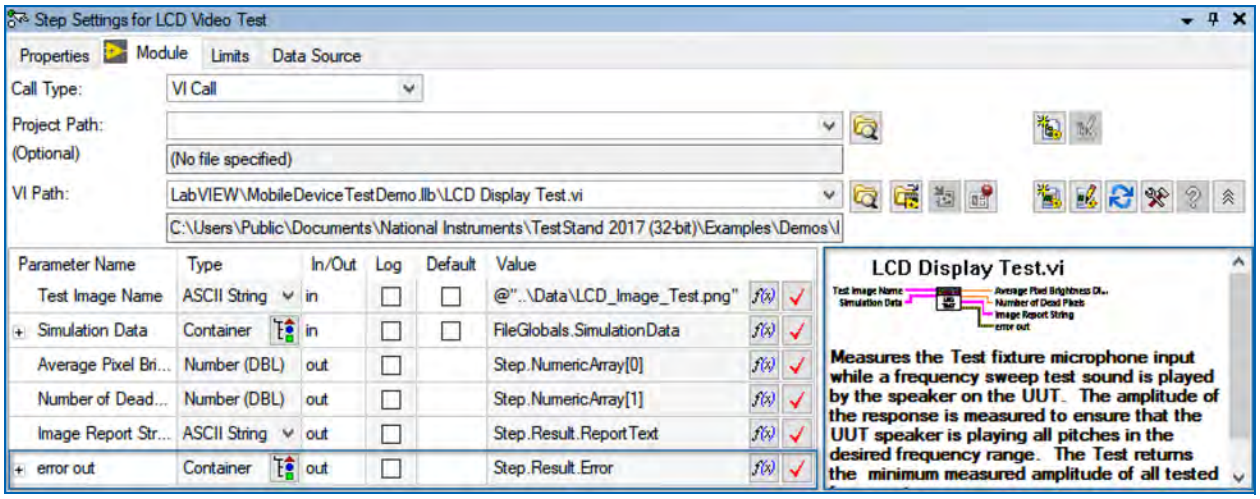


Passing Data to and from the Code Module

Parameters The inputs and outputs of a code module.

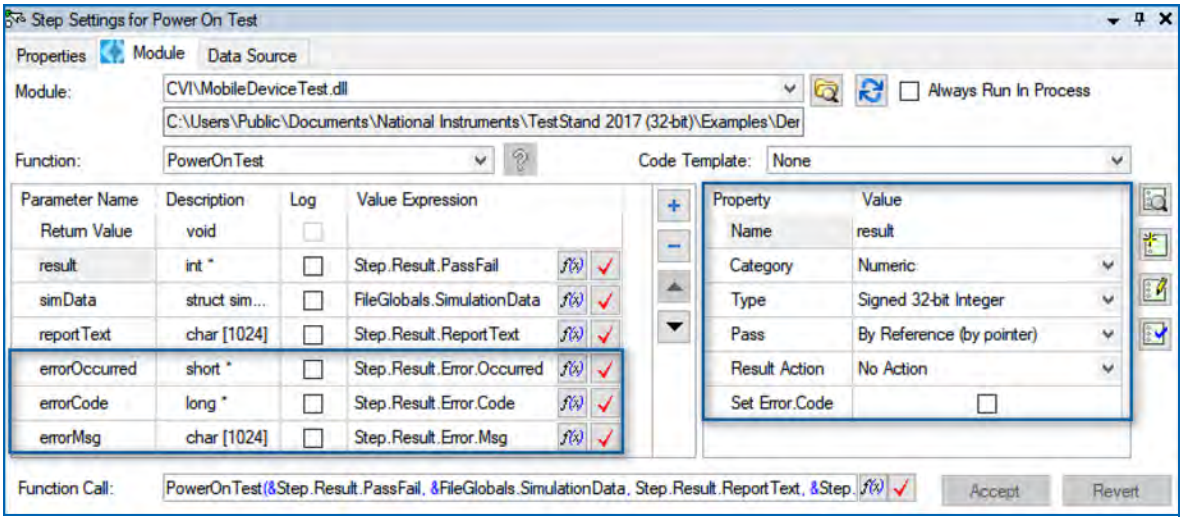
Configure LabVIEW Code Module Parameters

TestStand automatically loads the VI icon, connector pane information, and context help to show you exactly what parameters the VI expects and what values the code module will pass back to TestStand.



Configure LabWindows/CVI Code Module Parameters

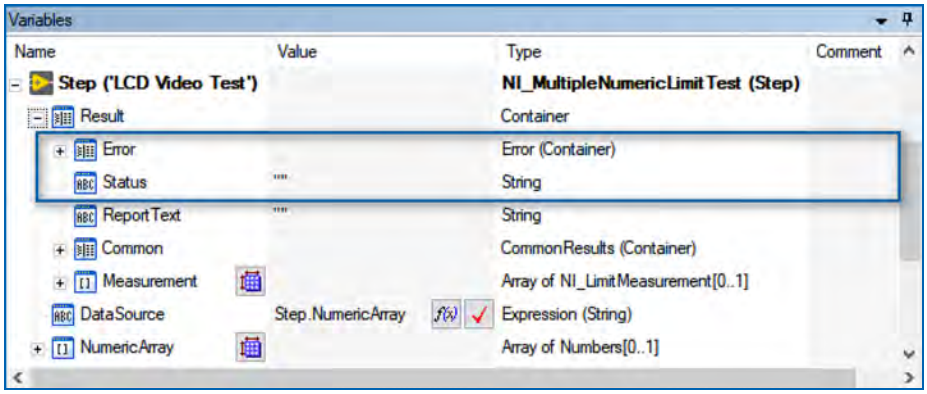
Parameters represent arguments in the function prototype. **Pass by value** parameters provide input to the code module. **Pass by reference** parameters provide input from and return data to the same location. **Function return** values provide output from the code module.



Where Does TestStand Store Code Module Outputs?

All steps have built-in properties which can be used to store results.

- Step.Result.Status
- Step.Result.Error



To pass error information out of the code module and store it in the step error property you must do the following:

- In LabVIEW, wire an Error Out cluster to the connector pane of the code module.
- In LabWindows/CVI, use output parameters with the following guidelines:
  - A short integer for whether or not an error occurred
  - A long integer for the error code
  - A character array for the error message

E. Creating Test Steps

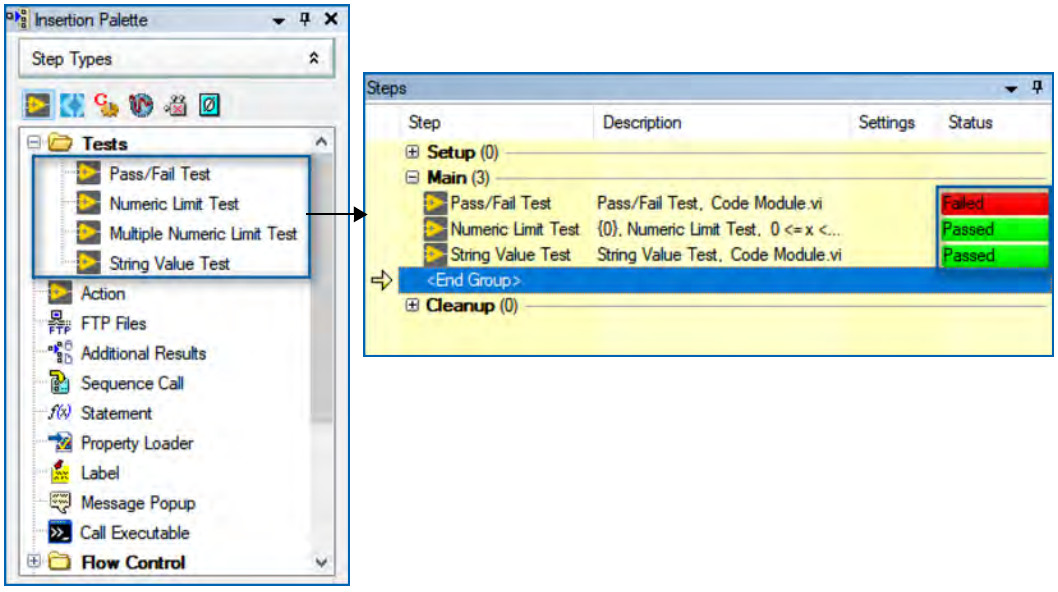
Objective: Construct a step in a test sequence to call code modules given test requirements.

Create a Test Step

TestStand comes with the following step types for performing boolean, numeric, and string tests.

- Pass/Fail Test
- Numeric Limit Test
- String Value Test

Test steps generally call a code module and return a **Passed** or **Failed** status value.



Choose a Test Step Type

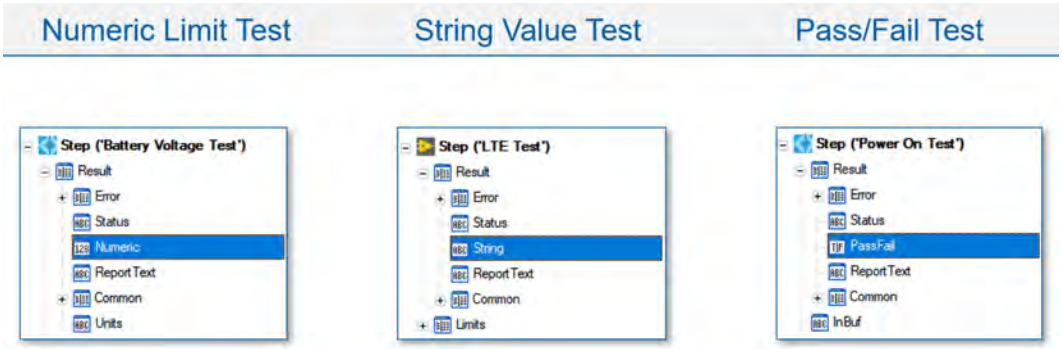
You must decide which kind of step type you need for your sequence. For example, what data should a test step return for a voltage test?

Voltage = 8 or Pass = True

Return the measured value if	Return pass/fail value if
Other test steps need to use that measurement.	The test generates a data structure that is large or complex.

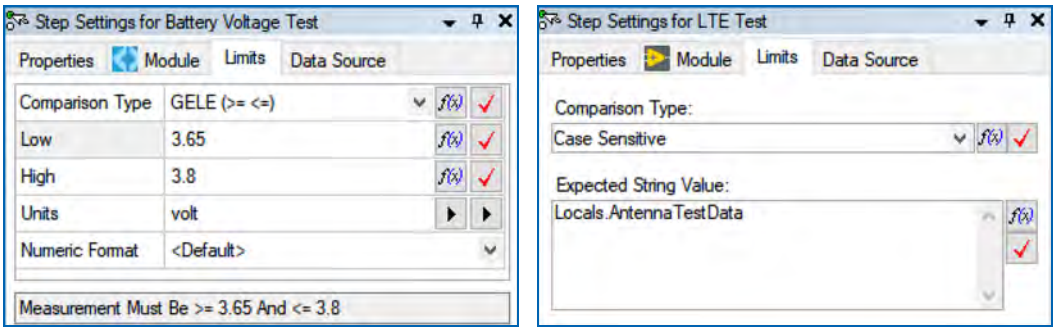
Return the measured value if	Return pass/fail value if
You want to display the result as part of the report that TestStand generates.	You don't need the specific measurement value for other steps.
You want to log the data along with the rest of the test data for the sequence.	You don't need the specific measurement value for logging or the test report.

Where is the Test Result Data Stored?



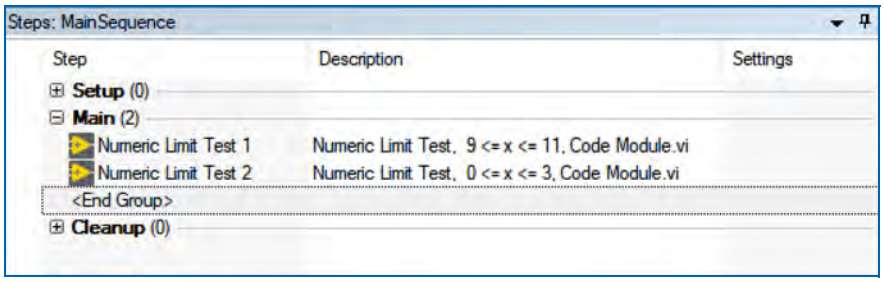
Determine if the Test Step Passed or Failed

Use the Limits tab to determine the pass/fail conditions for the step.



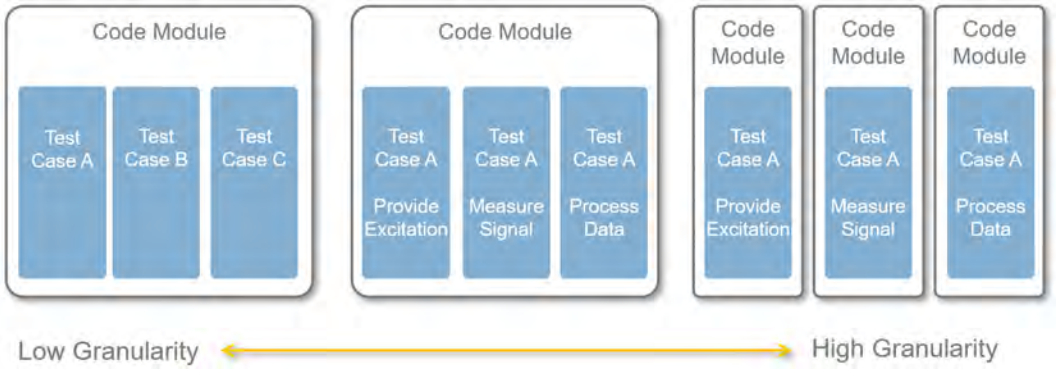
Demonstration: Create a Numeric Limit Test Step

In this demonstration, you create a numeric limit test step.



Choose a Scope for the Code Module

You do not want a single code module to encompass all test cases because it would make the test step too cumbersome and difficult to maintain. On the other hand, you do not want a code module for each individual process because each call to a code module creates overhead that slows down the sequence. Look for the balance in granularity.

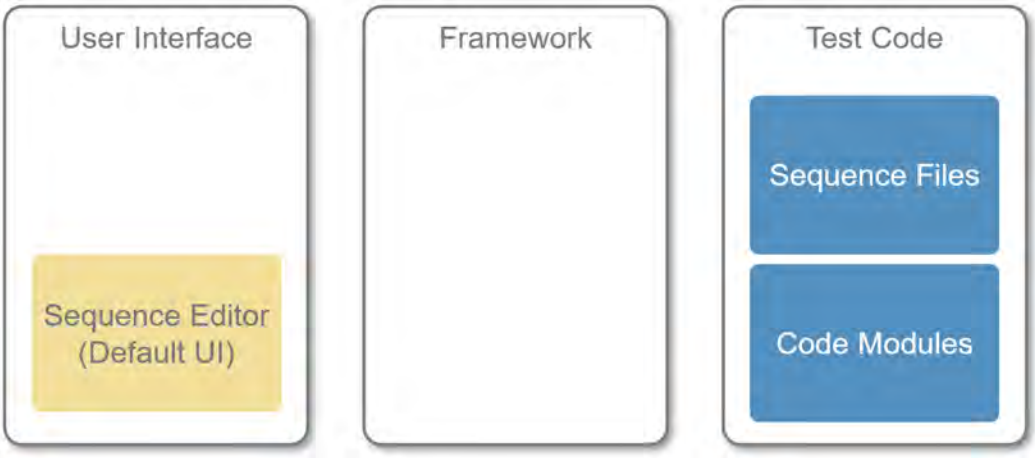


Activity: Implement Test code

List the test cases for the project and decide which test step type to use. Include a list of any data to pass to and from the code module. Refer to the *Test Requirements* section in the Appendix A.

Test Case	Step Type	Data In/Out

TestStand Execution Architecture



Exercise 2-2: (LabVIEW) Call a Code Module

Goal

Add steps that complete critical checks on the solar panel controller.

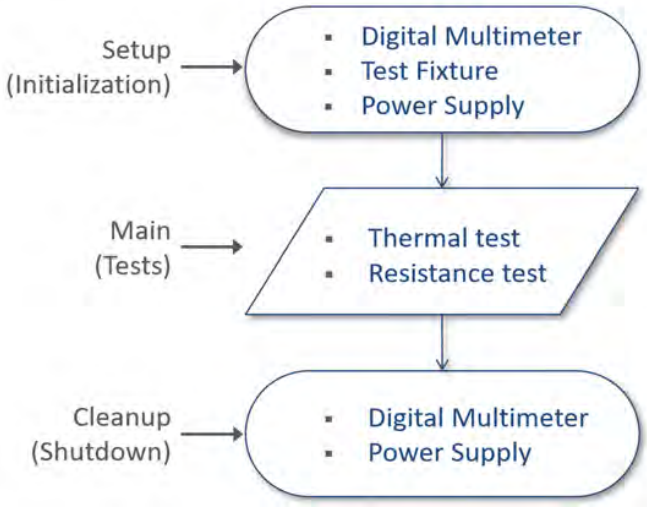
Scenario

You need to make sure that the solar panel controller does not overheat while testing the solar panels under maximum voltage. In addition, you want to abort testing on solar panels that do not have the proper resistance.

In this exercise, you create actions and tests and call them from the sequence. The two numeric limit test steps are described below:

- A resistance test to verify that the resistance across the solar panel, as read by the controller, falls within a normal range.
- A thermal test to ensure that the solar panel controller does not overheat.

Note You create the code module for the thermal test in this exercise.

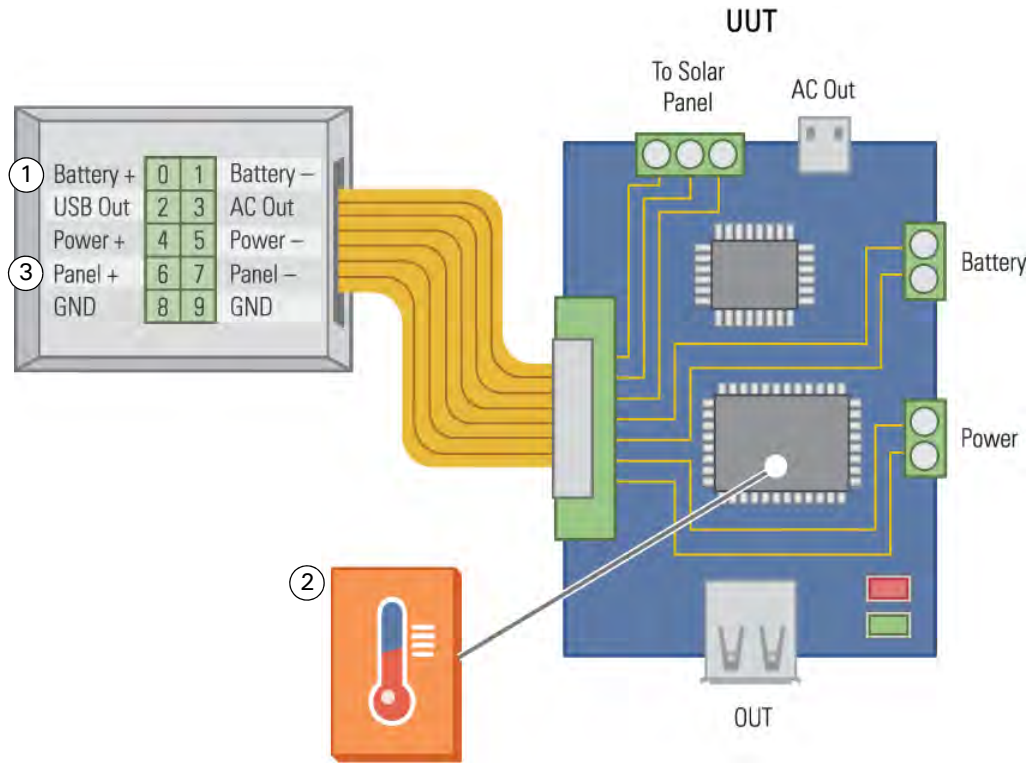


- **Setup step group**—Typically contains steps that initialize instruments, fixtures, or a UUT.
- **Main step group**—Typically contains the bulk of the steps in a sequence, including the steps that test the UUT.
- **Cleanup step group**—Typically contains steps that power off or restore the initial state of instruments, fixtures, and the UUT.

Requirements

- The solar panel controller board temperature shall not exceed 62 degrees C.
- The resistance measured across the solar panel shall be between 110 to 118 Ohms.

Note The following image is also available in Appendix A, *Solar Panel Testing Requirements*.



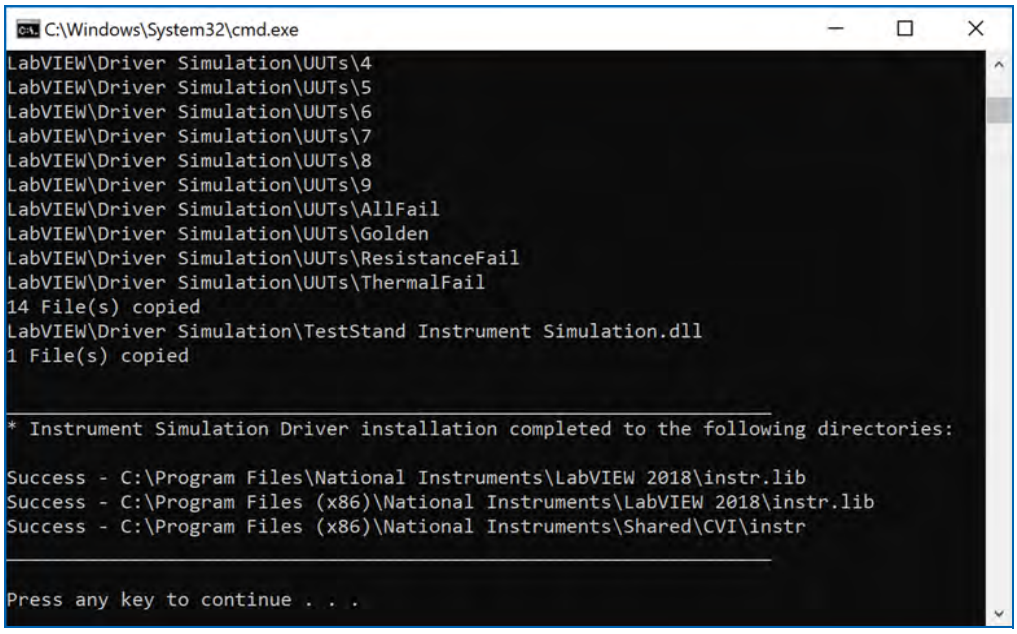
1 Input voltage on pin 0 is at 10V DC.
2 A K-type thermocouple is connected to channel 10 on the test fixture. The thermocouple is placed over the board to measure the solar panel controller board temperature.
3 Resistance from the panel is measured across pins 6 and 7.

Implementation

Note If you have not completed the previous exercise, please copy the solution of the Exercise 2-1 to the appropriate location in the Exercises folder.

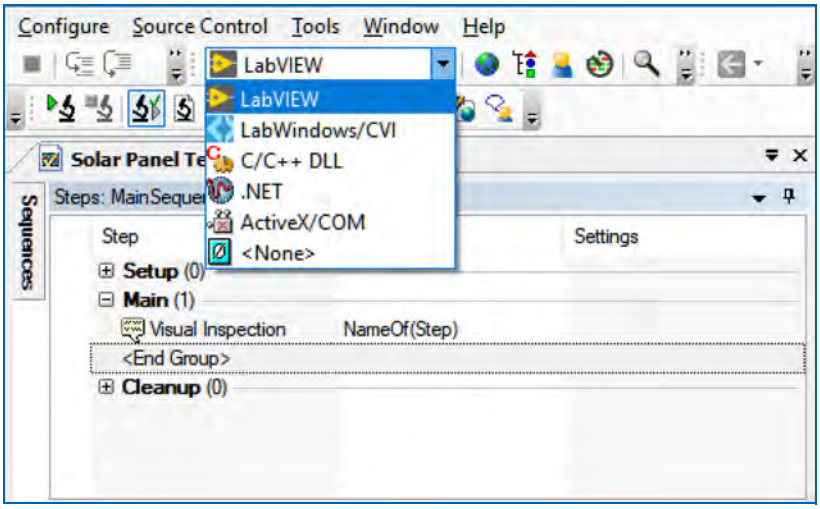
Note Before completing this exercise you must install the Driver Simulation API library.

To install the drivers, make sure that LabVIEW is not running and then navigate to `C:\Exercises\Developing Test Programs\Driver Simulation API` directory and run `Setup.bat` as administrator. The Setup.bat executable installs the necessary components in the LabVIEW and LabWindows/CVI directories.



