NI TestStand™ II: Customization Course Manual

Course Software Version 4.0
July 2007 Edition
Part Number 323038E-01

Contents

Lesson 5
User Interfaces

Lesson 6
Customizing Database Interaction

Lesson 7
Design Considerations

Appendix A
Additional Information and Resources

Course Evaluation

Lesson 3: Custom Steps

TOPICS

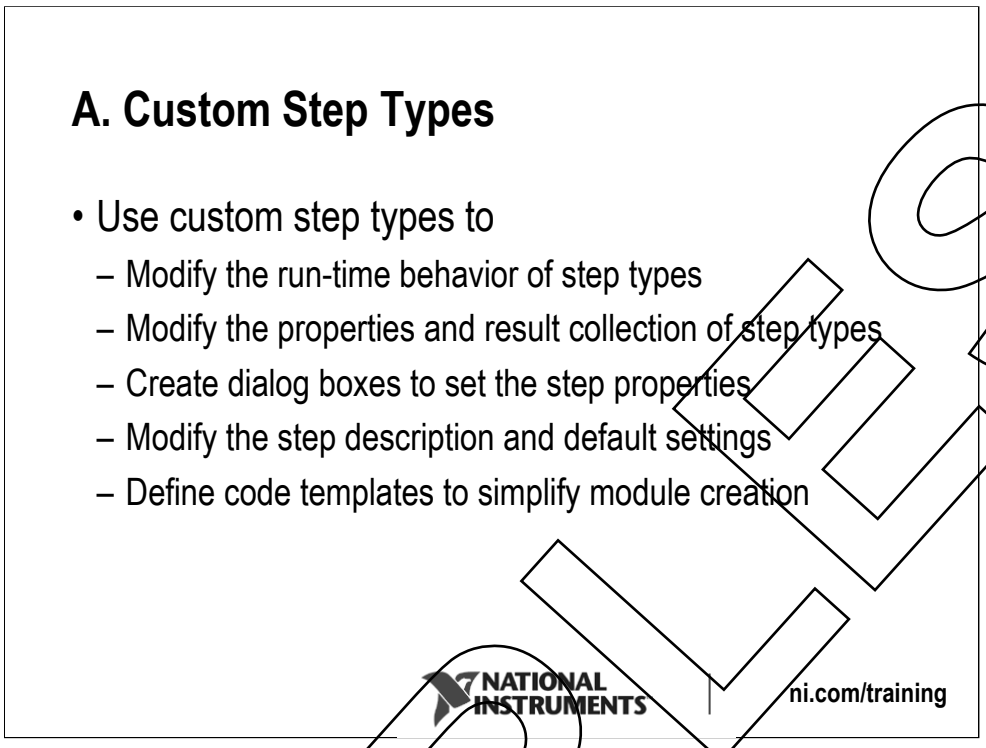
A. Custom Step Types
B. Step Templates

NATIONAL INSTRUMENTS | ni.com/training

Custom steps are useful tools you can provide to test developers along with the test framework. Some custom steps interact with the process model or user interface directly, so the developer can tie into the framework in new ways. There are two varieties of custom steps:

- Custom step types are completely new step types you can use to customize most aspects of a step, including data storage, run-time behavior, and default settings.
- Step templates are existing step types with a customized set of default settings.

This lesson describes custom step types and compares them to step templates.

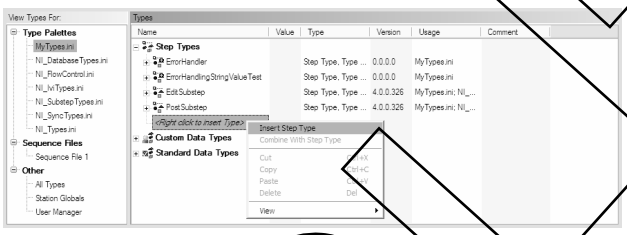


A. Custom Step Types

You can use custom step types to influence the behavior of a step much more than normal step settings. Using custom step types, you can modify the run-time behavior of the step to perform additional operations as the step runs. You can also modify the data that the step stores and the results that the step creates to log to databases or reports. Refer to Lesson 4, *Process Models,* for more information on result collection. Custom step types can define their own configuration dialog boxes so the test developer can configure the settings of the step. You can modify the step description, menu location, icon, and other appearance settings for a custom step type. Custom step types may have predefined settings for each of the standard step properties. You can disable built-in properties so that the test developer cannot modify them. You can also define code templates from a custom step type, so the test developer has a starting point when creating code modules.