1. Open the Solar Panel Test sequence in TestStand, if necessary.
2. Add a search directory to allow TestStand to find the Driver Simulation VIs.
   a. Select **Configure»Search Directories**.
   b. Click **Add**.
   c. Browse to the `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation` directory.
   d. Click **OK**. Observe that the directory is added to the search directories table.
   e. Enable the **Search Subdirectories** option.
   f. Click **OK** to dismiss the Edit Search Directories dialog box.

3. Select the **LabVIEW** adapter from the drop-down menu in the menu bar to create steps configured to call a LabVIEW VI.



Note You can specify which LabVIEW server to use with TestStand by configuring the LabVIEW adapter.

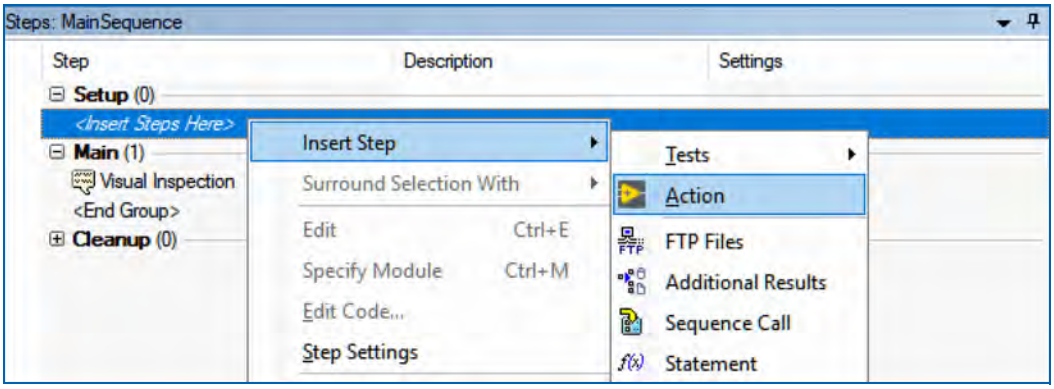
a. To load VIs in TestStand, select **Configure»Adapters** from the menu.

b. Click the **Configure** button to open the LabVIEW Adapter Configuration dialog box.

c. In the LabVIEW Adapter Configuration dialog box, select the **LabVIEW Development System** and click **OK**.

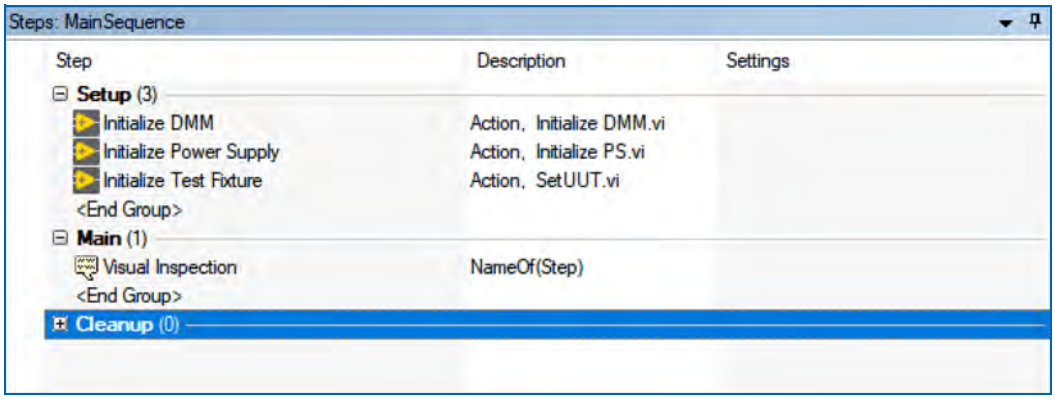
d. Click **Done** to dismiss the Adapter Configuration dialog box.

4. Create three action steps in the Setup step group to initialize the digital multimeter (DMM), power supply and test fixture according to the figure and table below.



Initialize DMM Action Step		
Name	Initialize DMM	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\DMM\Initialize DMM.vi`	(Module tab)
Parameter configuration	**DMM in**—`"DMM"` **error in**—Verify that the <**Default**> option is enabled **error out**—`Step.Result.Error`	(Module tab) The **error out** parameter stores the error information in the step error property. TestStand checks this property after each step and reports an error if one is present.
Initialize Power Supply Action Step		
Name	Initialize Power Supply	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\Power Supply\Initialize PS.vi`	(Module tab)
Parameter configuration	**Power Supply in**—`"PowerSupply"` **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab)

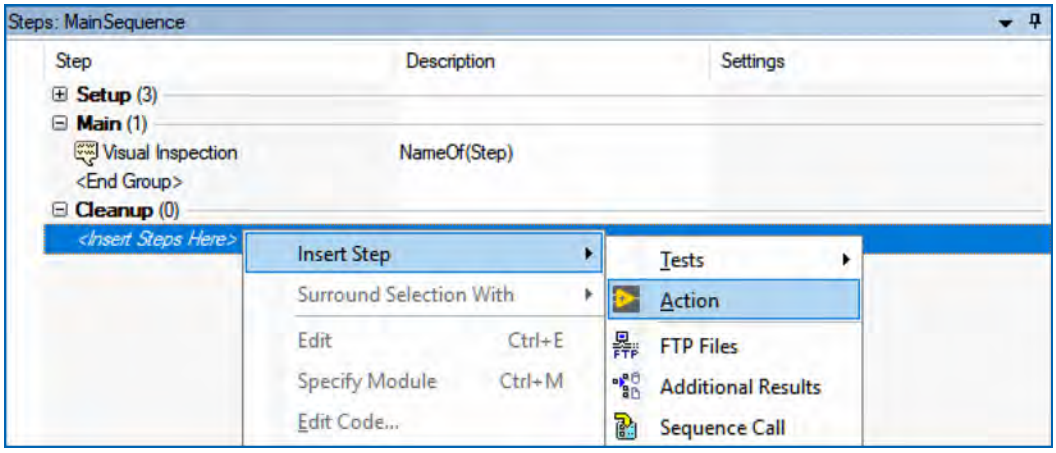
Initialize Test Fixture Action Step		
Name	Initialize Test Fixture	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\Fixture\SetUUT.vi`	(Module tab)
Parameter configuration	**Serial Number**—`"Golden"` **Test Socket**—<**Default** checkbox> **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab) The `"Golden"` serial number represents a known, passing UUT. The test fixture simulation uses this serial number to determine the behavior of the UUT.



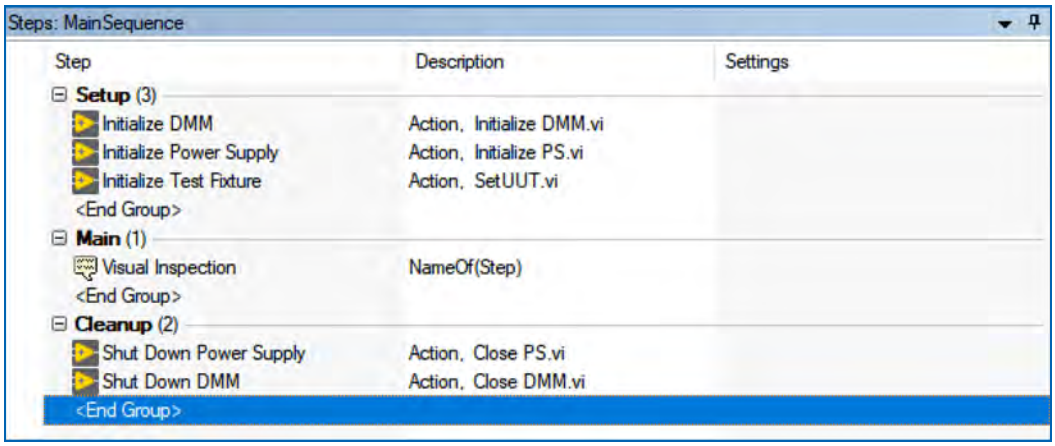
- Three action steps to initialize the DMM, power supply, and test fixture.

Note Do not execute these Setup step group steps before implementing the Cleanup step group, since shut down steps will close the references that you open in Setup. Similarly, if your sequence ever halts as a result of a run-time error or closing the execution, do not abort execution, because that will skip execution of the Cleanup step group.

5. Create two action steps in the Cleanup step group to shut down the power supply and DMM according to the figure and table below.



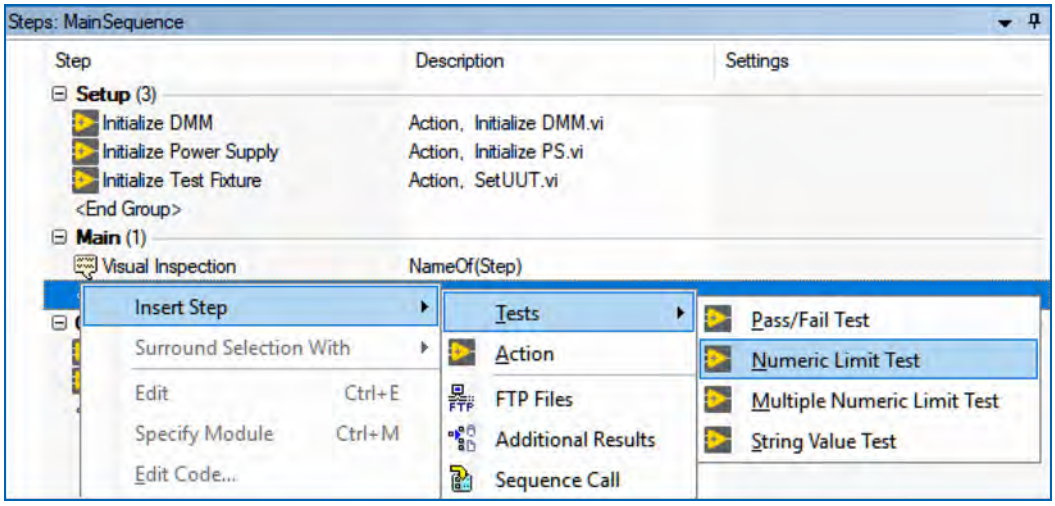
Shut Down Power Supply Action Step		
Name	Shut Down Power Supply	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\Power Supply\Close PS.vi`	(Module tab)
Parameter configuration	**Power Supply in**—`"PowerSupply"` **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab)
Shut Down DMM Action Step		
Name	Shut Down DMM	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\DMM\Close DMM.vi`	(Module tab)
Parameter configuration	**DMM in**—`"DMM"` **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab)



- Two action steps to shut down the power supply and DMM.

6. Create a numeric limit test step to implement the Resistance test.

   a. Insert a Numeric Limit Test step in the Main step group to test the resistance of the solar panel.



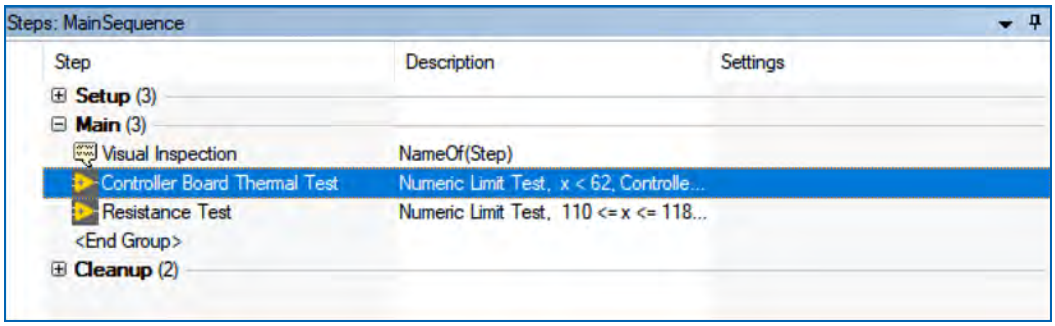
b. Configure the Resistance Test as outlined in the following table.

Resistance Test Numeric Limit Step		
Name	Resistance Test	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\DMM\Read Resistance.vi`.	(Module tab)
Parameter configuration	**Test Socket**—<**Default** checkbox> **DMM in**—`"DMM"` **Channel 1**—`6` **Channel 2**—`7` **error in**—<**Default** checkbox> **Resistance**—`Step.Result.Numeric` **error out**—`Step.Result.Error`	(Module tab) The **Resistance** step property is the default data source property, which is compared against the test limits to determine the step result. **DMM out** does not need any further configuring.
Data Source configuration	**Data Source Expression**—`Step.Result.Numeric`	(Data Source tab) You can change this field to configure the data source to use a different property to determine the step result.
Limits configuration	**Comparison Type**—GELE (>= <=) **Low**—110 **High**—118	(Limits tab) GELE means greater than or equal to and less than or equal to.

7. Create a numeric limit test step to test the temperature of the solar panel controller board.

   a. Add a new Numeric Limit Test in the Main step group.

   b. Rename the step `Controller Board Thermal Test` and move it between the Visual Inspection and Resistance Test.



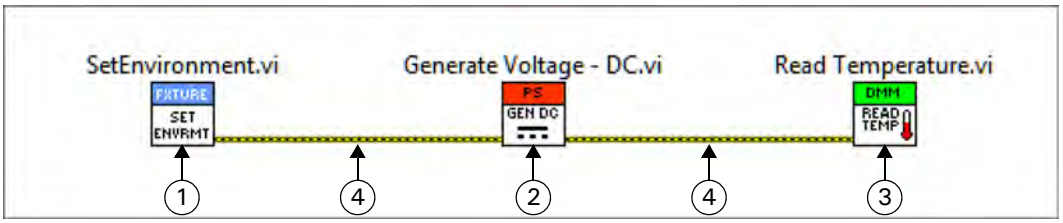
Note Before you can configure the Controller Board Thermal Test step, you need to create the code module that the test step calls.

8. Create a code module in LabVIEW to implement the Controller Board Thermal Test step.

   a. Launch LabVIEW, create a new VI and save it as `Controller Board Thermal Test.vi` in the `C:\Exercises\Developing Test Programs\LabVIEW\Code Modules` directory.

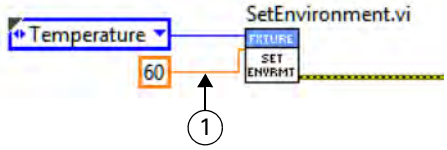
   b. Place a SetEnvironment VI, Generate voltage - DC VI and the Read Temperature VI from the Driver Simulation library to the block diagram of the Controller Board Thermal Test VI as shown below.

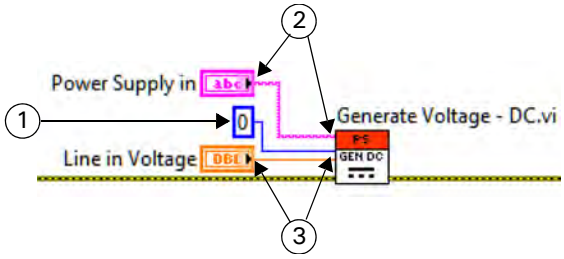
Tip Press <Ctrl-Space> to use the Quick Drop dialog box to locate the VIs or find them on the **Instrument I/O»Instrument Drivers»Instrument Simulation** palette.

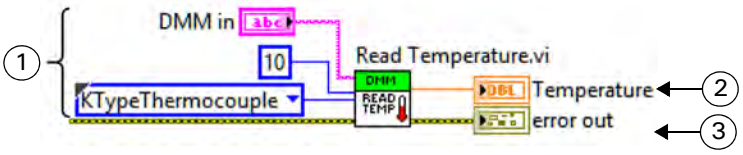


1 **SetEnvironment**—Sets the ambient temperature of the UUT to the value you specify.
2 **Generate Voltage - DC**—Sets the input voltage on the pin you specify.
3 **Read Temperature**—Measures the temperature of the controller board.
4 Wire the error terminals to enforce execution order and propagate error information.
An **error in** control is not needed for TestStand code modules because TestStand manages code module errors.

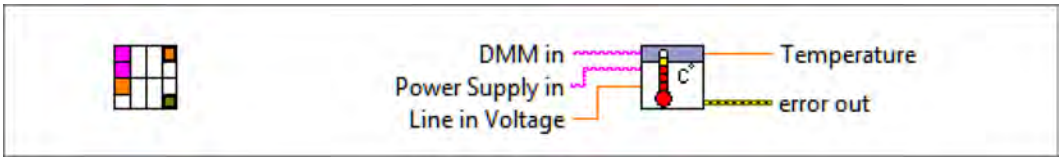
c. Configure the input and output terminals of the VIs as shown in the following table.

SetEnvironment VI

1 Set the ambient temperature of the UUT to 60 degrees C. • Right-click the **Setting** terminal and create a `temperature` constant. • Wire a `60` constant to the **Value** terminal.

Generate Voltage - DC VI

1 Set the input voltage on pin 0. Wire a `0` constant to the **Channel** input. For simplicity, the hardware simulation layer assumes a 1:1 mapping of hardware channels to pins on the UUT. For example, Channel 0 on the power supply is connected to pin 0 on the UUT. 2 Create a control for the **Power Supply in** input terminal. Right-click the input terminal and select **Create»Control** from the shortcut menu. 3 Create a control for the **Output Voltage** input terminal. Rename the control `Line in Voltage`.

Read Temperature VI

1 Configure the Read Temperature VI input terminals. • **DMM in**—Right-click the input and select **Create»Control**. • **Channel**—Right-click the input and select **Create»Constant.** Enter a value of `10`. • **Sensor Type**—Right-click the input and select **Create»Constant**. Set the value to **KTypeThermocouple**. 2 Create an indicator for the **Temperature (C)** output. Right-click the terminal and select **Create»Indicator**. Rename the indicator `Temperature`. 3 Right-click the error out and select **Create»Indicator.**

d. Configure the connector pane of the VI to match the image below.



e. Add a description to document the VI, by selecting **File»VI Properties** and entering the following text in the **Documentation** category.

```
After applying maximum voltage and maximum ambient temperature,
measure the temperature of the solar panel controller.
```

f. Save and close the VI.

9. Configure the Controller Board Thermal Test as outlined in the following table.

Controller Board Thermal Test Numeric Limit Step		
Name	Controller Board Thermal Test	(Properties tab—General category)
VI Path	Navigate to `C:\Exercises\Developing Test Programs\LabVIEW\Code Modules\Controller Board Thermal Test.vi`.	(Module tab) If you get a warning that the VI does not reside in a search directory, select the **Use a relative path for the file you selected** option.
Parameter configuration	**DMM in**—`"DMM"` **Power Supply in**—`"PowerSupply"` **Line in Voltage**—`10` **Temperature**—`Step.Result.Numeric` **error out**—`Step.Result.Error`	(Module tab) You must uncheck the **Default** checkbox to specify a value.
Limits configuration	**Comparison Type**—LT (<) **Low**—62	(Limits tab) Select LT (<) as the comparison type, because the temperature must be less than a specified value. TestStand uses the **Low** field for all comparison types that have a single value.

Note If the parameters do not match, double-check the connector pane in the VI you created.

10. Save the sequence.

Test

1. Test a passing UUT.
   a. Select **Execute»Run MainSequence** to execute the sequence.
   b. Ensure the initialize and close steps in the setup and cleanup groups complete with the Done status.
   c. Verify that the Thermal Test and Resistance Test pass.
2. Test a Failing UUT.
   a. In the Initialize Test Fixture step, change the Serial Number to `"AllFail"`. This serial number fails all tests except the Thermal Test.
   b. Run the test again, and ensure that the Resistance Test now fails.
3. Change the Serial Number back to `"Golden"`.

End of Exercise 2-2 (LabVIEW)

Exercise 2-2: (LabWindows/CVI) Call a Code Module

Goal

Add steps that complete critical checks on the solar panel controller.

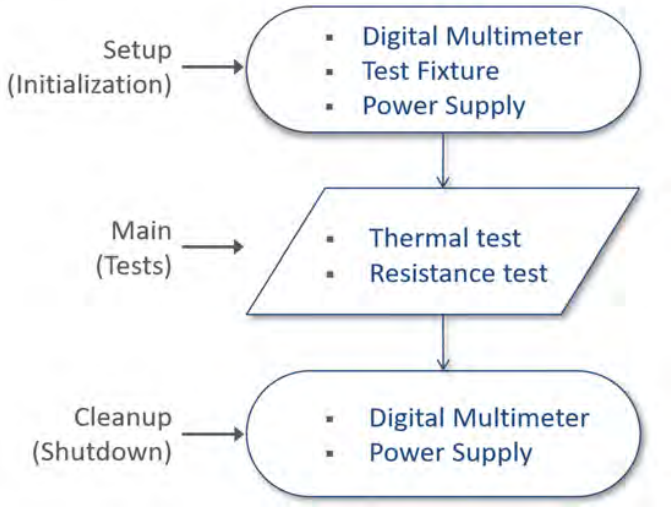
Scenario

You need to make sure that the solar panel controller does not overheat while testing the solar panels under maximum voltage. In addition, you want to abort testing on solar panels that do not have the proper resistance.

In this exercise, you create actions and tests and call them from the sequence. The two numeric limit test steps are described below:

- A resistance test to verify that the resistance across the solar panel, as read by the controller, falls within a normal range.
- A thermal test to ensure that the solar panel controller does not overheat.

Note You create the code module for the thermal test in this exercise.

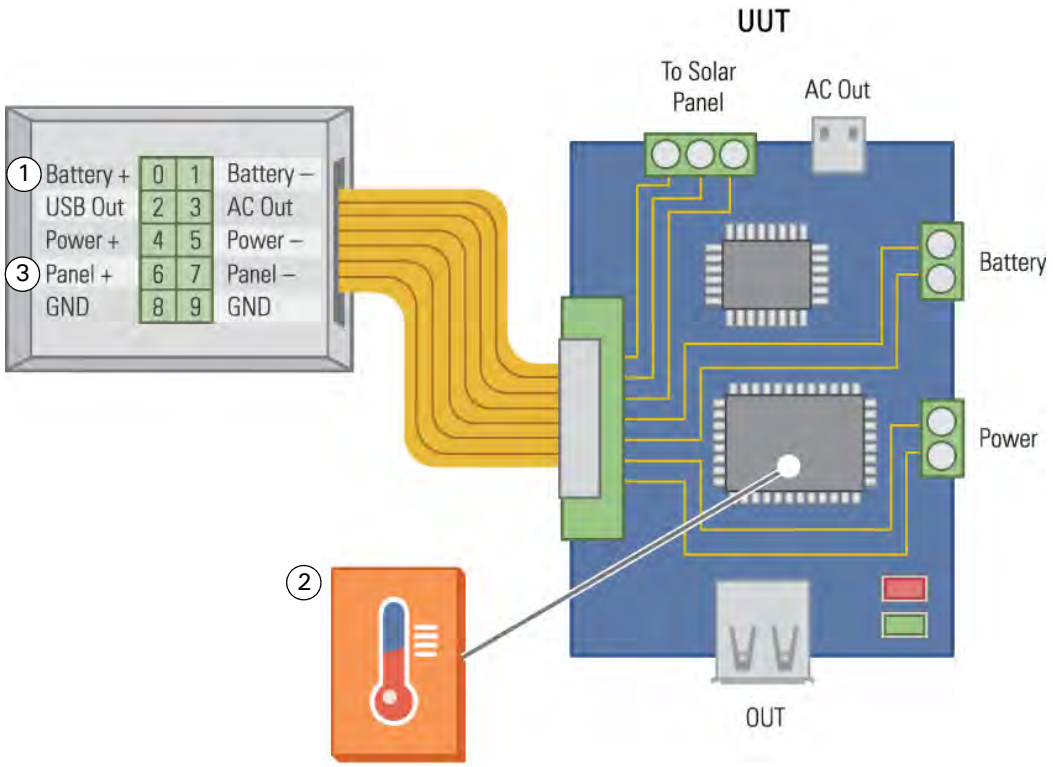


- **Setup step group**—Typically contains steps that initialize instruments, fixtures, or a UUT.
- **Main step group**—Typically contains the bulk of the steps in a sequence, including the steps that test the UUT.
- **Cleanup step group**—Typically contains steps that power off or restore the initial state of instruments, fixtures, and the UUT.

Requirements

- The solar panel controller board temperature shall not exceed 62 degrees C.
- The resistance measured across the solar panel shall be between 110 to 118 Ohms.

Note The following image is also available in Appendix A, *Solar Panel Testing Requirements*.



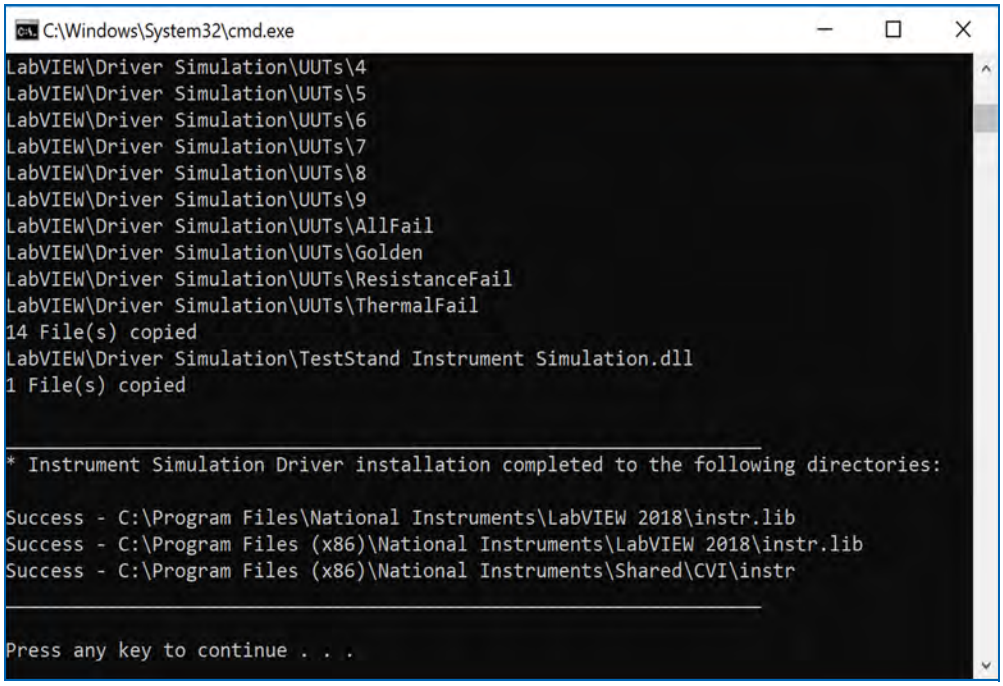
1 Input voltage on pin 0 is at 10V DC.
2 A K-type thermocouple is connected to channel 10 on the test fixture. The thermocouple is placed over the board to measure the solar panel controller board temperature.
3 Resistance from the panel is measured across pins 6 and 7.

Implementation

Note If you have not completed the previous exercise, please copy the solution of the Exercise 2-1 to the appropriate location in the Exercises folder.

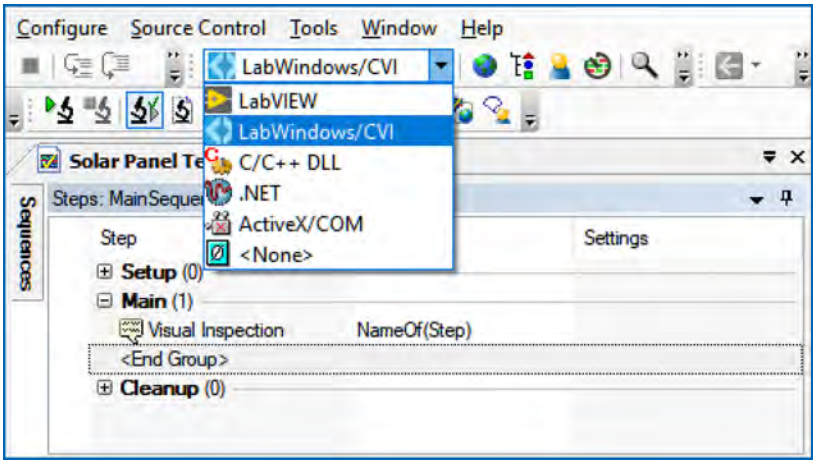
Note Skip the following step if you have already completed the Exercise 2-2 for LabVIEW.

To install the drivers, make sure that LabVIEW is not running and then navigate to `C:\Exercises\Developing Test Programs\Driver Simulation API` directory and run `Setup.bat` as administrator. The Setup.bat executable installs the necessary components in the LabVIEW and LabWindows/CVI directories.

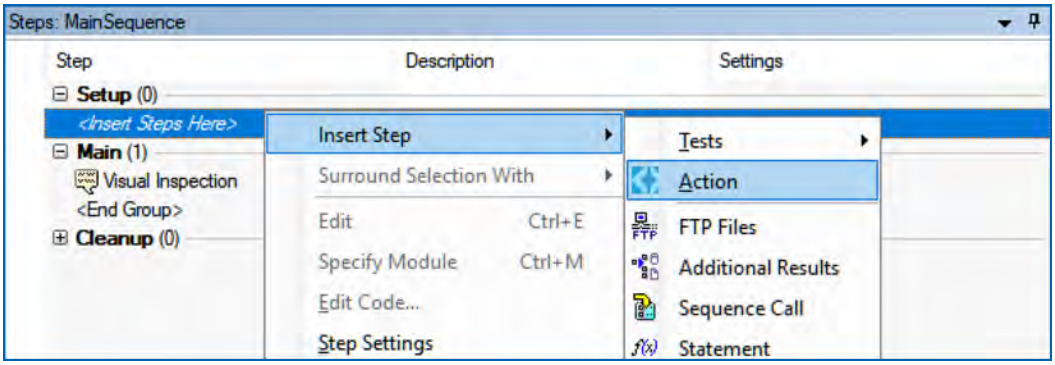


1. Open the Solar Panel Test sequence in TestStand, if necessary.
2. Add a search directory to allow TestStand to find the driver simulation DLL.
   a. Select **Configure»Search Directories.**
   b. Click **Add.**
   c. Browse to the `<instr>` directory for CVI. For example the shared directory might be located at `C:\Program Files(x86)\National Instruments\Shared\CVI` directory.
   d. Click **OK**. Observe that the directory is added to the search directories table
   e. Enable the **Search Subdirectories** option.
   f. Click **OK** to dismiss the Edit Search Directories dialog box.

3. Select the **LabWindows/CVI** adapter from the drop-down menu in the menu bar to create steps configured to call a CVI DLL.

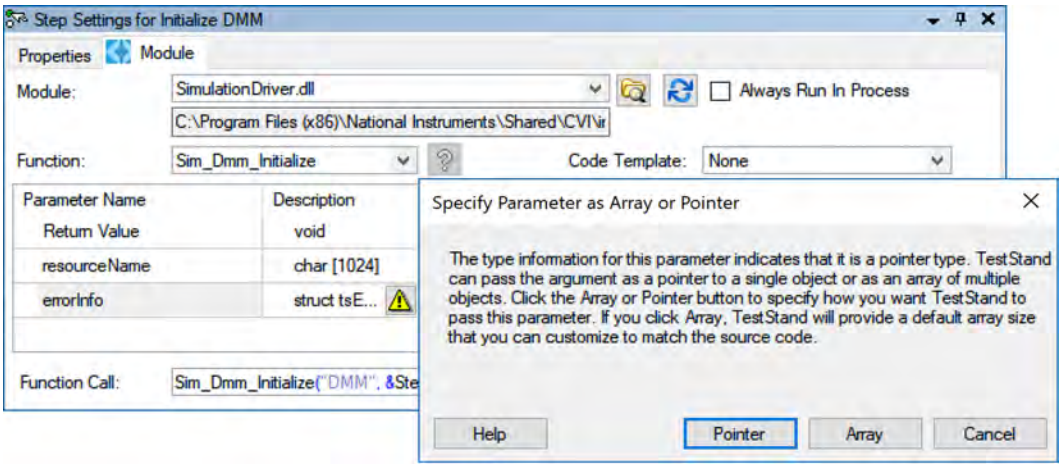


4. Create three action steps in the Setup step group to initialize the digital multimeter (DMM), power supply and test fixture according to the figure and tables below.



Initialize DMM Action Step		
Name	Initialize DMM	(Properties tab—General category)
Module	Navigate to `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.dll` **Function:** Sim_Dmm_Initialize	(Module tab)
Parameter configuration	**resourceName** —`"DMM"` **errorInfo**—`Step.Result.Error`	(Module tab) **errorInfo** stores the error information in the step error property. TestStand checks this property after each step and reports an error if one is present.

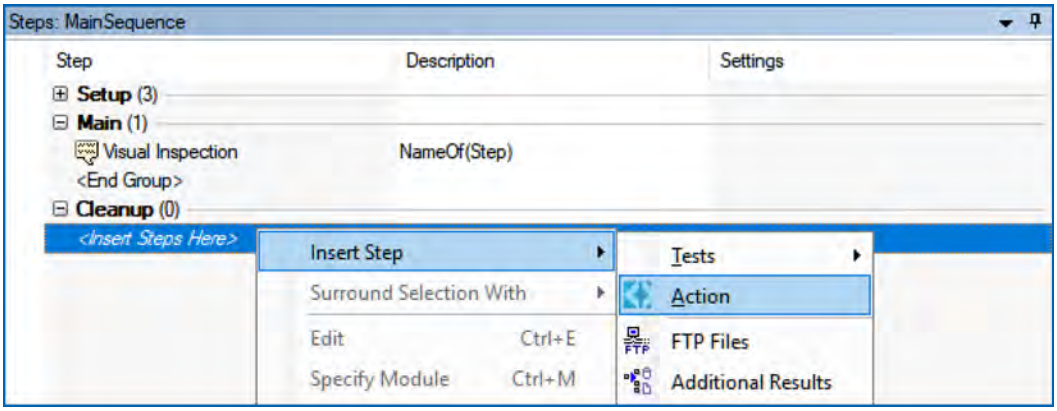
Note If you see a warning icon in the **Description** field, click the yellow triangle and select **Pointer** from the dialog box.



Initialize Power Supply Action Step		
Name	Initialize Power Supply	(Properties tab—General category)
Module	Navigate to `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.dll` **Function:** Sim_PowerSupply_Initialize	(Module tab)
Parameter configuration	**resourceName**—`"PowerSupply"` **errorInfo** —`Step.Result.Error` (Specify as pointer, if necessary.)	(Module tab)
Initialize Test Fixture Action Step		
Name	Initialize Test Fixture	(Properties tab—General category)
Module	Navigate to `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.dll` **Function:** Sim_Fixture_SetUUT	(Module tab)
Parameter configuration	**Serial Number**—`"Golden"` **socketIndex**—Enter `0` **errorInfo**—`Step.Result.Error` (Specify as pointer, if necessary.)	(Module tab) The `"Golden"` serial number represents a known, passing UUT. The test fixture simulation uses this serial number to determine the behavior of the UUT.

Note Do not execute these Setup step group steps before implementing the Cleanup step group, since shut down steps will close the references that you open in Setup. Similarly, if your sequence ever halts as a result of a run-time error or closing the execution, do not abort execution, because that will skip execution of the Cleanup step group.

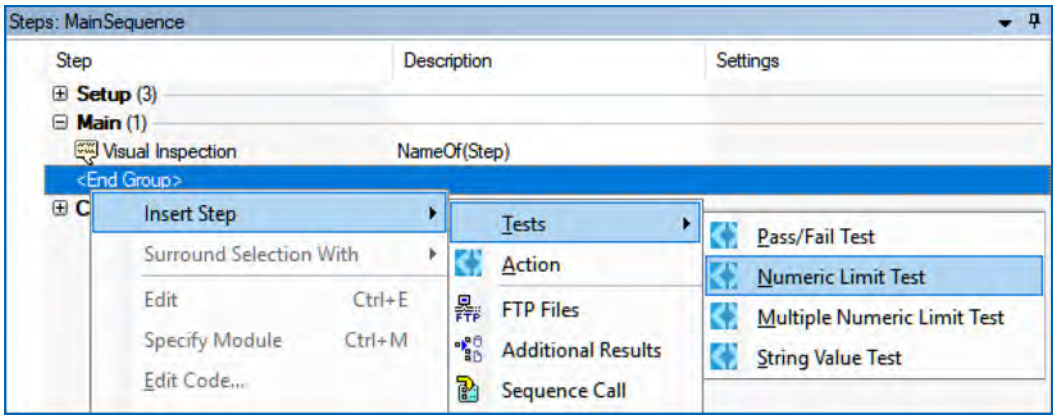
5. Create two action steps in the Cleanup step group to shut down the power supply and DMM according to the figure and table below.



Shut Down Power Supply Action Step		
Name	Shut Down Power Supply	(Properties tab—General category)
Module	Navigate to `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.dll` **Function:** Sim_PowerSupply_Close	(Module tab)
Parameter configuration	**resourceName**—`"PowerSupply"` **errorInfo**—`Step.Result.Error` (Specify as pointer, if necessary.)	(Module tab)
Shut Down DMM Action Step		
Name	Shut Down DMM	(Properties tab—General category)
Module	Navigate to `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.dll` **Function:** Sim_Dmm_Close	(Module tab)
Parameter configuration	**resourceName**—`"DMM"` **errorInfo**—`Step.Result.Error` (Specify as pointer, if necessary.)	(Module tab)

6. Create a numeric limit test step to implement the resistance test.

   a. Create a Numeric Limit Test step in the Main step group to test the resistance of the solar panel.



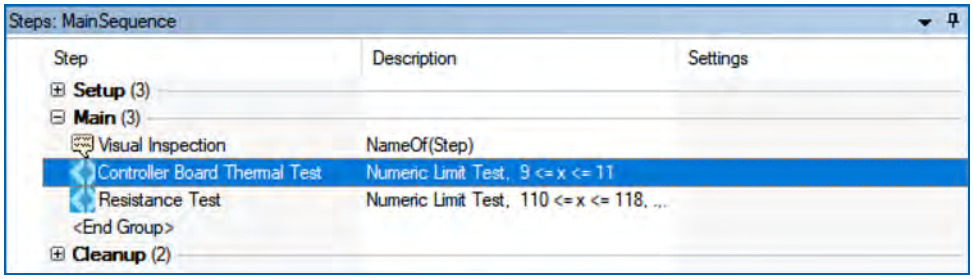
   b. Configure the Resistance Test as outlined in the following table.

Resistance Test Numeric Limit Step		
Name	Resistance Test	(Properties tab—General category)
Module	Navigate to `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.dll` **Function:** Sim_Dmm_ReadResistance	(Module tab)
Parameter configuration	**resourceName**—`"DMM"` **Channel 1**—`6` **Channel 2**—`7` **testSocket**—`0` **result**—`Step.Result.Numeric` (Specify as pointer, if necessary.) **errorInfo**—`Step.Result.Error` (Specify as pointer, if necessary.)	(Module tab) The **result** step property is the default data source property, which is compared against the test limits to determine the step result.
Data Source configuration	Data Source Expression—`Step.Result.Numeric`	(Data Source tab) You can change this field to configure the data source to use a different property to determine the step result.
Limits configuration	**Comparison Type**—GELE (>= <=) **Low**—110 **High**—118	(Limits tab) GELE means greater than or equal to and less than or equal to.

7. Create a numeric limit test step to test the temperature of the controller board.

   a. Add a new Numeric Limit Test in the Main step group.

   b. Rename the step `Controller Board Thermal Test` and move it between the Visual Inspection and Resistance Test.

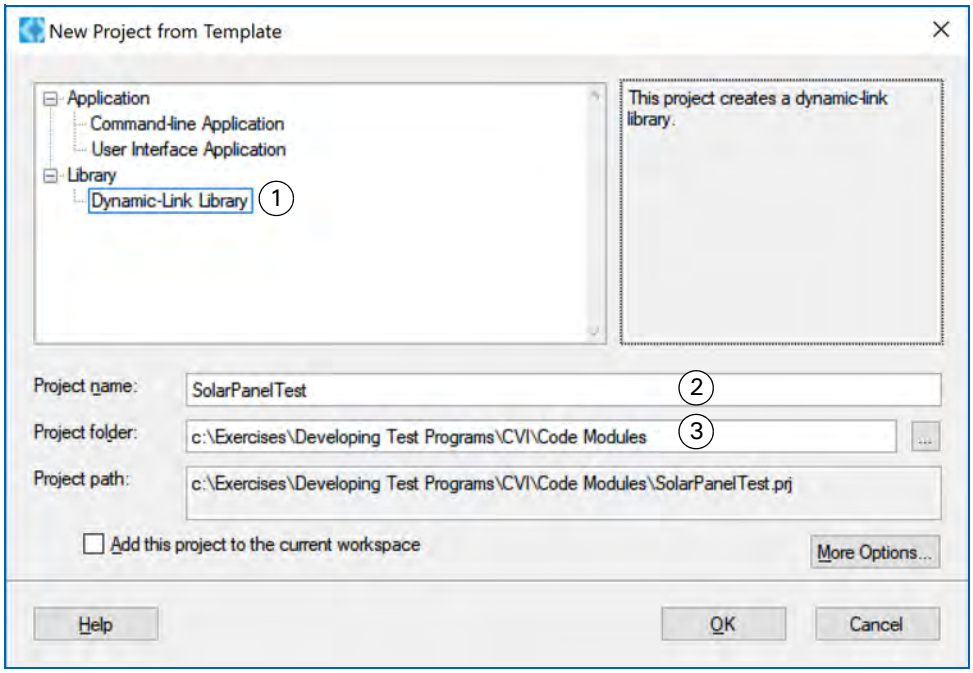


Note Before you can configure the Controller Board Thermal Test step, you need to create the code module that the test step calls.

8. Create a code module in CVI to implement the Controller Board Thermal Test step.

   a. In the LabWindows/CVI Development System, create a **New»Project from Template**.

   b. Configure the new project as described below.

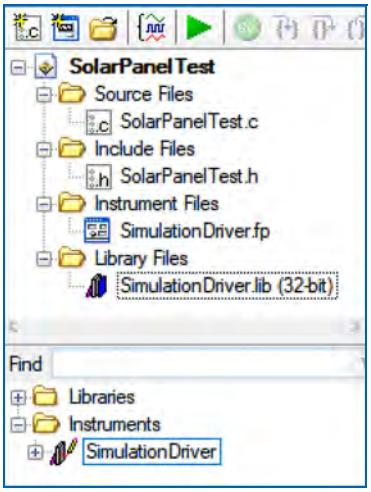


1. Choose **Dynamic-Link Library** as a type.
2. Name the project `SolarPanelTest`.
3. In the **Project Folder** field, browse to the `C:\Exercises\Developing Test Programs\CVI\Code Modules` folder and click **OK**.

Note The template generates a source file (.c) and a header file (.h).

c. Add the library and instrument files for the simulation driver.

- Right-click the project in the project pane and select **Add Existing File**.
- Navigate to the file `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.fp`.
- Right-click the project in the project pane and select **Add Existing File.**
- Browse to the file `C:\Program Files (x86)\National Instruments\Shared\CVI\instr\Driver Simulation\SimulationDriver.lib`.
- Verify the project files are added.



d. Add a reference to the simulation driver in the SolarPanelTest header file and declare a new function for the Controller Board thermal test.

- Double-click **SolarPanelTest.h** and edit the file based on the image below.

SolarPanelTest.h ×

```
#include "cvidef.h"
#include "SimulationDriver.h"

//==============================================================================
// Global functions

void ControllerBoardThermalTest (char* dmm, char* powerSupply, double lineInVoltage,
                                 double* temperature, Error* errorInfo);
```

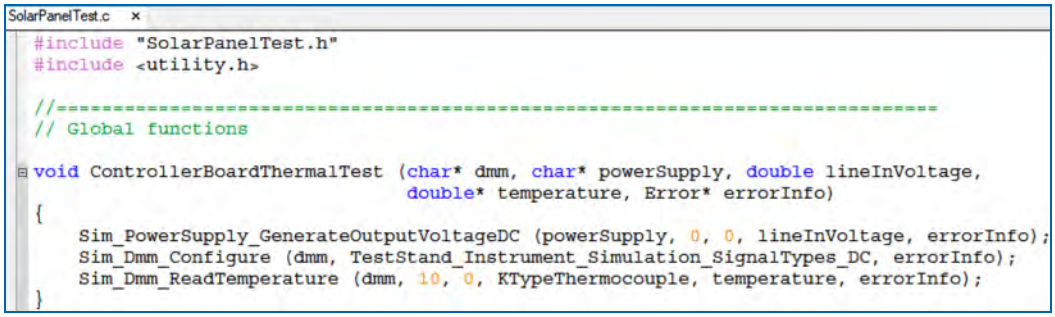
Note **Error** is an enumeration defined to match TestStand's error container structure.

Note Output values can be implemented using the return value or using a pointer (passing by reference). Passing by reference propagates any changes made to the value back to the TestStand property you configure for that parameter. For example, the step property Step.Result.Numeric can be passed into the temperature variable function and it will be updated with the step result when the function executes.

Tip Press <Ctrl-S> to save changes.

e. Double-click **SolarPanelTest.c** to edit the SolarPanelTest source file.

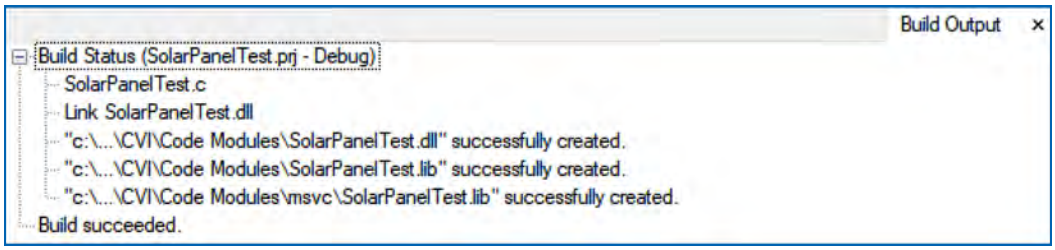
- Move the <**utility.h**> source file under the "**SolarPanelTest.h**".
- Implement the new **ControllerBoardThermalTest** function to the source file based on the image below.



- Save changes.

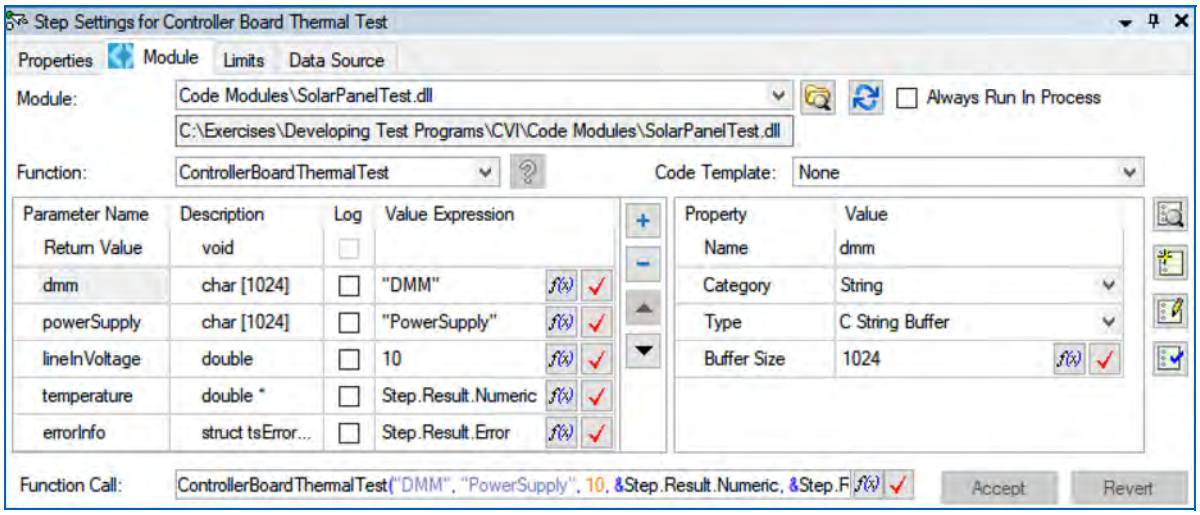
f. Build the code module.

- Select **Build»Build** to create the DLL file. Verify in the status pane that no errors occur while building.
- Verify that the DLL is created in the `C:\Exercises\Developing Test Programs\CVI\Code Modules` directory.

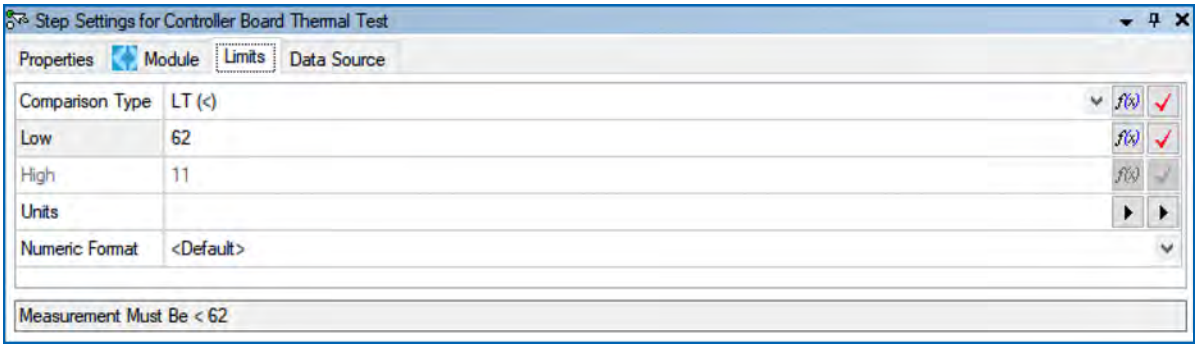


g. Configure the Controller Board Thermal Test step to call the new DLL.