Custom Step Types – Creation

Create custom step types from the Types pane



NATIONAL INSTRUMENTS | ni.com/training

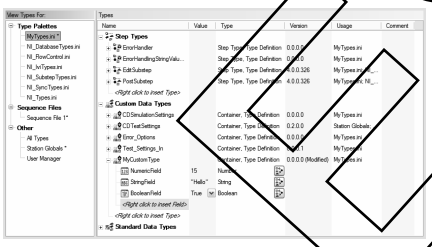
Custom Step Types – Creation

To create a custom step type, right-click within the Type palette and select **Insert Step Type**.

You may want to use an existing step type as a starting point for a custom step type. To use an existing step type, copy a step type from one of the built-in palettes and place the step type in your Type palette. You must rename the step type before you can edit it. The easiest way to rename the step type is to paste two copies of the step type into your Type palette. The second copy automatically renames itself to avoid a conflict. Delete the first copy. Rename and customize the second copy.

Custom Step Types – Locations

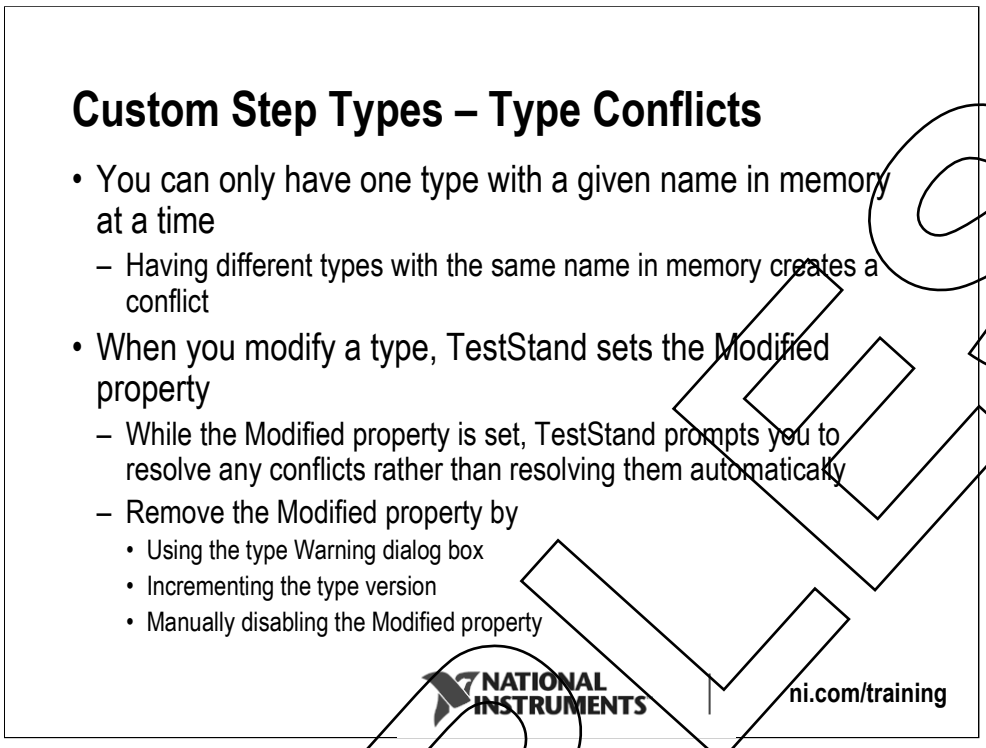
- Type palettes
  - MyTypes.ini is most common
  - Best for types used by multiple sequence files
- Sequence files
  - Best for types used only within a single sequence file
  - Sequence files maintain a copy of types used from a type palette in case the type palette is missing



NATIONAL INSTRUMENTS | ni.com/training

Custom Step Types – Locations

Store custom step types in a Type palette, such as `MyTypes.ini`, or directly in the sequence file where you use them. Store the types in a Type palette if you plan to use them in multiple sequence files. When you use a type from a Type palette, the sequence file automatically maintains a copy
of the type in case the Type palette is missing or the type is removed from the palette. The version settings resolve any differences between the sequence file version of the type and the Type palette version.