- In TestStand, select the Controller Board Thermal Test step and configure the Module tab as follows.



h. Configure the test limits for the step in the Limits tab:



9. Save the sequence.

Test

1. Test a passing UUT.
   a. Execute the sequence using **Execute»Run MainSequence**.
   b. Ensure that the Initialize and Shutdown steps in the Setup and Cleanup groups complete with the Done status.
   c. Ensure that both tests passed.
2. Test a Failing UUT.
   a. In the Initialize Test Fixture step, change the serial number to `"AllFail"`. This represents a known bad UUT.
   b. Run the test again, and ensure that the Resistance Test now fails.
   c. Restore the serial number to `"Golden"`.

End of Exercise 2-2 (LabWindows/CVI)

Exercise 2-2: (TestStand Only) Call a Code Module

Goal

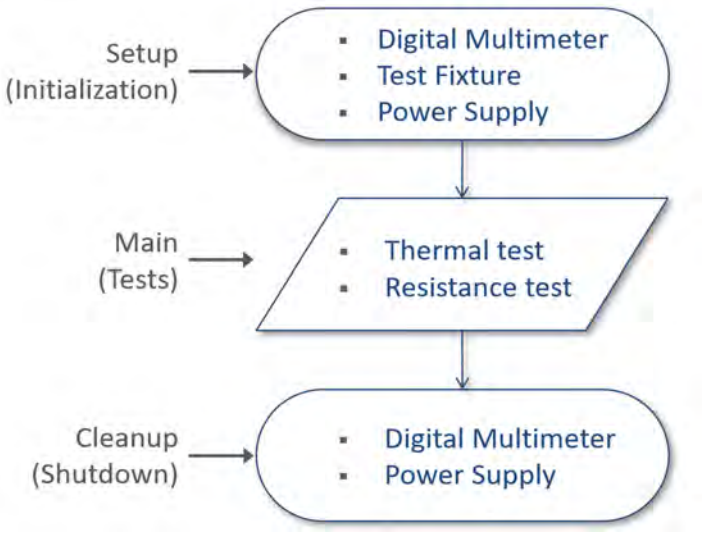
Add steps that complete critical checks on the solar panel controller.

Scenario

You need to make sure that the solar panel controller does not overheat while testing the solar panels under maximum voltage. In addition, you want to abort testing on solar panels that do not have the proper resistance.

In this exercise, you create actions and tests and call them from the sequence. The two numeric limit test steps are described below:

- A resistance test to verify that the resistance across the solar panel, as read by the controller, falls within a normal range.
- A thermal test to ensure that the solar panel controller does not overheat.

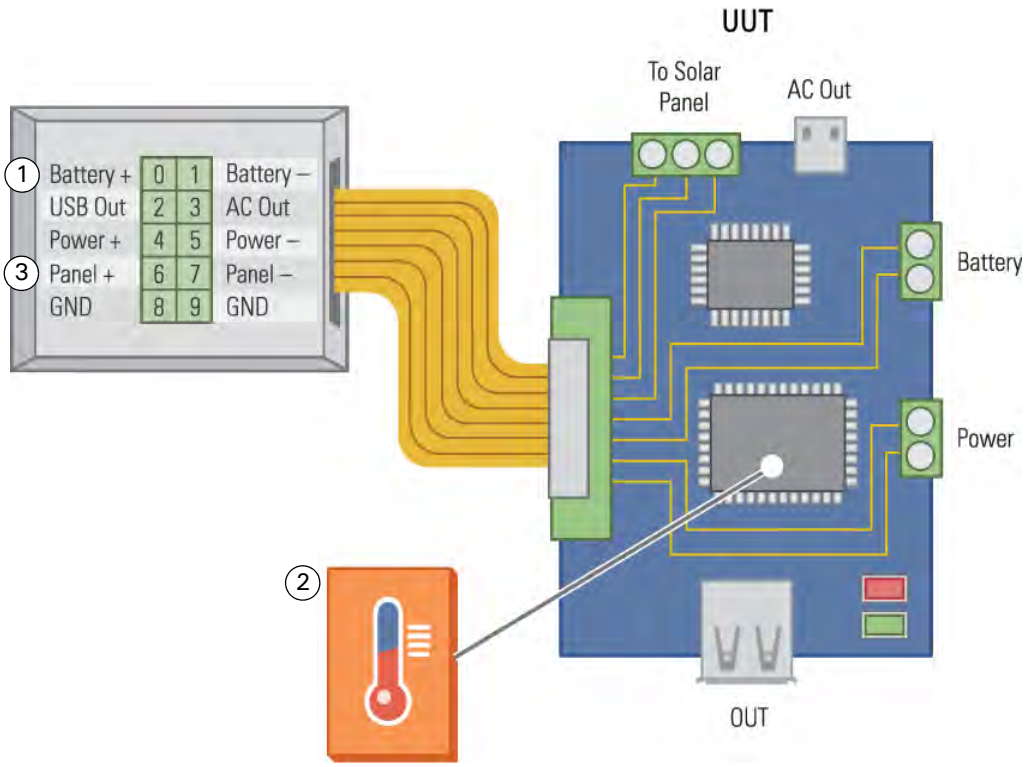


- **Setup step group**—Typically contains steps that initialize instruments, fixtures, or a UUT.
- **Main step group**—Typically contains the bulk of the steps in a sequence, including the steps that test the UUT.
- **Cleanup step group**—Typically contains steps that power off or restore the initial state of instruments, fixtures, and the UUT.

Requirements

- The solar panel controller board temperature shall not exceed 62 degrees C.
- The resistance measured across the solar panel shall be between 110 to 118 Ohms.

Note The following image is also available in Appendix A, *Solar Panel Testing Requirements*.



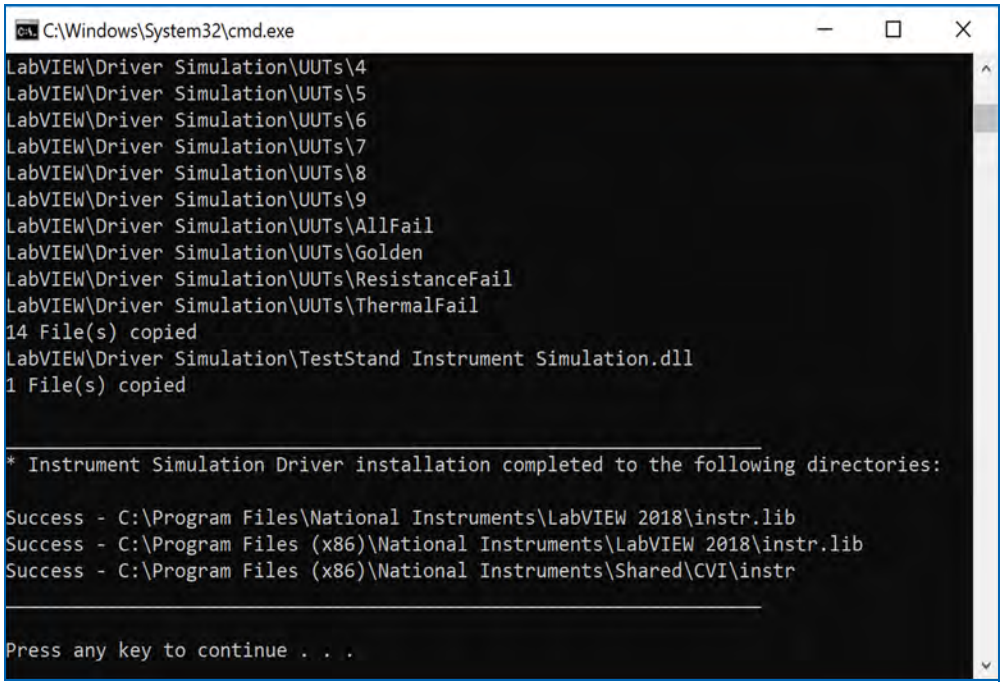
1 Input voltage on pin 0 is at 10V DC.
2 A K-type thermocouple is connected to channel 10 on the test fixture. The thermocouple is placed over the board to measure the controller board temperature.
3 Resistance from the panel is measured across pins 6 and 7.

Implementation

Note If you have not completed the previous exercise, please copy the solution of the Exercise 2-1 to the appropriate location in the Exercises folder.

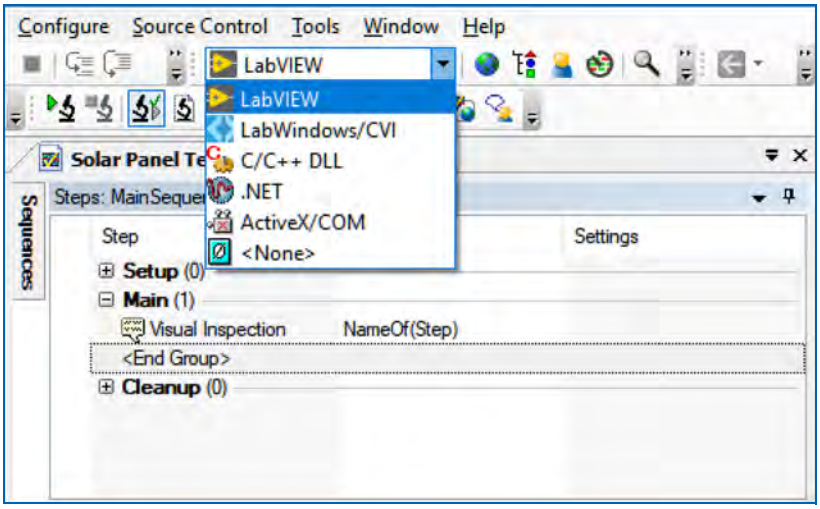
Note Skip the following step, if you have already finished the Exercise 2-2 for LabVIEW or LabWindows/CVI.

To install the drivers, make sure that LabVIEW is not running and then navigate to `C:\Exercises\Developing Test Programs\Driver Simulation API` directory and run `Setup.bat` as administrator. The Setup.bat executable installs the necessary components in the LabVIEW and LabWindows/CVI directories.



1. Open the Solar Panel Test sequence in TestStand, if necessary.
2. Add a search directory to allow TestStand to find the Driver Simulation VIs.
   a. Select **Configure»Search Directories**.
   b. Click **Add**.
   c. Browse to the `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation` directory.
   d. Click **OK**. Observe that the directory is added to the search directories table.
   e. Place a checkmark in the **Search Subdirectories** option.
   f. Click **OK** to dismiss the **Edit Search Directories** dialog box.

3. Select the **LabVIEW** adapter from the drop-down menu in the menu bar to create steps configured to call a LabVIEW VI.



Note You can specify which LabVIEW server to use with TestStand by configuring the LabVIEW adapter.

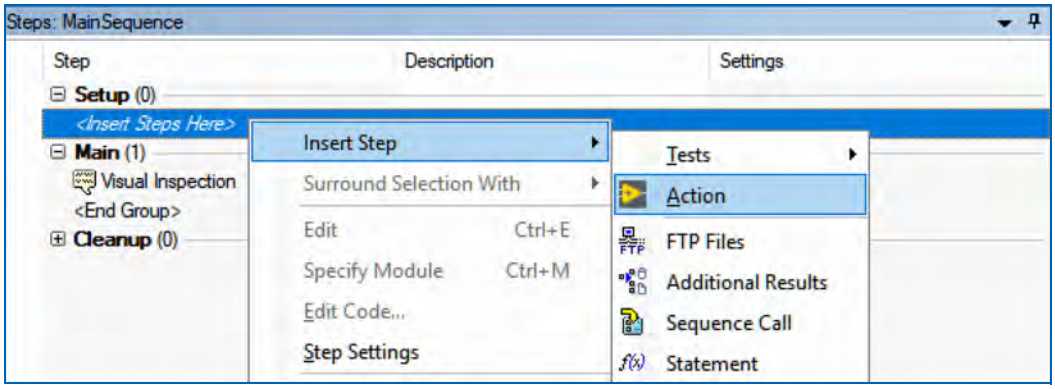
a. To load VIs in TestStand, select **Configure»Adapters** from the menu.

b. Click the **Configure** button to open the LabVIEW Adapter Configuration dialog box.

c. In the LabVIEW Adapter Configuration dialog box, select the **LabVIEW Development System** and click **OK**.

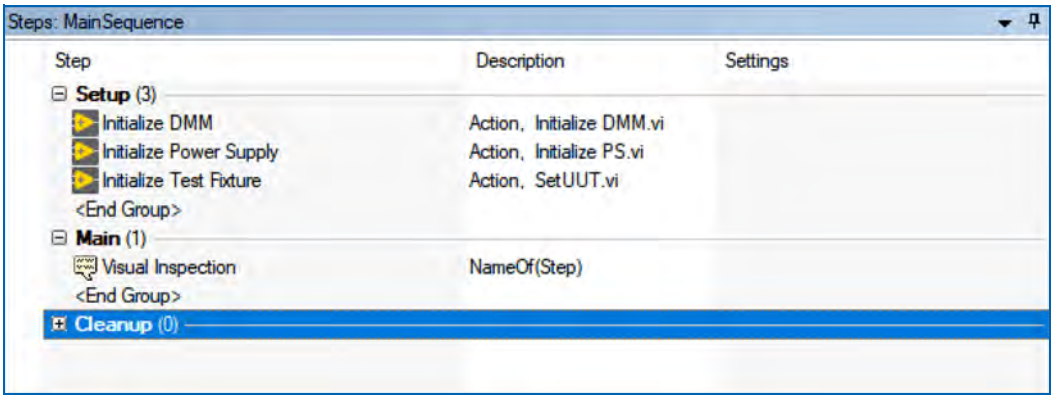
d. Click **Done** to dismiss the Adapter Configuration dialog box.

4. Create three action steps in the Setup step group to initialize the digital multimeter (DMM), power supply and test fixture according to the figure and table below.



Initialize DMM Action Step		
Name	Initialize DMM	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\DMM\Initialize DMM.vi`	(Module tab)
Parameter configuration	**DMM in**—`"DMM"` **error in**—Verify that the **Default** checkbox is enabled. **error out**—`Step.Result.Error`	(Module tab) The **error out** parameter stores the error information in the step error property. TestStand checks this property after each step and reports an error if one is present.
Initialize Power Supply Action Step		
Name	Initialize Power Supply	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\Power Supply\Initialize PS.vi`	(Module tab)
Parameter configuration	**Power Supply in**—`"PowerSupply"` **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab)

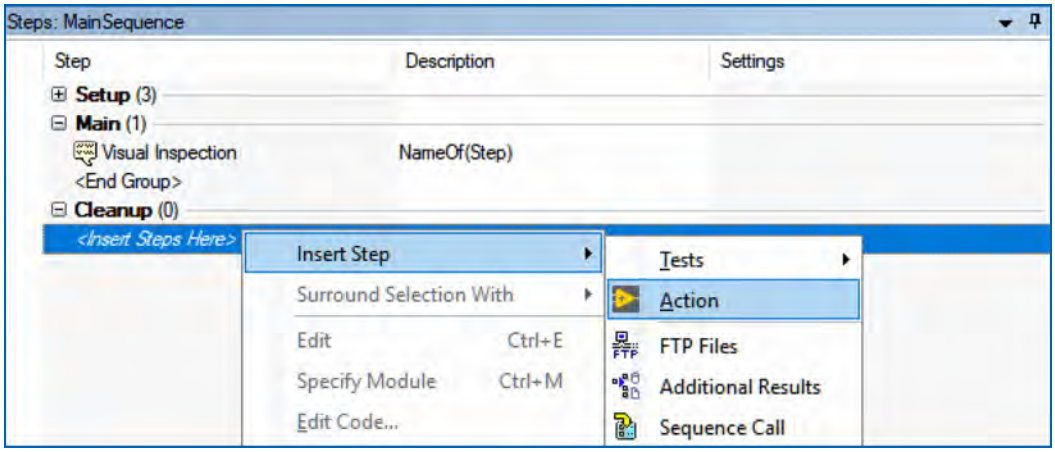
Initialize Test Fixture Action Step		
Name	Initialize Test Fixture	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\DriverSimulation\Fixture\SetUUT.vi`	(Module tab)
Parameter configuration	**Serial Number**—`"Golden"` **Test Socket**—<**Default** checkbox> **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab) The `"Golden"` serial number represents a known, passing UUT. The test fixture simulation uses this serial number to determine the behavior of the UUT.



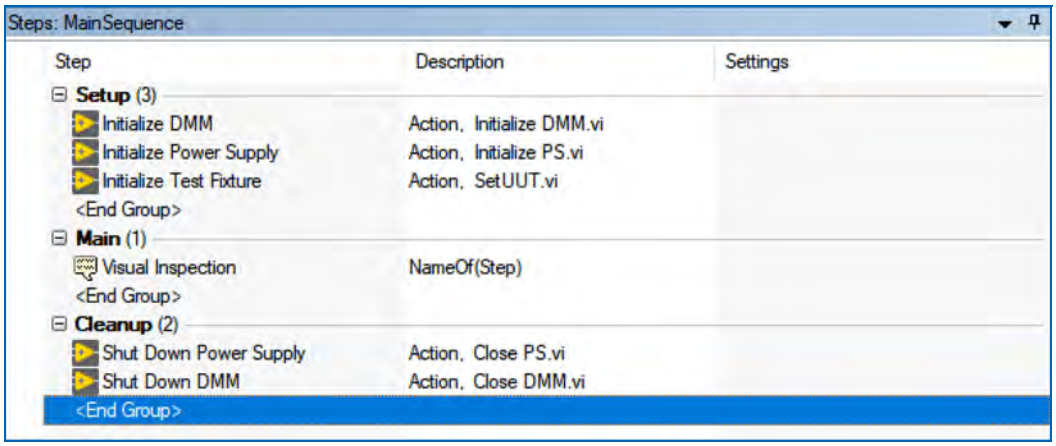
- Three action steps to initialize the DMM, power supply, and test fixture.

Note Do not execute these Setup step group steps before implementing the Cleanup step group, since shut down steps will close the references that you open in Setup. Similarly, if your sequence ever halts as a result of a run-time error or closing the execution, do not abort execution, because that will skip execution of the Cleanup step group.

5. Create two action steps in the Cleanup step group to shut down the power supply and DMM according to the figure and table below.



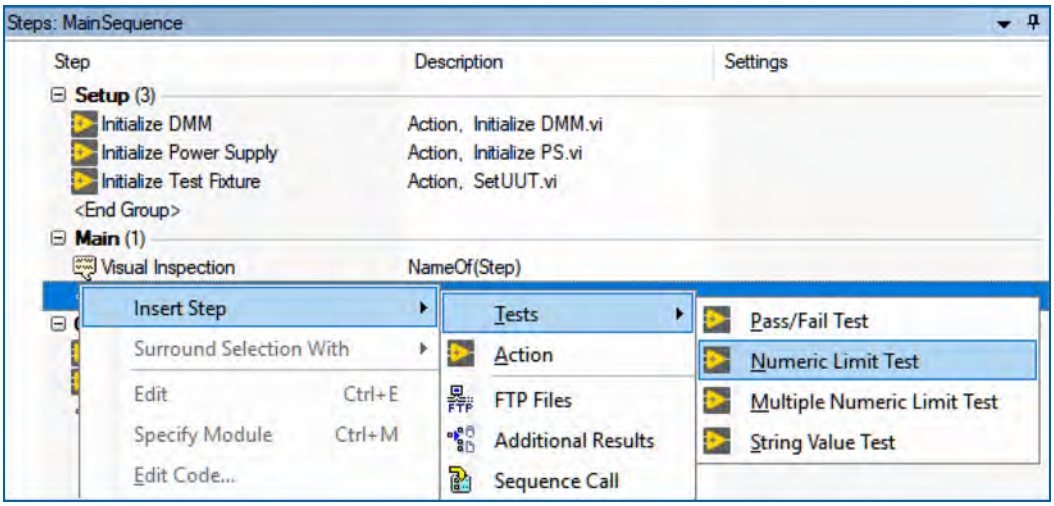
Shut Down Power Supply Action Step		
Name	Shut Down Power Supply	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\Power Supply\Close PS.vi`	(Module tab)
Parameter configuration	**Power Supply in**—`"PowerSupply"` **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab)
Shut Down DMM Action Step		
Name	Shut Down DMM	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\DMM\Close DMM.vi`	(Module tab)
Parameter configuration	**DMM in**—`"DMM"` **error in**—<**Default** checkbox> **error out**—`Step.Result.Error`	(Module tab)



- Two action steps to shutdown the power supply and DMM.

6. Create a numeric limit test step to implement the Resistance test.

   a. Insert a Numeric Limit Test step in the Main step group to test the resistance of the solar panel.



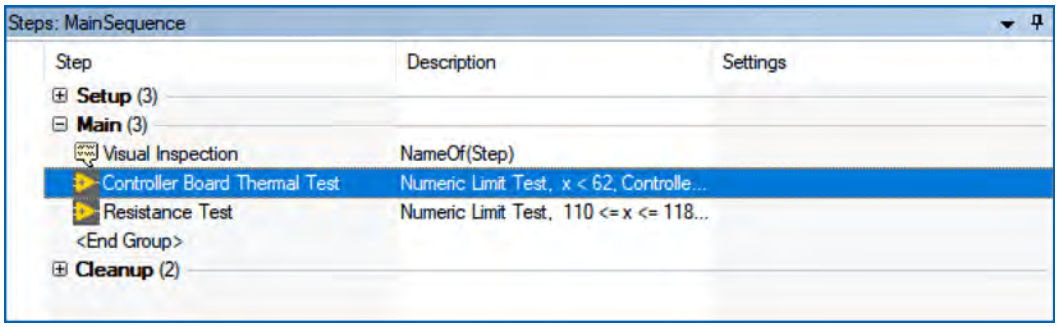
b. Configure the Resistance Test as outlined in the following table.

Resistance Test Numeric Limit Step		
Name	Resistance Test	(Properties tab—General category)
VI Path	Navigate to `C:\Program Files (x86)\National Instruments\<LabVIEW>\instr.lib\Driver Simulation\DMM\Read Resistance.vi`.	(Module tab)
Parameter configuration	**Test Socket**—<**Default** checkbox> **DMM in**—`"DMM"` **Channel 1**—`6` **Channel 2**—`7` **error in**—<**Default** checkbox> **Resistance**—`Step.Result.Numeric` **error out**—`Step.Result.Error`	(Module tab) The **Resistance** step property is the default data source property, which is compared against the test limits to determine the step result. **DMM out** does not need any further configuring.
Data Source configuration	**Data Source Expression**—`Step.Result.Numeric`	(Data Source tab) You can change this field to configure the data source to use a different property to determine the step result.
Limits configuration	**Comparison Type**—GELE (>= <=) **Low**—110 **High**—118	(Limits tab) GELE means greater than or equal to and less than or equal to.

7. Create a numeric limit test step to test the temperature of the solar panel controller board.

   a. Add a new Numeric Limit Test in the Main step group.

   b. Rename the step `Controller Board Thermal Test` and move it between the Visual Inspection and Resistance Test.



8. Configure the Controller Board Thermal Test as outlined in the following table.

Controller Board Thermal Test Numeric Limit Step		
Name	Controller Board Thermal Test	(Properties tab—General category)
VI Path	Navigate to `C:\Exercises\Developing Test Programs\TestStand Only\Code Modules\Controller Board Thermal Test.vi`.	(Module tab) If you get a warning that the VI does not reside in a search directory, select the **Use a relative path for the file you selected** option.
Parameter configuration	**DMM in**—`"DMM"` **Power Supply in**—`"PowerSupply"` **Line in Voltage**—`10` **Temperature**—`Step.Result.Numeric` **error out**—`Step.Result.Error`	(Module tab) You must uncheck the **Default** checkbox to specify a value.
Limits configuration	**Comparison Type**—LT (<) **Low**—62	(Limits tab) Select LT (<) as the comparison type, because the temperature must be less than a specified value. TestStand uses the **Low** field for all comparison types that have a single value.

9. Save the sequence.

Test

1. Test a passing UUT.
   a. Select **Execute»Run MainSequence** to execute the sequence.
   b. Ensure the initialize and close steps in the setup and cleanup groups complete with the Done status.
   c. Verify that the Thermal Test and Resistance Test pass.
2. Test a Failing UUT.
   a. In the Initialize Test Fixture step, change the Serial Number to `"AllFail"`. This serial number fails all tests except the Thermal Test.
   b. Run the test again, and ensure that the Resistance Test now fails.
3. Change the Serial Number back to `"Golden"`.

End of Exercise 2-2 (TestStand Only)

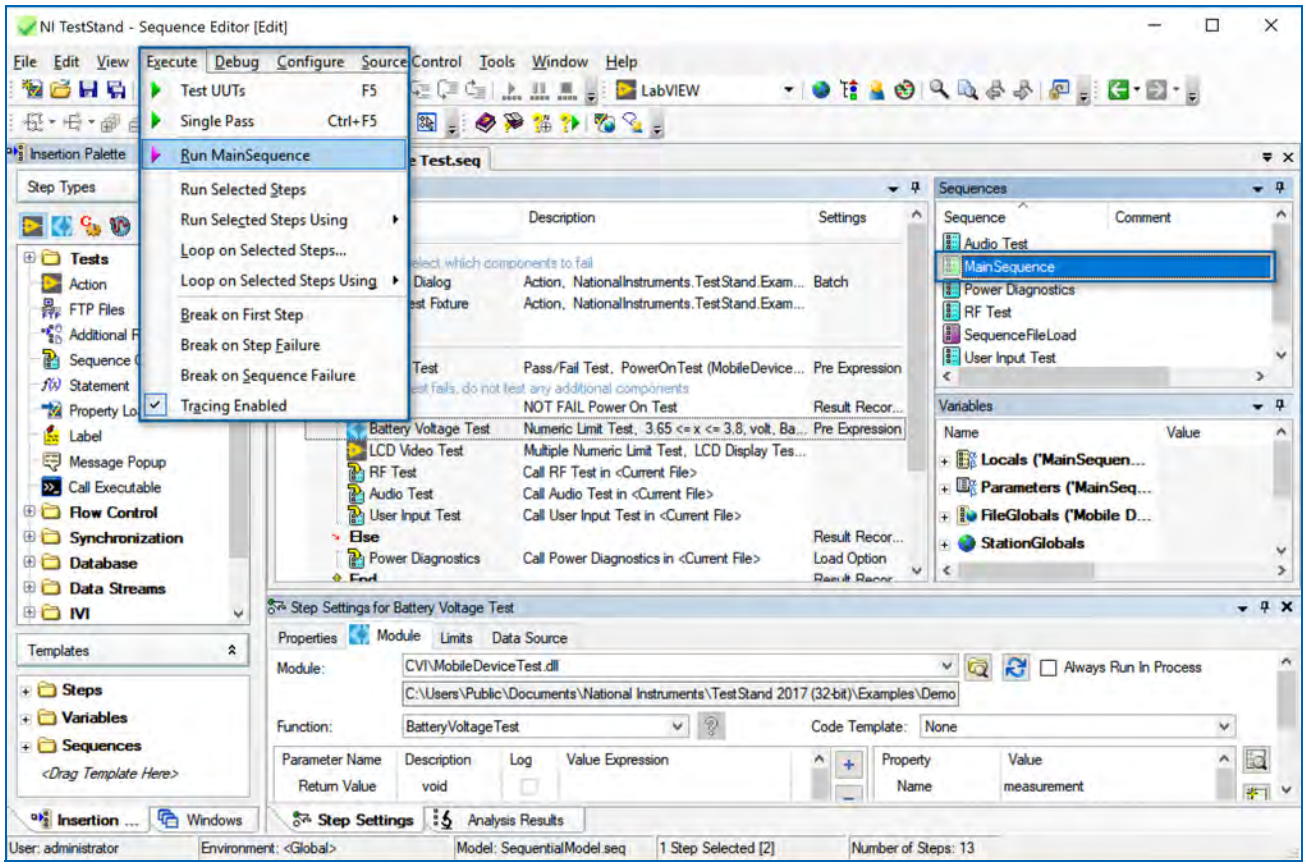
F. Executing a Test Sequence

Objective: Identify the different ways of executing a test sequence and run a test sequence using Run MainSequence command.

Execute the Selected Sequence

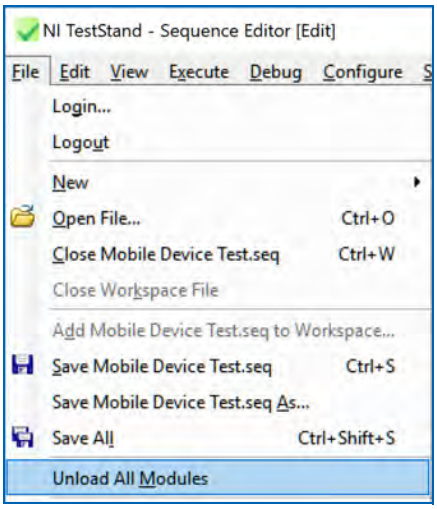
Select **Execute»Run <Name of Selected Sequence>** to execute the currently selected sequence of a sequence file. You can also execute sequences using execution entry points that contain additional functionality such as serial number management, looping, or report generation.

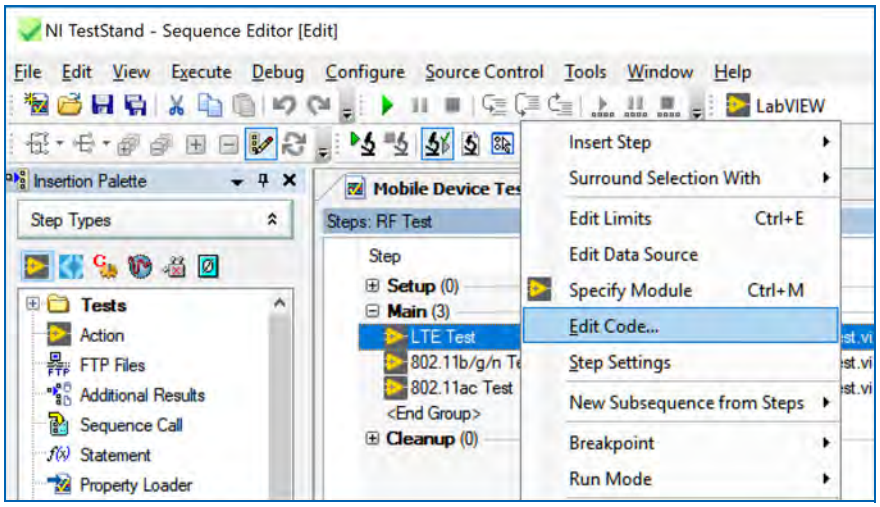
In the following image, **Test UUTs** and **Single Pass** are execution entry points.



Edit Code Modules After Execution

After you execute a sequence, you must unload the module from memory to edit it.



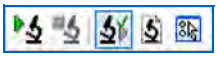


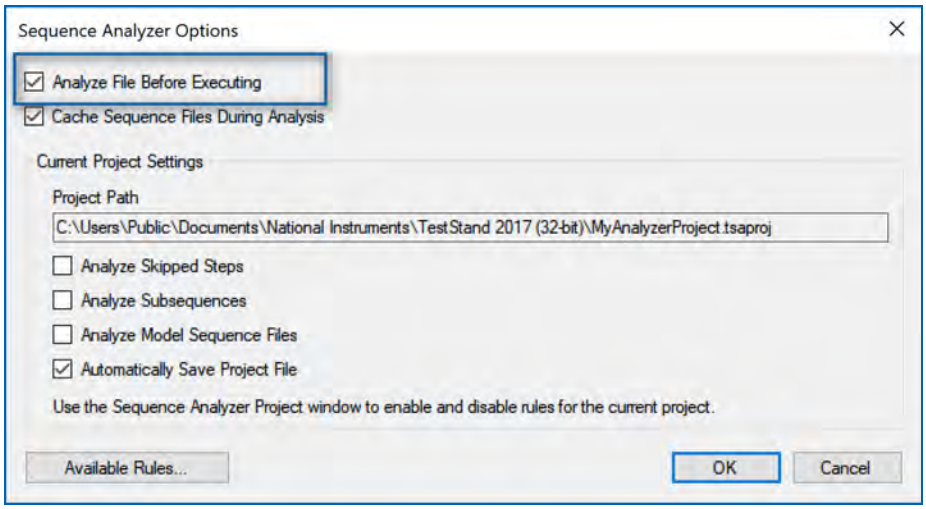
You have the following options to unload the module from memory.

- Unload all modules to release all code modules from memory.
- Right-click the step that calls the module you want to modify and choose **Edit Code** from the shortcut menu.
- Click the **Edit step** button in the module tab.

Identify Potential Errors Before Executing

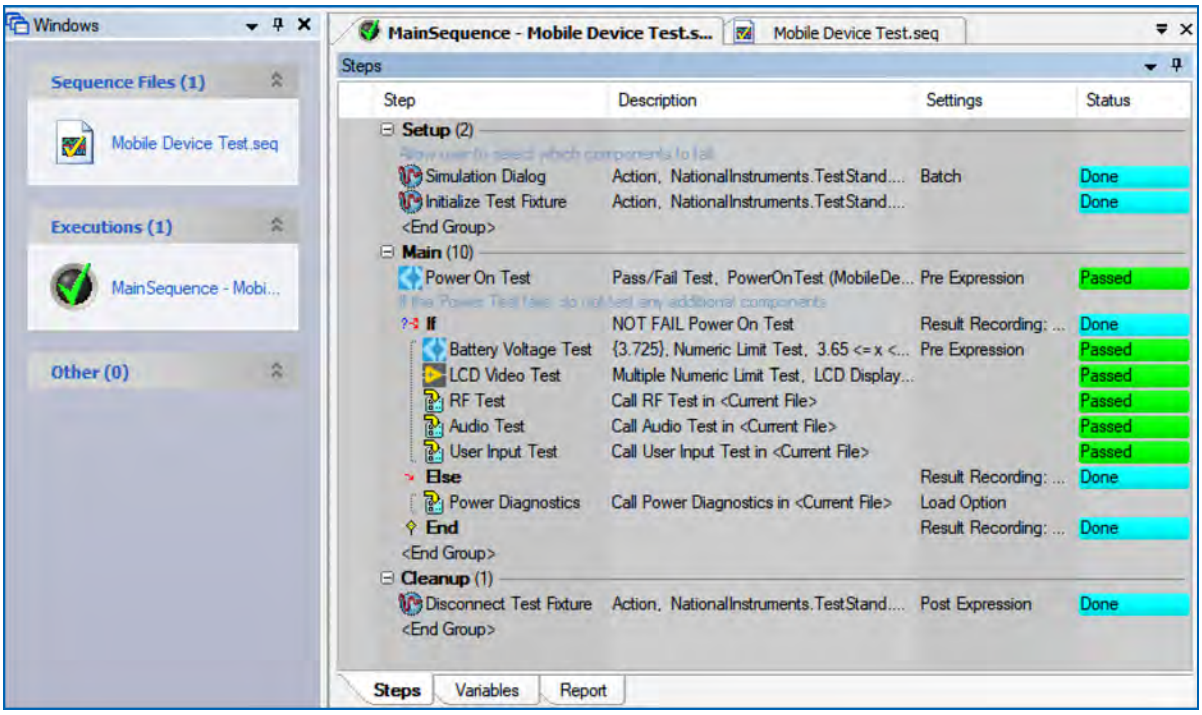
Use the TestStand Sequence Analyzer to find errors, enforce custom development guidelines you establish, and gather statistics about different types of TestStand files during development or before deployment. The built-in rules are designed to detect the most common situations that can cause run-time failures.





View Test Results During Execution

You can view test results in the Execution window during and after the test executes.

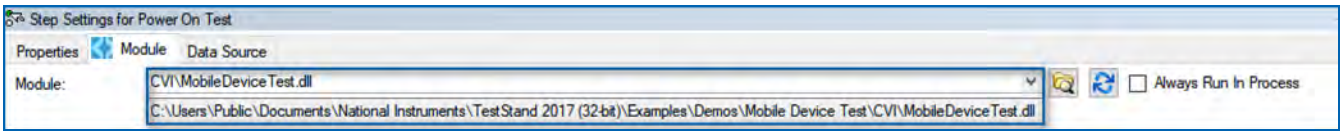


Activity: Lesson Review

1. Match each TestStand term to its definition.

Step	a. An ordered list of steps
Code Module	b. A program developed in an ADE and used by TestStand
Sequence	c. One operation or test in a system
Sequence File	d. An object containing one or more sequences

2. When you specify a relative path to a code module, how does TestStand find the file?
   a. Using the sequence file location
   b. Using paths set by the operating system
   c. Using the list of search directories



3. Which of the following tasks should be implemented in a code module?
   a. Evaluating a numeric result based on test limits
   b. Taking a measurement from a DMM
   c. Performing data analysis
   d. Adding test results to a log file

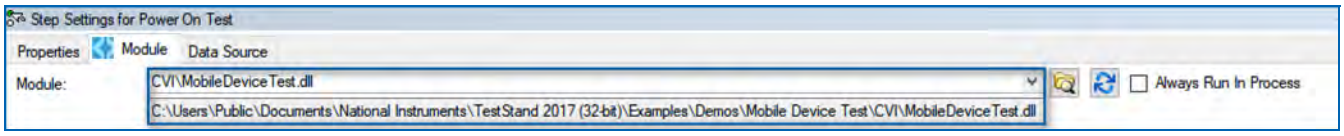
4. How can you read the output voltage measured by a code module to determine the result of a numeric limit test?
   a. In the Limits tab, set “Voltage Output” as the result.
   b. In the Parameter table, specify the value for the voltage parameter as “Step.Result.Numeric”.
   c. In the code module, ensure the voltage output parameter is named “Step Result”.

Activity: Lesson Review – Answers

1. Match each TestStand term to its definition.

Step	c.	**One operation or test in a system**
Code Module	b.	**A program developed in an ADE and used by TestStand**
Sequence	a.	**An ordered list of steps**
Sequence File	d.	**An object containing one or more sequences**

2. When you specify a relative path to a code module, how does TestStand find the file?
   a. **Using the sequence file location**
   b. Using paths set by the operating system
   c. **Using the list of search directories**



3. Which of the following tasks should be implemented in a code module?
   a. Evaluating a numeric result based on test limits
   b. **Taking a measurement from a DMM**
   c. **Performing data analysis**
   d. Adding test results to a log file

4. How can you read the output voltage measured by a code module to determine the result of a numeric limit test?
   a. In the Limits tab, set “Voltage Output” as the result.
   b. **In the Parameter table, specify the value for the voltage parameter as “Step.Result.Numeric”.**
   c. In the code module, ensure the voltage output parameter is named “Step Result”.

3 Controlling TestStand Execution

Modify a test sequence to execute differently depending on test conditions or settings.

Topics

+ Sharing Data Using Local Variables
+ Changing Execution Flow
+ Changing Execution Based on a Test Failure

Exercises

Exercise 3-1 (LabVIEW) Create and Use a Local Variable
Exercise 3-1 (LabWindows/CVI) Create and Use a Local Variable
Exercise 3-1 (TestStand Only) Create and Use a Local Variable
Exercise 3-2 Modify the Post Action Property of a Step

A. Sharing Data Using Local Variables

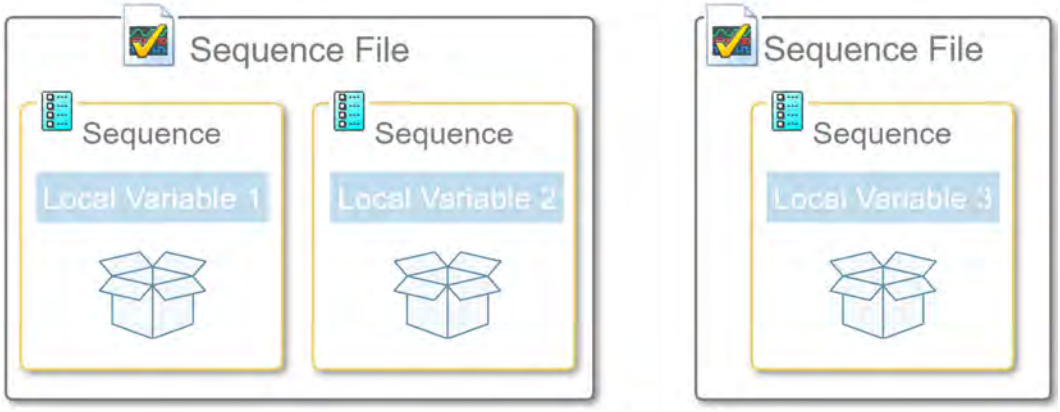
Objective: Recognize how local variables are a useful data transfer and storage tool within a test sequence.

What is a Local Variable?

Variable A user-defined container for storing data.

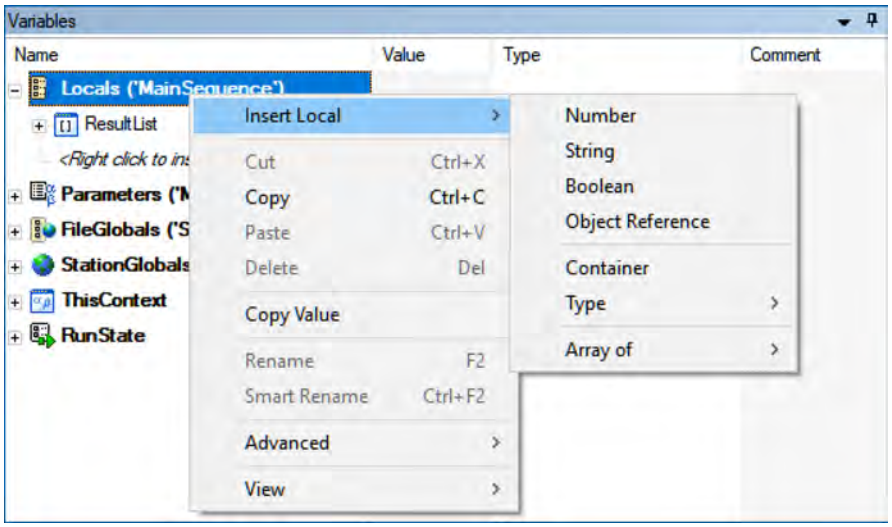
Local Variable A variable used to store data for use within a single sequence.

Use local variables when you need to transfer data among steps in a sequence.



Create a Local Variable

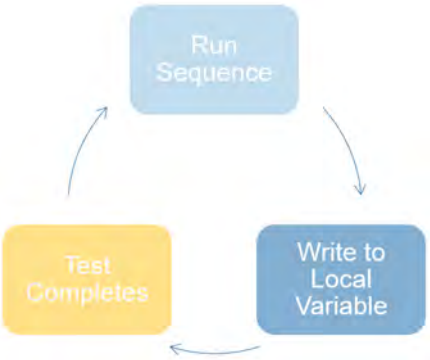
There are multiple ways to create a local variable.

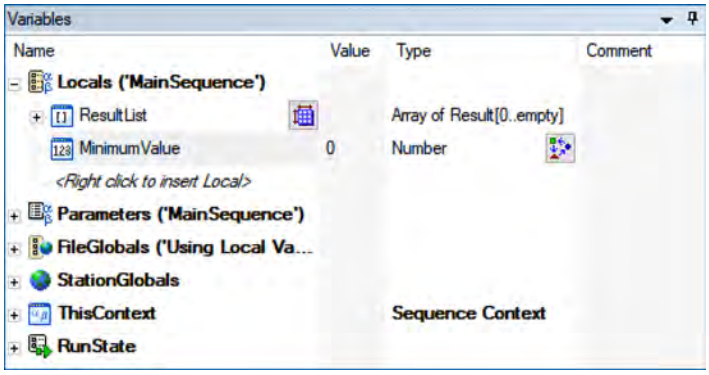


- Right-click one of the variable categories in the Variables pane and select **Insert Local** from the shortcut menu.
- Right-click one of the variable containers in the Expression Browser and select **Insert Local** from the shortcut menu.
- Right-click the variable name you want to define and select **Create** from the shortcut menu.

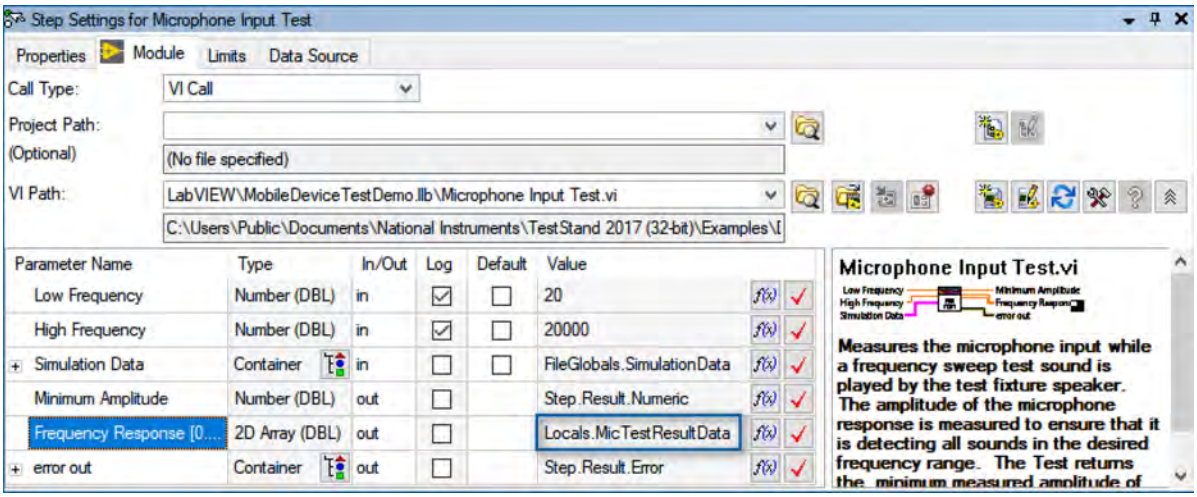
What is the Default Value for a Local Variable?

Define the default value by setting the Value column for the variable in the Variables pane. Local variables do not retain values that you write to them between executions.

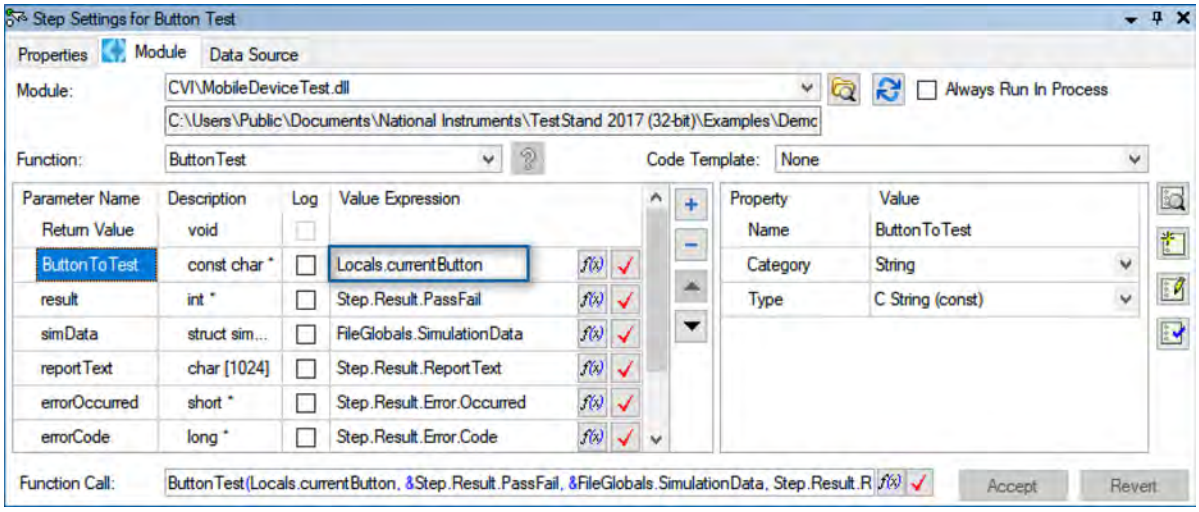




Write Data to a Local Variable



Read Data from a Local Variable



B. Changing Execution Flow

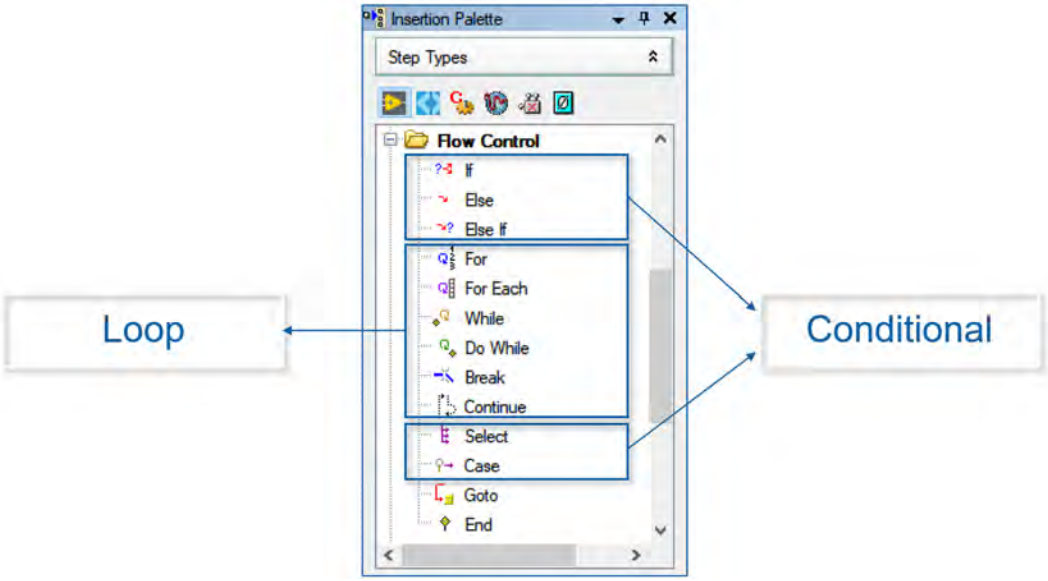
Objective: Examine techniques that control the execution of a test sequence.