Custom Step Types – Type Conflicts

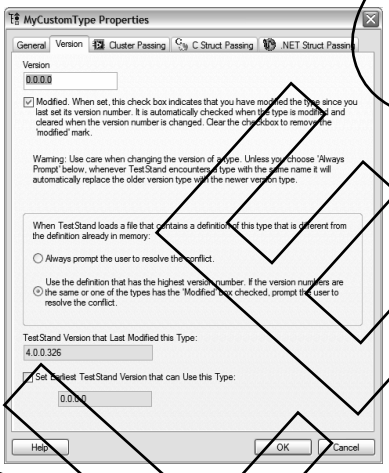
TestStand permits only one definition for each uniquely named type in memory. Although the type can appear in multiple files, only one underlying definition of the type exists in memory. If you modify the type in one file, the type updates in all files.

When you modify a type, TestStand enables the built-in Modified property for the type. When you complete your modifications to the type, disable the Modified property by incrementing the version of the type on the Step Type Properties dialog box or the Type Properties dialog box. TestStand cannot automatically resolve type conflicts until you have disabled the Modified property. If you fail to remove the Modified property, TestStand forces you to resolve any type conflicts manually by using the Type Conflict in File dialog box.

If you load a file that contains a type definition and another type definition of the same name already exists in memory, TestStand verifies that the two type definitions are identical. When TestStand compares two types with the same name, TestStand also compares all the built-in and custom subproperties in the types. If the types are not identical, TestStand attempts to resolve this conflict.

Custom Step Types – Versioning

- TestStand automatically resolves type conflicts by selecting the type with the higher version number if
  - You load two types with the same name
  - The types are different
  - Neither type has the Modified property set and the types have different versions



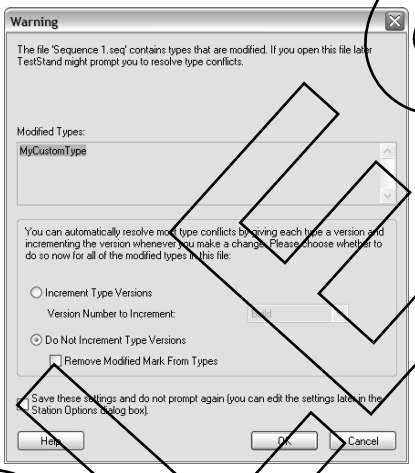
NATIONAL INSTRUMENTS | ni.com/training

Custom Step Types – Versioning

TestStand uses the version number to determine whether to load a type from a file when the type is already in memory and the version of a type to use when the version numbers are different. TestStand automatically selects the type with the greater version number if the Modified property for both types is disabled and each type definition does not specifically require a prompt to resolve the type conflict. If TestStand cannot automatically determine which type to use, or an execution is running and the type that TestStand wants to use is located in the file being loaded, TestStand informs you of the conflict through the Type Conflict in File dialog box. You can resolve the conflict using the Type Conflict in File dialog box.

Custom Step Types – Type Warning Dialog Box

- The type Warning dialog box displays when
  - You make changes to a type
  - You do not remove the modified property
  - You save the sequence file or type library
- You can configure the dialog box or station options to automatically increment the version



NATIONAL INSTRUMENTS | ni.com/training

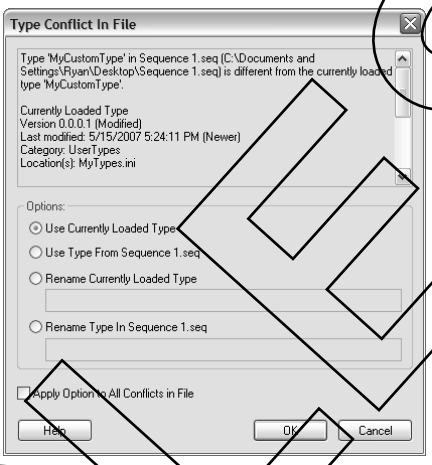
Custom Step Types – Type Warning Dialog Box

The type Warning dialog box displays when you attempt to perform an operation that might cause type conflicts that TestStand cannot automatically resolve. If you make changes to a type, leave the Modified property set, and save the sequence file or type library, the next time you open the library, TestStand recognizes that the type may conflict with existing types. However, TestStand cannot resolve the conflict because the Modified property is set.