Activity: Execution Flow in Code Modules

What are at least five different ways that code modules, such as LabVIEW and LabWindows/CVI, can change execution flow? Consider structures and programming techniques in your answer.

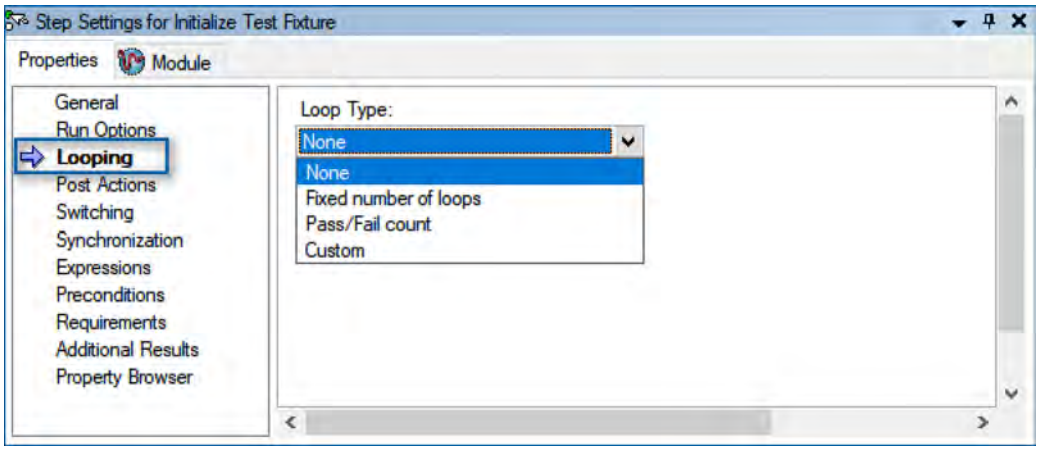
Use Flow Control Step Types

Use flow control steps to modify the default order of steps in a sequence. These conditional and looping structures are similar to conditional and looping structures found in other programming languages.



Configure a Step to Loop Automatically

In the Properties tab of the Step Settings pane, you can specify to run a single step multiple times in succession by selecting a Loop Type in the Looping Category.



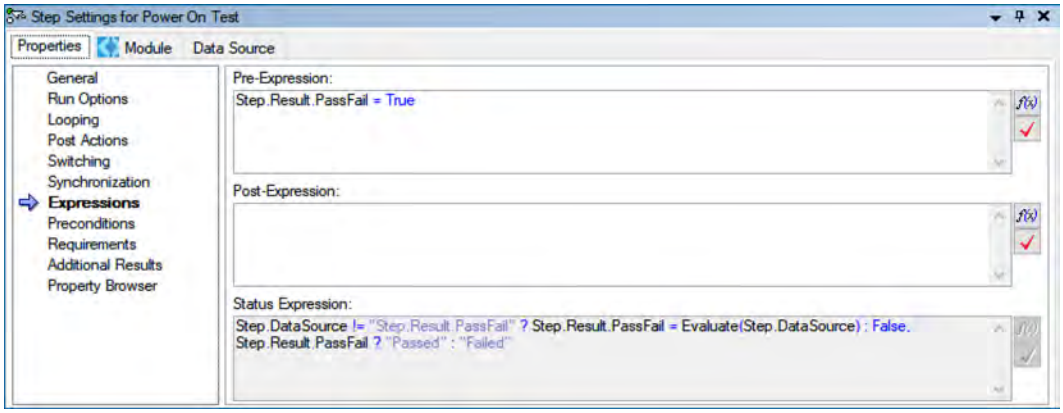
Use Expressions to Dynamically Determine Values for Flow Control

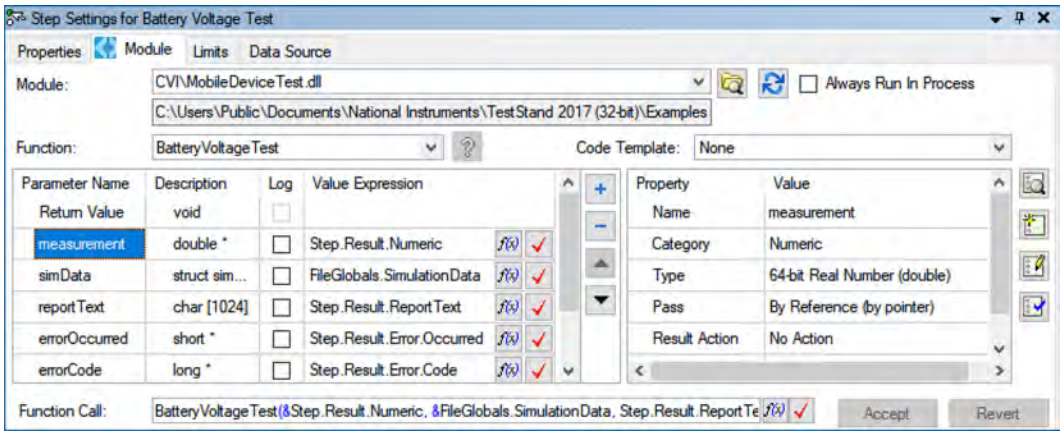
Expression Text-based syntax that TestStand uses to resolve values or perform operations.

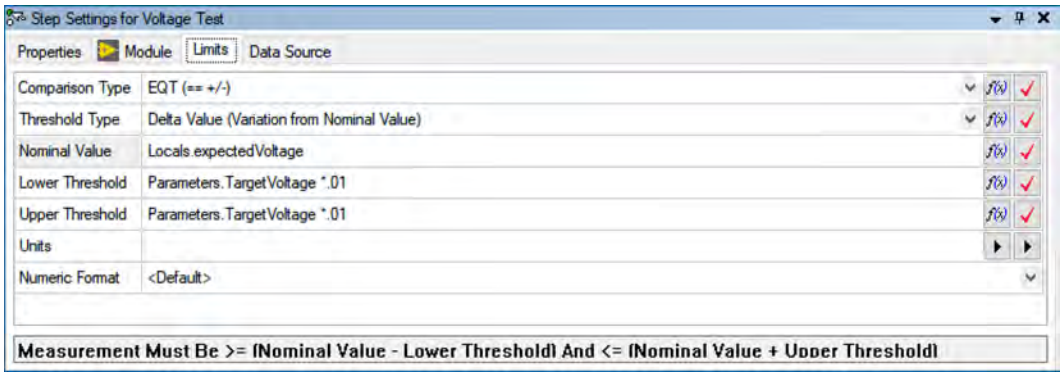
Where Can You Use Expressions?

Use expressions to programmatically access or change the data TestStand gathers as it runs a sequence. You can use expressions in the following places:

Expressions Panel | Code Module Parameters | Test Limits







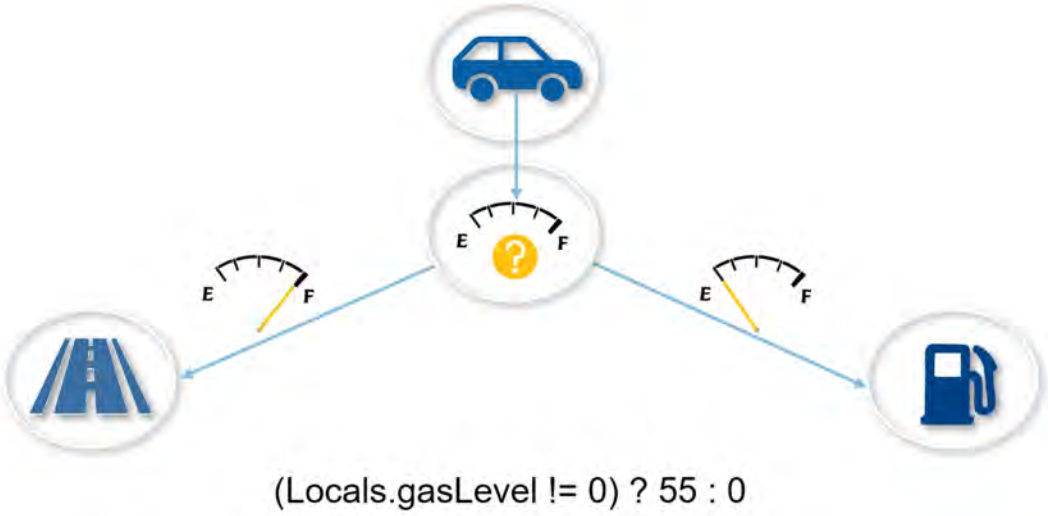
What Can an Expression Include?

Expressions can include many different elements.

Different types of expressions exist.

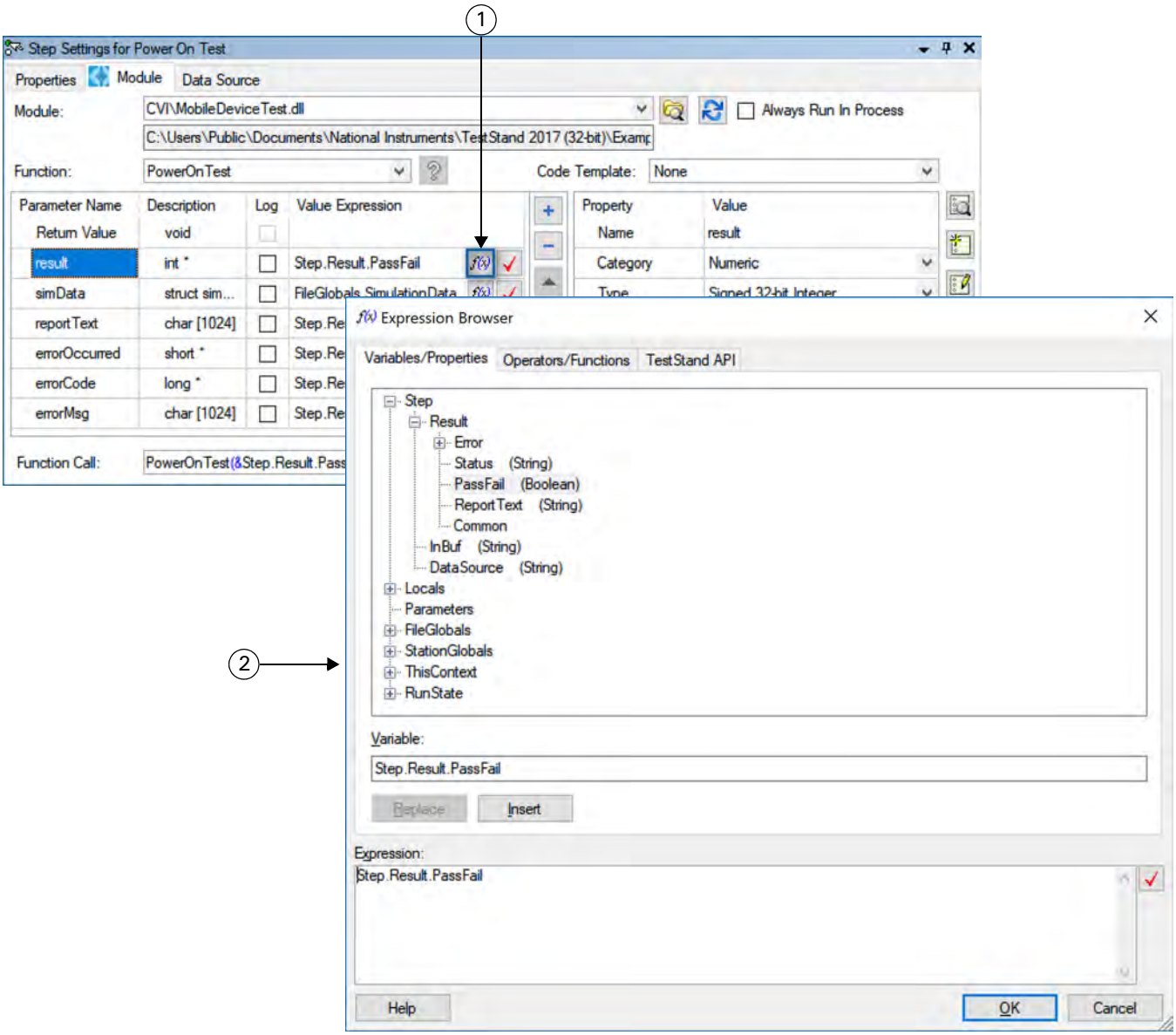
- Data expressions are evaluated at run time and return a value TestStand passes to the item for which the expression was designed. Parameter values are examples of data expressions.
- Conditional expressions evaluate to a Boolean data item and determine whether an If statement or precondition executes the steps within its block.
- General expressions do not return values but allow you to execute a statement and assign custom values. Pre- and post-expressions are examples of general expressions.

Use Expressions to Dynamically Determine Values



Use Expression Browser to Create Expressions

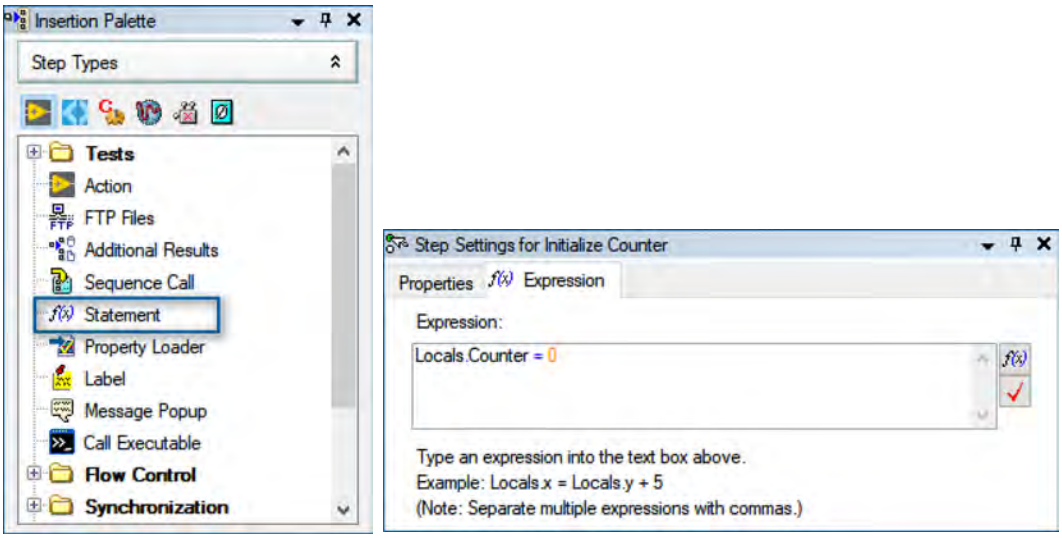
Expression Browser assists in constructing expressions. Click the Expression Browser button next to any expression text-box to launch the Expression Browser dialog box.



1 Expression Browser button
2 The Expression Browser dialog box

Use Expressions in Statement Steps

Statement steps execute expressions independently of other steps. For example, you can use an expression in a Statement step to initialize a local variable.



Use Expressions to Access Properties of a Specific Step

If you need to access a property of the same step that uses the expression, then you can simply access the Step object. For example, to obtain the status of the current step, you can specify `Step.Result.Status`.

To access properties from other steps, you can use one of three approaches. The following expressions are examples of how you can use expressions to get information about a step.

- **Step Index**—Search for the step using its position in the step group.
  `RunState.Sequence.Main[2].Result.Status`
- **Step Name**—Search for the step by its step name.
  `RunState.Sequence.Main["Test 1"].Result.Status`
- **Step ID**—Search for the step using a unique alphanumeric ID
  `RunState.Sequence.Main["ID#:mcNBH0u5EE6O40o+9ijikA"/* Unique Id of 'Test 1' */].Result.Status`

Demonstration: Expressions and Local Variables

Access step properties and set a local variable.

1. Open a blank sequence.
2. Create a Statement step and name it `My Statement Step`.
3. Create a Message Popup step that occurs after the Statement step.
4. Configure the Message Popup to display the status of the previous step.
   Set the message expression to: `"Step " + RunState.Sequence.Main ["My Statement Step"].Result.Status`
5. Check the expression for errors.
6. Run the sequence.

Exercise 3-1: (LabVIEW) Create and Use a Local Variable

Goal

Use local variables and flow control steps to create a test for DC voltage output under varying input conditions.

Scenario

The USB output port of the system is regulated such that it always produces a standard 5V. If the system cannot supply enough power to maintain this voltage, the port outputs 0V. Create a test to ensure that the DC output is always maintained at either 0V or 5V based on a supplied input voltage.

In this exercise you create steps to test the DC output voltage while the input voltage varies from 0 – 9V.

Requirements

- The DC output (pin 2) shall output 5V ± .05 (1%) when the system voltage is 5V or greater.
- The DC output shall output 0V ± .05 when the system voltage is below 5V.

Note The system voltage can be set for testing purposes by supplying power to the battery terminal (pin 0).

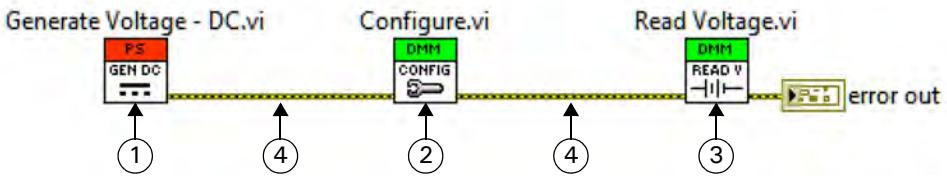
Implementation

Note If you have not completed the previous exercise, please refer to the *Course Project Exercise Setup* section of the Student Guide.

1. Create a code module in LabVIEW to implement the Voltage Test step.

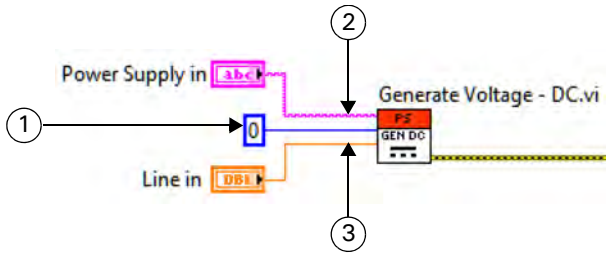
   a. In LabVIEW, create a new VI and save it as `Voltage Output Test.vi` in the `C:\Exercises\Developing Test Programs\LabVIEW\Code Modules` directory.

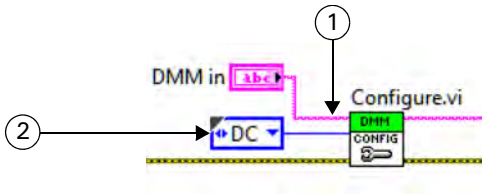
   b. Place a Generate Voltage - DC VI, Configure VI, and the Read Voltage VI on the block diagram of the Voltage Output Test VI.

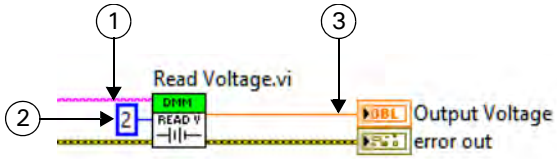


1 **Generate Voltage - DC**—Generates the supply voltage for this test. This VI is located on the Power Supply palette.
2 **Configure**—Configures the DMM to read either AC or DC voltage. Use the Configure.vi from the DMM palette.
3 **Read Voltage**—Measures the voltage on the channel you specify. This VI is located on the DMM palette.
4 Wire the error terminals to enforce execution order and propagate error information.

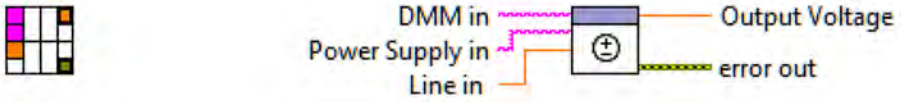
c. Configure the input and output terminals of the VIs as shown in the following table.

Generate Voltage - DC VI

1 Set the input voltage on pin 0. Wire a `0` constant to the **Channel** input. 2 Create a control for the **Power Supply in** input terminal. Right-click the input terminal and select **Create»Control** from the shortcut menu. 3 Create a control for the **Output Voltage** input terminal. Rename the control `Line in`.

Configure VI

1 Create a control for the **DMM in** input terminal. Right-click the input terminal and select **Create»Control** from the shortcut menu. 2 Specify the **Signal Type**. Right-click the input terminal and select **Create»Constant** from the shortcut menu. Select **DC** from the pull-down menu.

Read Voltage VI

1 Wire the **DMM out** terminal of the Configure VI to the **DMM in** terminal of the Read Voltage VI. 2 Wire a constant with a value of `2` to the **Channel** input. According to the requirements document, the DC output is on channel 2. 3 Create an indicator for the **Voltage** output. Right-click the terminal and select **Create»Indicator.** Rename the indicator `Output Voltage`.

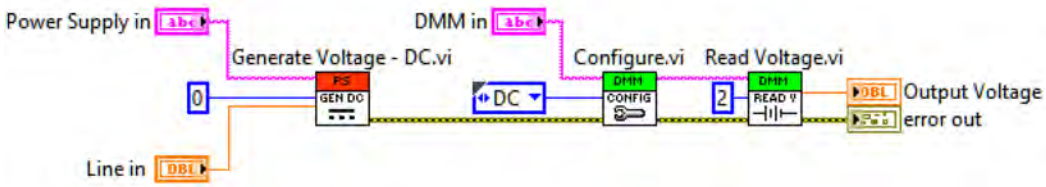
d. Configure the connector pane of the VI to match the image below.



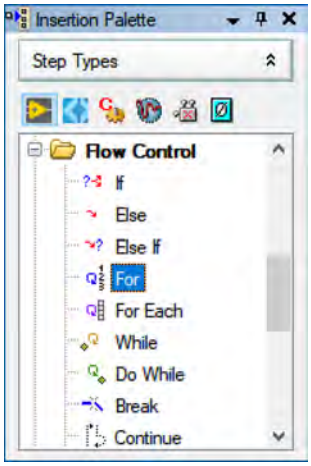
e. Add a description to document the VI, by selecting **File»VI Properties** and entering the following text in the **Documentation** category.

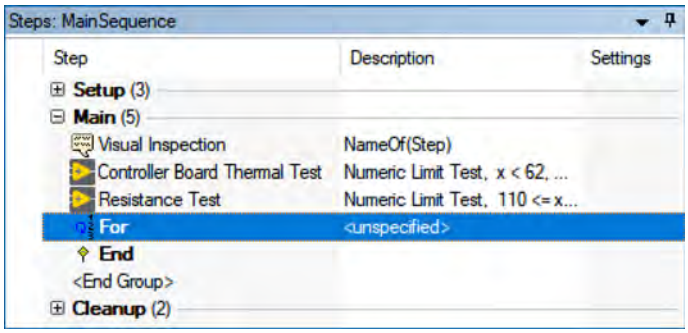
```
Measures the output DC voltage when providing an input voltage from
the battery pins.
```

f. Verify that the block diagram is similar to the one below:



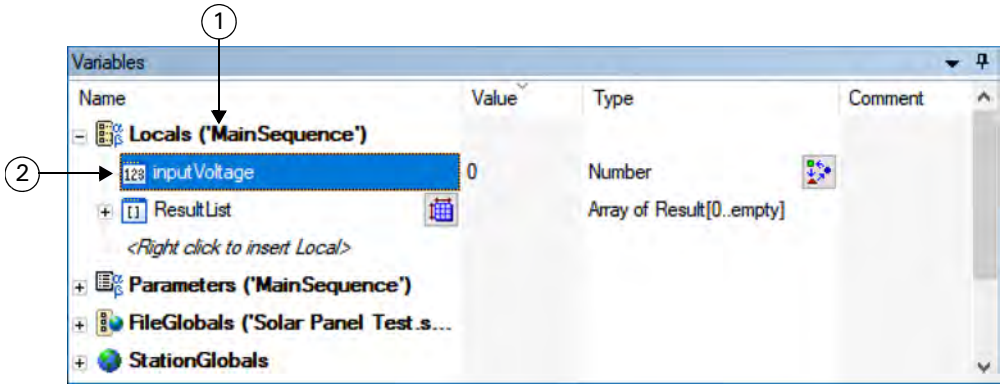
2. In TestStand, add a loop to iterate over a range of input voltages. Each iteration of the loop will test a different input condition.
   a. Add a For loop step after the Resistance Test step.





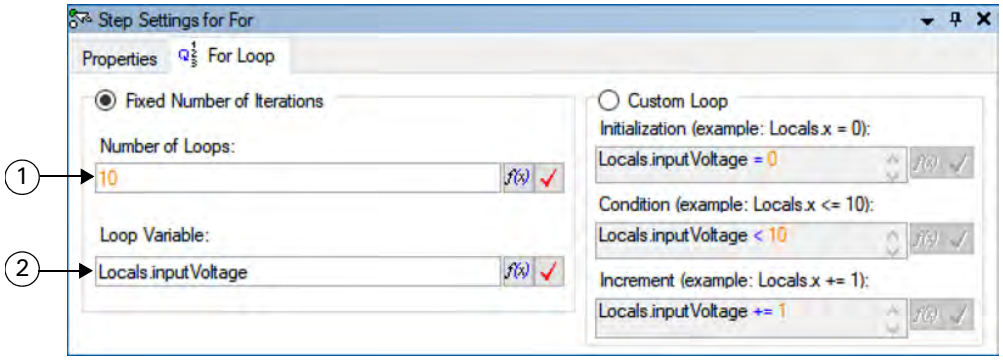
Note TestStand automatically creates an End step when you create a For step.

b. Create a new local variable to store the value of the current input voltage value.



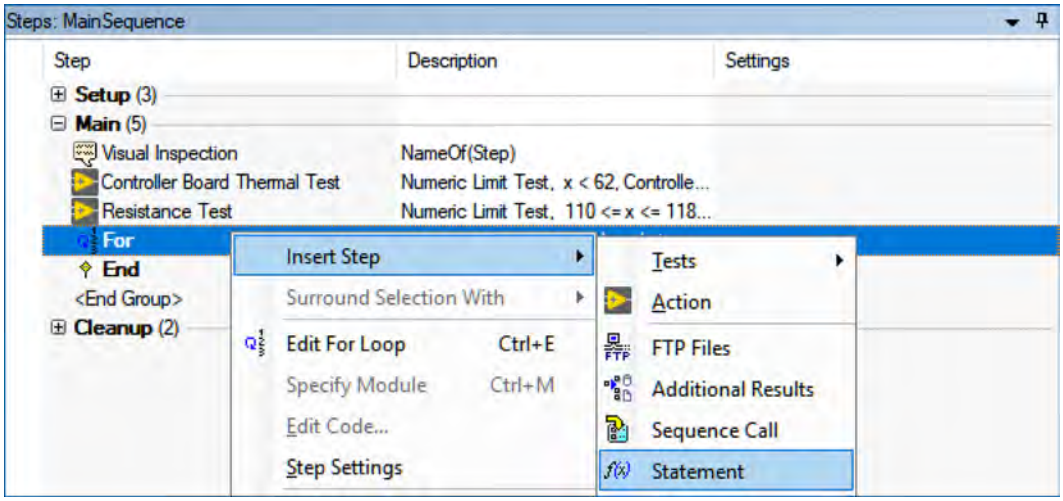
1 In the Variables pane, right-click **Locals ('MainSequence')** and select **Insert Local»Number**.
2 Name the new variable `inputVoltage`.

c. Configure the For loop to iterate the new local variable from 0 – 9 with an increment of 1.



1 Enter `10` in the **Number of Loops** field.
2 Enter `Locals.inputVoltage` in **Loop Variable** field. Locals.inputVoltage is the variable you just created.

3. Create a statement step to determine the expected voltage output of the test using a TestStand expression and configure it as outlined in the following table.



Determine Expected Result Statement Step		
Name	Determine Expected Result	(Properties tab—General category)
Expression	`Locals.expectedVoltage = (Locals.inputVoltage >= 5)?5:0`	(Expression tab) Use the conditional operator to set the expected voltage to 5 if the input voltage is 5 or greater, or to 0 if the input voltage is less than 5.

The conditional operator, `?:` , chooses one of two values depending on the Boolean value of a condition.

1 Boolean condition that determines which value the expression outputs.
2 Output value if Boolean condition is true.
3 Output value if Boolean condition is false.

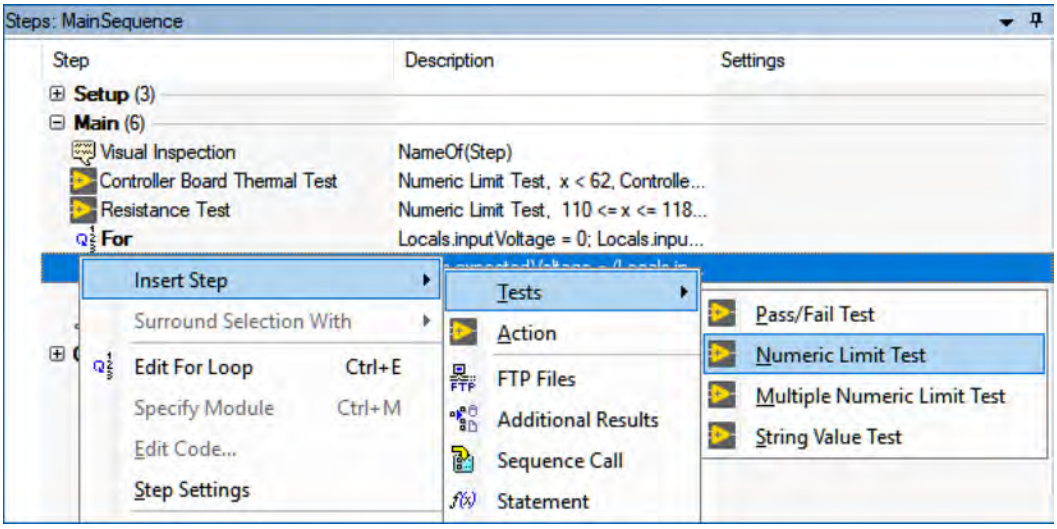
Tip The conditional operator in a TestStand expression is similar to the Select function in LabVIEW.

Figure 3-1. The Select Function in LabVIEW

a. Create the Locals.expectedVoltage variable by right-clicking `Locals.expectedVoltage` in the expression and selecting **Create "Locals.expectedVoltage"»Number** from the shortcut menu.

Note The Locals.expectedVoltage variable now appears in the Locals section of the Variables pane.

4. Create a numeric limit test step within the For loop, after the statement step, to call the Voltage Output Test code module.



   a. Configure the test step as outlined in the following table.

Voltage Test Numeric Limit Step		
Name	Voltage Test	(Properties tab—General category)
VI Path	Navigate to `C:\Exercises\Developing Test Programs\LabVIEW\Code Modules\Voltage Output Test.vi`	(Module tab) If you get a warning that the VI does not reside in a search directory, select the **Use a relative path for the file you selected** option.
Parameter configuration	**DMM in**—`"DMM"` **Power Supply in**—`"PowerSupply"` **Line in**—`Locals.inputVoltage` **Output Voltage**—`Step.Result.Numeric` **error out**—`Step.Result.Error`	(Module tab)
Limits configuration	**Comparison Type**—**EQT (= = +/-)** **Threshold Type**—**Delta Value** **Nominal Value**—`Locals.expectedVoltage` **Lower Threshold**—`0.05` **Upper Threshold**—`0.05`	(Limits tab) The limits of this step change based on the current test case. Because the Nominal Value field is an expression, you can use the expected voltage local variable to set the limit dynamically.

Test

1. Test a passing UUT.
   a. Verify that the Serial Number in the Initialize Test Fixture step is `"Golden"`.
   b. Execute the sequence using **Execute»Run MainSequence**.
   c. Observe that all iterations of the Voltage Test pass.
2. Test a Failing UUT.
   a. In the Initialize Test Fixture step, change the Serial number to `"AllFail"`. This serial number fails all tests except the Thermal Test.
   b. Run the test again using **Execute»Run MainSequence**.
   c. Verify that some iterations of the Voltage Test fail.
   d. Change the serial number back to `"Golden"`.

End of Exercise 3-1 (LabVIEW)

Exercise 3-1: (LabWindows/CVI) Create and Use a Local Variable

Goal

Use local variables and flow control steps to create a test for DC voltage output under varying input conditions.

Scenario

The USB output port of the system is regulated such that it always produces a standard 5V. If the system cannot supply enough power to maintain this voltage, the port outputs 0V. Create a test to ensure that the DC output is always maintained at either 0V or 5V based on a supplied input voltage.

In this exercise you create steps to test the DC output voltage while the input voltage varies from 0 – 9V.

Requirements

- The DC output (pin 2) shall output 5V ± .05 (1%) when the system voltage is 5V or greater.
- The DC output shall output 0V ± .05 when the system voltage is below 5V.

Note The system voltage can be set for testing purposes by supplying power to the battery terminal (pin 0).

Implementation

Note If you have not completed the previous exercise, please refer to the *Course Project Exercise Setup* section of the Student Guide.

1. Create a code module to implement the Voltage Output Test step.

   a. In the LabWindows/CVI development system, open `SolarPanelTest.h` in the SolarPanelTest project.

   b. Add a new function declaration for the DC voltage test:

SolarPanelTest.h

```
void VoltageOutputTest (char* dmm, char* powerSupply, double lineInVoltage, double* DCout,
                        Error* errorInfo);
```

   c. Open `SolarPanelTest.c` to implement the function. This function supplies a specified DC voltage (lineInVoltage), configures the DMM to read a DC voltage, and takes a DMM measurement:

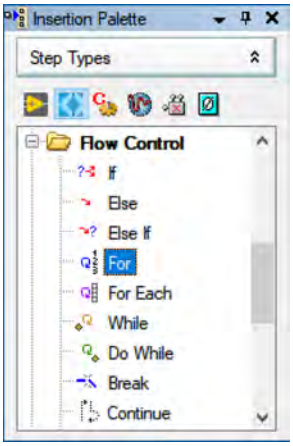
SolarPanelTest.c

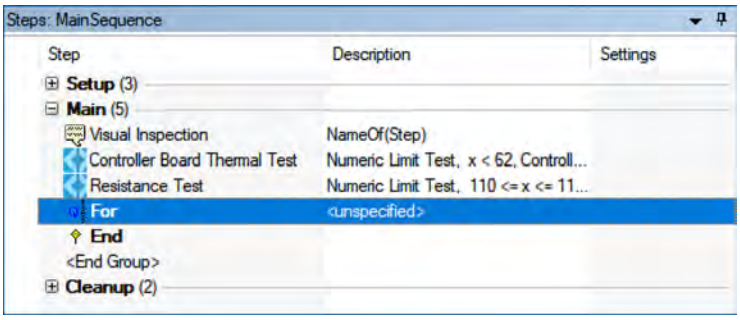
```
void VoltageOutputTest (char* dmm, char* powerSupply, double lineInVoltage, double* DCout,
                        Error* errorInfo)
{
    Sim_PowerSupply_GenerateOutputVoltageDC (powerSupply, 0, 0, lineInVoltage, errorInfo);
    Sim_Dmm_Configure (dmm, DC, errorInfo);
    Sim_Dmm_ReadVoltage (dmm, 2, 0, DCout, errorInfo);
}
```

d. In TestStand navigate to **File»Unload All modules** to unload the DLL from TestStand.

e. Rebuild the DLL.

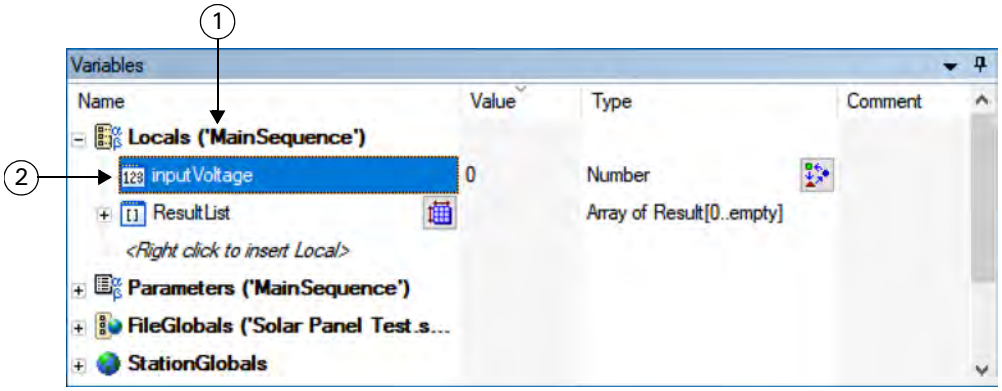
2. In TestStand, add a loop to iterate over a range of input voltages. Each iteration of the loop will test a different input condition.
   a. Add a For loop step after the Resistance Test step.





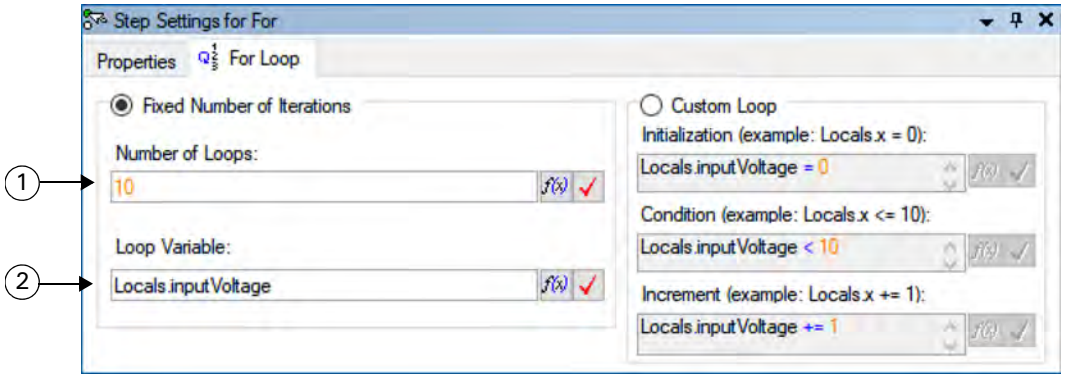
Note TestStand automatically creates an End step when you create a For step.

b. Create a new local variable to store the value of the current input voltage value.



1 In the Variables pane, right-click **Locals ('MainSequence')** and select **Insert Local»Number**.
2 Name the new variable `inputVoltage`.

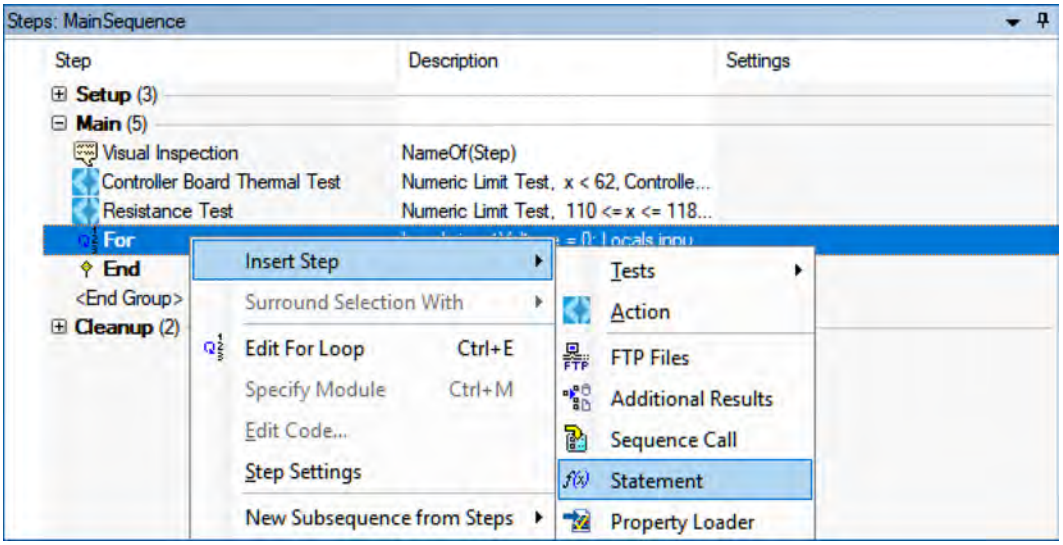
c. Configure the For loop to iterate the new local variable from 0 – 9 with an increment of 1.



1 Enter `10` in the **Number of Loops** field.
2 Enter `Locals.inputVoltage` in **Loop Variable** field. Locals.inputVoltage is the variable you just created.

3. Create a statement step to determine the expected voltage output of the test using a TestStand expression.

   a. Insert a Statement step within the For loop.



b. Configure the Statement step as outlined in the following table.

Determine Expected Result Statement Step		
Name	Determine Expected Result	(Properties tab—General category)
Expression	`Locals.expectedVoltage = (Locals.inputVoltage >= 5)?5:0`	(Expression tab) Uses the conditional operator to set the expected voltage to 5 if the input voltage is 5 or greater or 0 if the input voltage is less than 5.

The conditional operator, `?:` , chooses one of two values depending on the Boolean value of a condition.

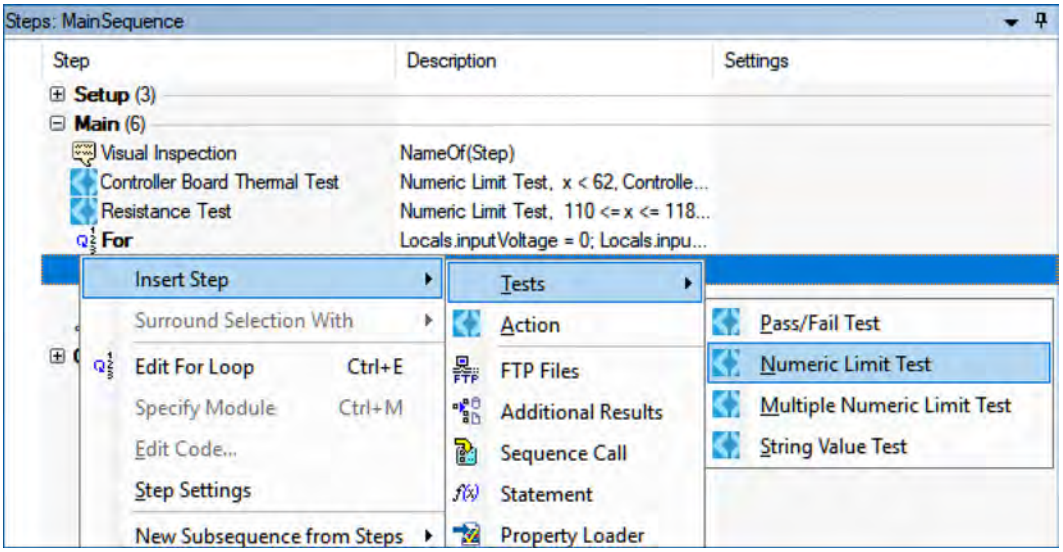
1 Boolean condition that determines which value the expression outputs.
2 Output value if Boolean condition is true.
3 Output value if Boolean condition is false.

c. Create the Locals.expectedVoltage variable by right-clicking `Locals.ExpectedVoltage` in the expression and selecting **Create "Locals.expectedVoltage"»Number** from the shortcut menu.

Note The Locals.expectedVoltage variable now appears in the Locals section of the Variables pane.

4. Create a numeric limit test within the For loop, after the statement step, to call the Voltage Output Test code module.

   a. Right-click the statement step and insert a Numeric Limit Test step.



   b. Configure the test step as outlined in the following table.

Voltage Test Numeric Limit Step		
Name	Voltage Test	(Properties tab—General category)
Module	Navigate to `C:\Exercises\Developing Test Programs\CVI\Code Modules\SolarPanelTest.dll` **Function:** `VoltageOutputTest`	(Module tab) If you get a warning that the VI does not reside in a search directory, select the **Use a relative path for the file you selected** option.
Parameter configuration	**dmm**–`"DMM"` **powerSupply**–`"PowerSupply"` **lineInVoltage**–`Locals.inputVoltage` **DCout**–`Step.Result.Numeric` **errorInfo**–`Step.Result.Error`	(Module tab)
Limits configuration	**Comparison Type**—**EQT (= = +/-)** **Threshold Type**—**Delta Value** **Nominal Value**—`Locals.expectedVoltage` **Lower Threshold**—`0.05` **Upper Threshold**—`0.05`	(Limits tab) The limits of this step change based on the current test case. Because the Nominal Value field is an expression, you can use the expected voltage local variable to set the limit dynamically.

Test

1. Test a passing UUT.
   a. Verify that the Serial Number in the Initialize Test Fixture step is `"Golden"`.
   b. Execute the sequence using **Execute»Run MainSequence**.
   c. Observe that all iterations of the Voltage Test pass.
2. Test a Failing UUT.
   a. In the Initialize Test Fixture step, change the Serial number to `"AllFail"`. This serial number fails all tests except the Thermal Test.
   b. Run the test again using **Execute»Run MainSequence**.
   c. Verify that some iterations of the Voltage Test fail.
   d. Change the serial number back to `"Golden"`.

End of Exercise 3-1 (LabWindows/CVI)

Exercise 3-1: (TestStand Only) Create and Use a Local Variable

Goal

Use local variables and flow control steps to create a test for DC voltage output under varying input conditions.

Scenario

The USB output port of the system is regulated such that it always produces a standard 5V. If the system cannot supply enough power to maintain this voltage, the port outputs 0V. Create a test to ensure that the DC output is always maintained at either 0V or 5V based on a supplied input voltage.

In this exercise you create steps to test the DC output voltage while the input voltage varies from 0 – 9V.

Requirements

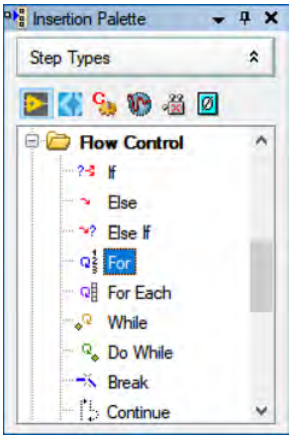
- The DC output (pin 2) shall output 5V ± .05 (1%) when the system voltage is 5V or greater.
- The DC output shall output 0V ± .05 when the system voltage is below 5V.

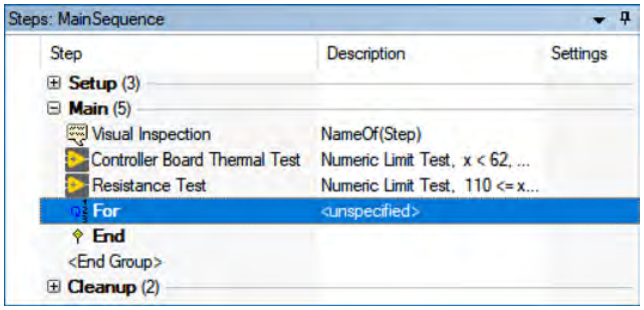
Note The system voltage can be set for testing purposes by supplying power to the battery terminal (pin 0).

Implementation

Note If you have not completed the previous exercise, please refer to the *Course Project Exercise Setup* section of the Student Guide.

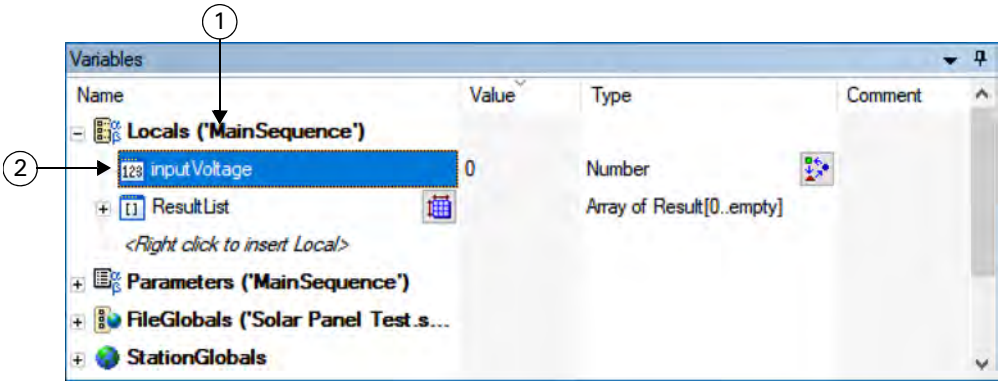
1. In TestStand, add a loop to iterate over a range of input voltages. Each iteration of the loop will test a different input condition.
   a. Add a For loop step after the Resistance Test step.





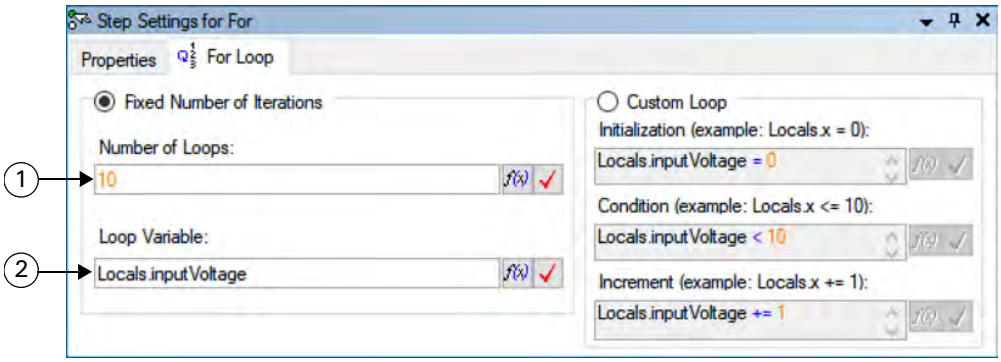
Note TestStand automatically creates an End step when you create a For step.

b. Create a new local variable to store the value of the current input voltage value.



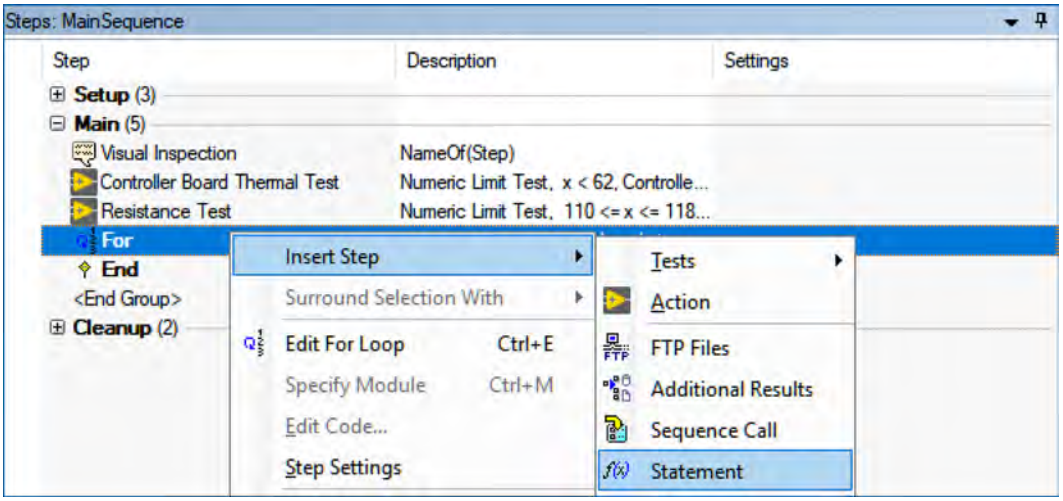
1 In the Variables pane, right-click **Locals ('MainSequence')** and select **Insert Local»Number**.
2 Name the new variable `inputVoltage`.

c. Configure the For loop to iterate the new local variable from 0 – 9 with an increment of 1.



1 Enter `10` in the **Number of Loops** field.
2 Enter `Locals.inputVoltage` in **Loop Variable** field. Locals.inputVoltage is the variable you just created.

2. Create a statement step to determine the expected voltage output of the test using a TestStand expression and configure it as outlined in the following table.



Determine Expected Result Statement Step		
Name	Determine Expected Result	(Properties tab—General category)
Expression	`Locals.expectedVoltage = (Locals.inputVoltage >= 5)?5:0`	(Expression tab) Use the conditional operator to set the expected voltage to 5 if the input voltage is 5 or greater, or to 0 if the input voltage is less than 5.

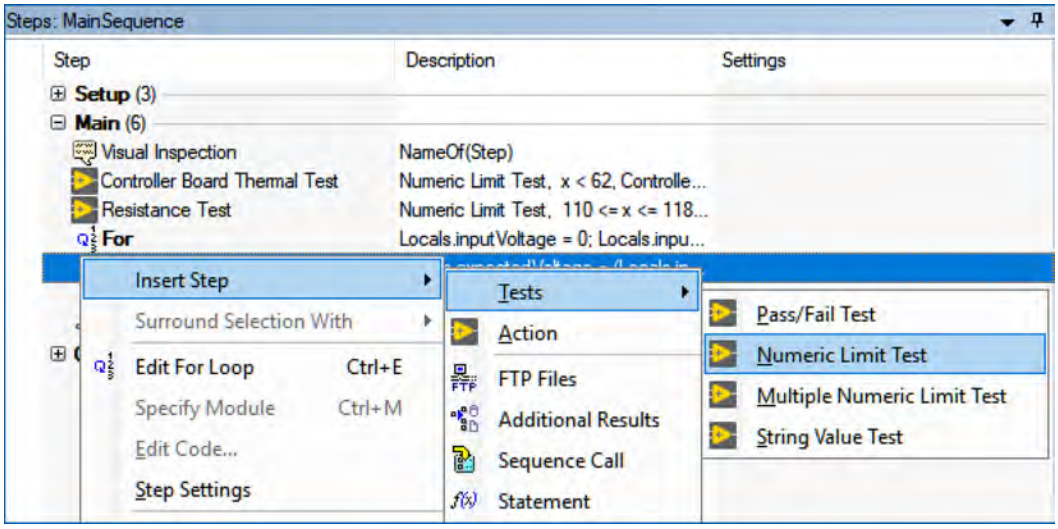
The conditional operator, `?:` , chooses one of two values depending on the Boolean value of a condition.

1 Boolean condition that determines which value the expression outputs.
2 Output value if Boolean condition is true.
3 Output value if Boolean condition is false.

a. Create the Locals.expectedVoltage variable by right-clicking `Locals.expectedVoltage` in the expression and selecting **Create "Locals.expectedVoltage"»Number** from the shortcut menu.

Note The Locals.expectedVoltage variable now appears in the Locals section of the Variables pane.

3. Create a numeric limit test step within the For loop, after the statement step, to call the Voltage Output Test code module.



a. Configure the test step as outlined in the following table.

Voltage Test Numeric Limit Step		
Name	Voltage Test	(Properties tab—General category)
VI Path	Navigate to `C:\Exercises\Developing Test Programs\TestStand Only\Code Modules\Voltage Output Test.vi`	(Module tab) If you get a warning that the VI does not reside in a search directory, select the **Use a relative path for the file you selected** option.
Parameter configuration	**DMM in**—`"DMM"` **Power Supply in**—`"PowerSupply"` **Line in**—`Locals.inputVoltage` **Output Voltage**—`Step.Result.Numeric` **error out**—`Step.Result.Error`	(Module tab)
Limits configuration	**Comparison Type**—**EQT (= = +/-)** **Threshold Type**—**Delta Value** **Nominal Value**—`Locals.expectedVoltage` **Lower Threshold**—`0.05` **Upper Threshold**—`0.05`	(Limits tab) The limits of this step change based on the current test case. Because the Nominal Value field is an expression, you can use the expected voltage local variable to set the limit dynamically.

Test

1. Test a passing UUT.
   a. Verify that the Serial Number in the Initialize Test Fixture step is `"Golden"`.
   b. Execute the sequence using **Execute»Run MainSequence**.
   c. Observe that all iterations of the Voltage Test pass.
2. Test a Failing UUT.
   a. In the Initialize Test Fixture step, change the Serial number to `"AllFail"`. This serial number fails all tests except the Thermal Test.
   b. Run the test again using **Execute»Run MainSequence**.
   c. Verify that some iterations of the Voltage Test fail.
   d. Change the serial number back to `"Golden"`.

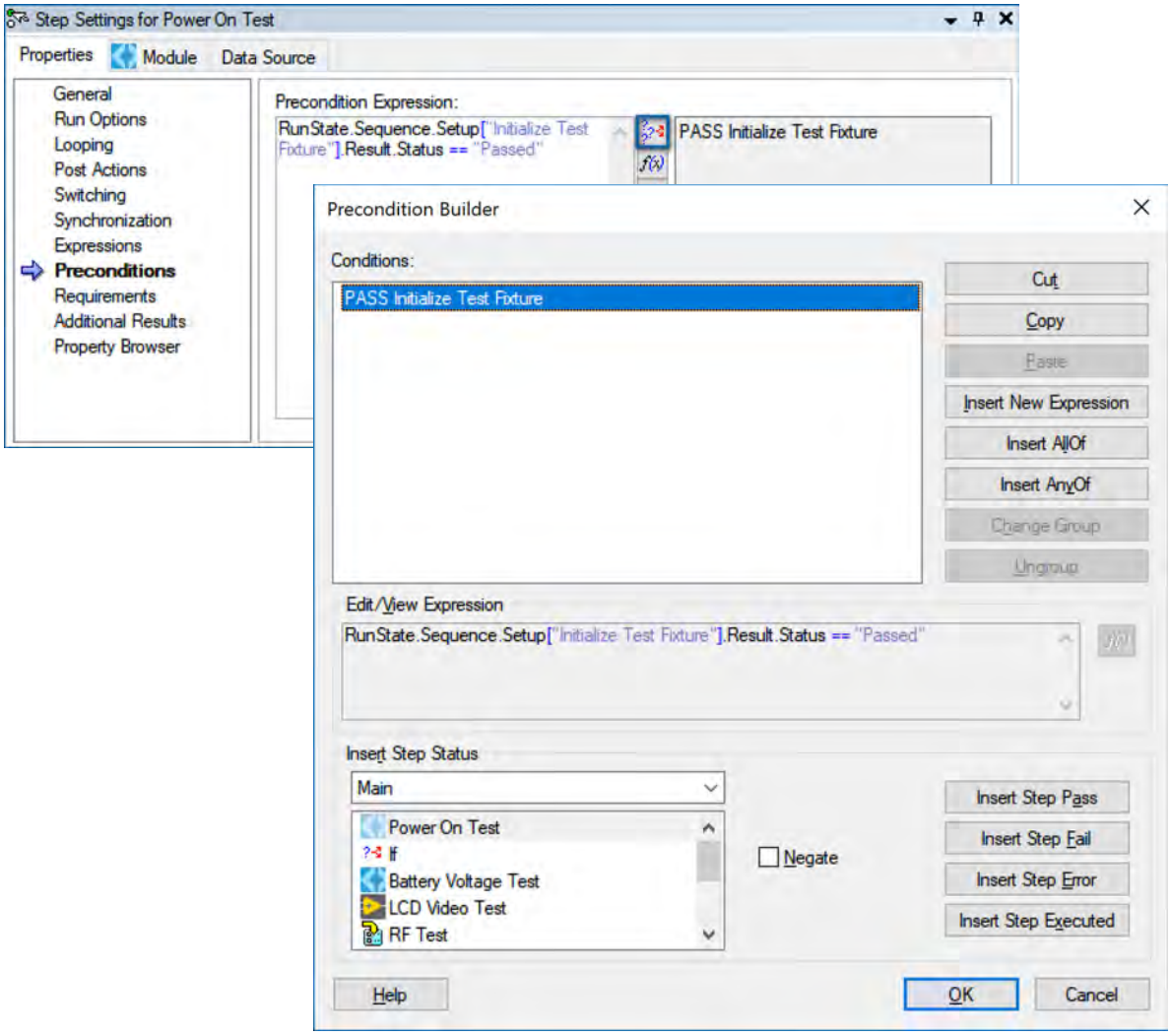
End of Exercise 3-1 (TestStand Only)

C. Changing Execution Based on a Test Failure

Objective: Use step settings to control which steps execute as a result of a step failure.

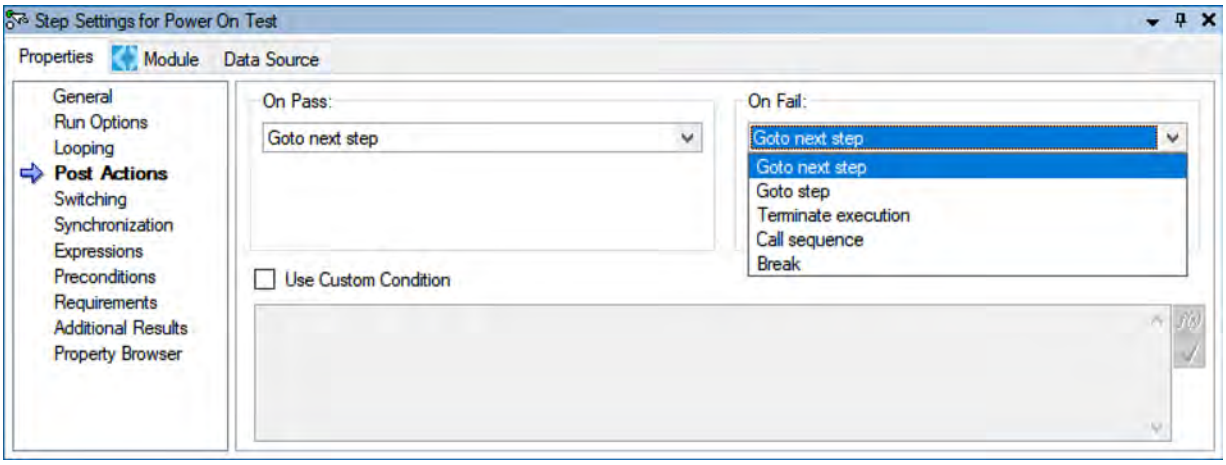
Change Execution Based on a Previous Step

The Preconditions panel of the Properties tab specifies conditions that must be True for the step to execute.



Stop Execution if a Step Fails

The Post Actions panel of the Properties tab specifies an action that occurs after a step executes.

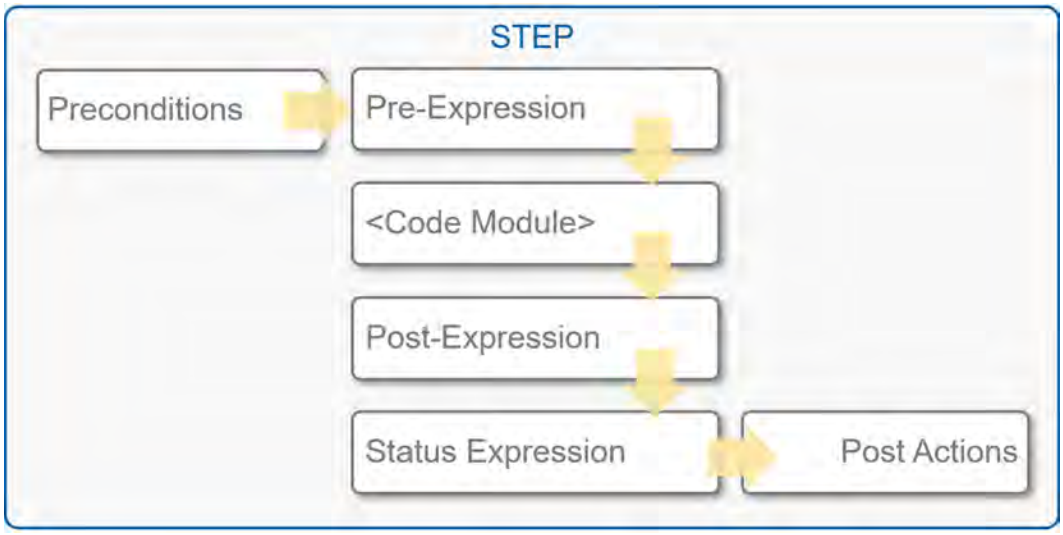


What is the Step Execution Order?

The following list outlines the order of execution for a step.

1. Preconditions
2. Pre-Expression
3. <Code Module>
4. Post-Expression
5. Status Expression
6. Post Actions

Understanding the order of execution for step settings can help you to avoid unexpected behavior. For example, if you attempt to configure a post-expression based on the status expression for the current step, you will encounter a problem because the Status Expression has not been evaluated yet.



Exercise 3-2: Modify the Post Action Property of a Step

Goal

Use Step Post Actions to terminate the sequence if a critical failure occurs.

Scenario

In the case that something is critically wrong with the UUT, the test should be terminated to prevent further damage to the UUT or the test fixture. Post actions can be used to change the execution flow based on the results of a test. For the solar panel test, failures on the visual inspection or thermal tests are critical, and should cause the sequence to terminate.

In this exercise, you add a post action to the Controller Board Thermal Test to terminate upon failure.

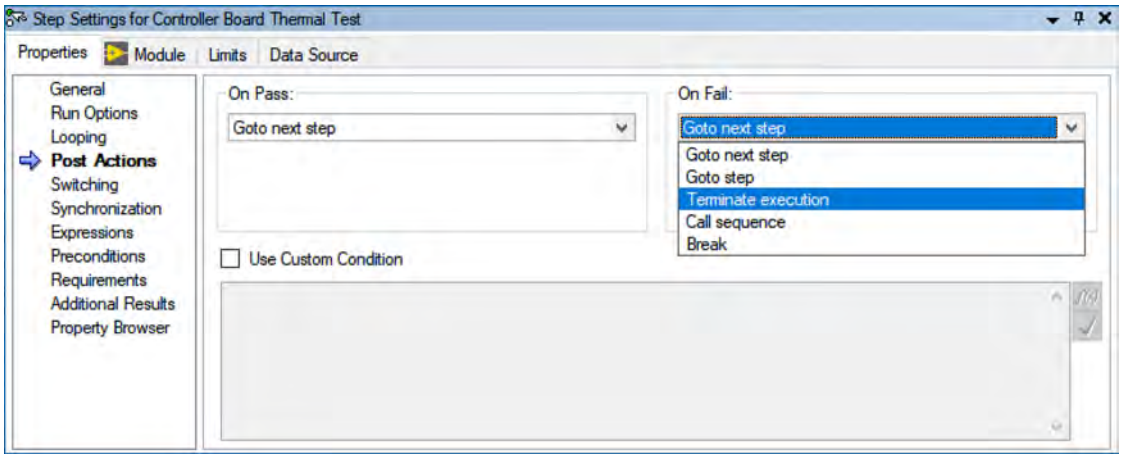
Requirements

- The test sequence shall immediately terminate if the UUT temperature exceeds 62 C.
- The test sequence shall immediately terminate if the UUT has visible defects.

Implementation

Note If you have not completed the previous exercise, please refer to the *Course Project Exercise Setup* section of the Student Guide.

1. Add a Post Action to the Controller Board Thermal Test.
   a. Select the Controller Board Thermal Test in the MainSequence window.
   b. On the Properties tab, select **Post Actions**.



c. Select **Terminate execution** in the **On Fail** drop-down menu to cause the step to terminate execution upon failure.

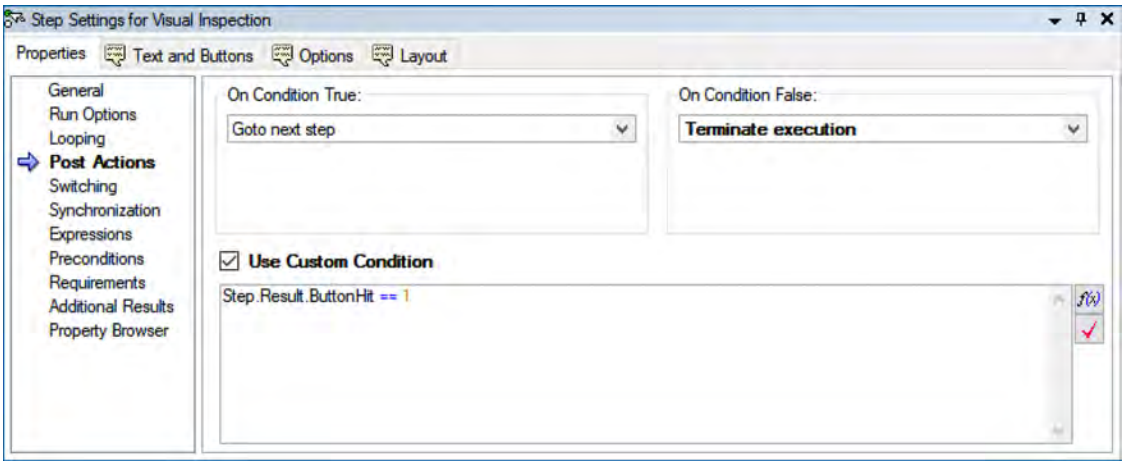
Note Cleanup steps still execute even though the Post Action terminates execution. Performing cleanup steps closes the hardware references and prevents future run-time errors.

2. Add a post action to the Visual Inspection step, so that the testing terminates if a user pressed the **Cancel** button.

   a. Select the **Visual Inspection** step in the MainSequence window.

   b. In the Properties tab, select **Post actions**.

   c. Place a checkmark in the **Use Custom Condition** box because the post action for this step is not determined by the step result.



   d. Enter the following expression in the **Use Custom Condition** field to check if the **OK** button was pressed.

   `Step.Result.ButtonHit == 1`

   The ButtonHit property is a specific property of the Message Popup step. The ButtonHit property is a one-based index of the buttons on the message popup windows.

Button Clicked	ButtonHit Value
First button in the Message Popup window	1
Second button in the Message Popup window	2
Close window button	0

   e. Select **Terminate execution** in the **On Condition False** drop-down menu.

   If a user clicks the OK button (button 1), then the condition `Step.Result.ButtonHit == 1` is true and the sequence goes to the next step. If the user hits any other button or the close window button, the condition is false and the sequence terminates.

3. Save the sequence file.

Test

1. Test a passing UUT.
   a. Execute the sequence using **Execute»Run MainSequence**.
   b. Click the **OK** button when prompted to inspect the UUT.
   c. Ensure that the sequence passes.
2. Test a visual inspection failure.
   a. Execute the sequence using **Execute»Run MainSequence.**
   b. Click **Cancel** when prompted to inspect the UUT.
   c. Ensure that the sequence terminates.
3. Test a Controller Board Thermal Test failure.
   a. In the Initialize Test Fixture step, change the Serial number to `"ThermalFail"`. This represents a UUT that is expected to fail the thermal test.
   b. Run the test again.
   c. Click the **OK** button when prompted to inspect the UUT.
   d. Ensure that the sequence terminates.
4. Change the Serial number back to `"Golden"`.

End of Exercise 3-2

Additional Resources

Topic	Location
Local Variables	`ni.com`—*How Can I Programmatically Change the Default Value of a TestStand Local Variable?*
Expressions	• *TestStand Help—Expression Operators* topic • `<TestStand Public>\Examples\Built-In Step Types\Statement Step Type\Statement Step Type.seq`

Activity: Lesson Review

1. If Locals.x has a value of 5, what behavior occurs when evaluating the following expression?

   (Locals.x == 3) ? nothing : (Locals.y = Locals.x + 2)

   a. Nothing will happen
   b. Locals.y will be set to 7
   c. Locals.x will be set to 3
   d. Locals.x will be set to 3, and locals.y will be set to 5

2. How could you configure a step to execute only if the previous step failed?

_____

_____

_____

_____

3. Arrange the following components in the order that TestStand executes them.
   - Status Expression
   - Post Action
   - Post-Expression
   - <Code Module>
   - Pre-Expression
   - Preconditions

Activity: Lesson Review – Answers

1. If Locals.x has a value of 5, what behavior occurs when evaluating the following expression?

   (Locals.x == 3) ? nothing : (Locals.y = Locals.x + 2)

   a. Nothing will happen
   b. Locals.y will be set to 7
   c. Locals.x will be set to 3
   d. Locals.x will be set to 3, and Locals.y will be set to 5

2. How could you configure a step to execute only if the previous step failed?

   Set the precondition of the step to:
   `Runstate.PreviousStep.Result.Status == "Failed"`

3. Arrange the following components in the order that TestStand executes them.
   - **Preconditions**
   - **Pre-Expression**
   - **<Code Module>**
   - **Post-Expression**
   - **Status Expression**
   - **Post Action**