You can avoid this dialog box by manually incrementing the type version after you modify a type. You can also disable the dialog box. You can permit TestStand to automatically increment versions by enabling the save option in the type Warning dialog box or on the Preferences tab of the Station Options dialog box.

Custom Step Types – Type Conflict In File Dialog Box

- The Type Conflict In File dialog box displays when
  - You attempt to load two types with the same name and version but different definitions
  - You attempt to load two types with the same name and version when one type has the modified property currently set



NATIONAL INSTRUMENTS | ni.com/training

Custom Step Types – Type Conflict In File Dialog Box

TestStand launches the Type Conflict In File dialog box if you load a file that contains a type definition while another type definition of the same name already exists in memory and TestStand verifies that the two type definitions are not identical. Specifically, if the type definitions are not identical and the type versions are identical, or if at least one type is marked as modified, TestStand uses the Type Conflict in File dialog box to inform you of the conflict. The textbox displays information about the conflicting type.

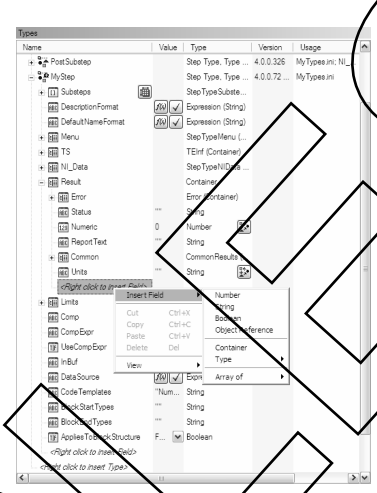
Use this dialog box to select which definition to use or rename one definition so they can both exist in memory.

The Type Conflict in File dialog box contains the following options:

- **Use Currently Loaded Type**—Converts the instance of the type in the file to use the currently loaded type. The newly loaded file is marked as modified.
- **Use Type From <Sequence File>**—Converts the instance of the type in memory to use the type from the specified file. All previously loaded files that contain the type and instances of the type are marked as modified.
- **Rename Currently Loaded Type**—Renames the type in memory and converts instances of the type in memory to use the new name. All previously loaded files that contain the type and instance of the type are marked as modified.
- **Rename Type in <Sequence File>**—Renames the type in the file and converts instances of the type in the file to use the new name. The newly loaded file is marked as modified.
- **Apply Option to All Conflicts in File**—Applies the selected option to all conflicts in the sequence file.

Custom Step Types – Adding Properties

- Step types can use custom properties to store data
- Result container
  - Important results of the step
  - Data intended for the report or database
- Root of step or other subfolders
  - Temporary data items
  - Configuration settings
  - Minor results
  - Can add to results programmatically, if necessary



NATIONAL INSTRUMENTS | ni.com/training

Custom Step Types – Adding Properties

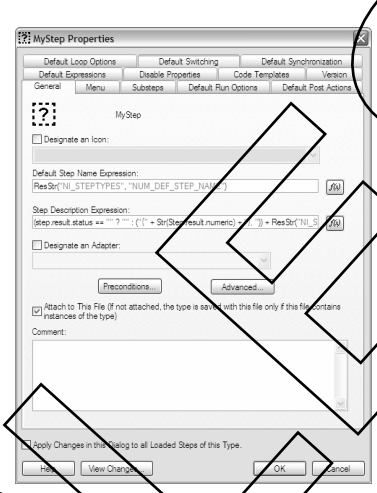
You can add properties to a custom step type to store data. To add a property, right-click inside the step type and select **Insert Field**. Properties added to any folder except the Results folder are generally used for internal data storage. Examples of properties you might create outside the Result container include:

- temporary data items to store data from one substep to the next.
- configuration settings to be set by an edit substep.
- minor results of the step that you may not want to include in the report or the database.

Reserve the Result container for important results of the step that you always intend to include in the report or database. You can programmatically transfer properties into the result container in order to include optional results in the report or database. Refer to Lesson 4, *Process Models,* for more information on results and adding items to reports.

Custom Step Types – Step Type Settings

- Right-click the step and select **Properties** to edit the step type settings
- Step type settings allow you to
  - Define
    - Menu item name and location
    - Step description
    - Substeps
    - Code templates
  - Set default values for the step settings
  - Disable step settings to make the default values permanent



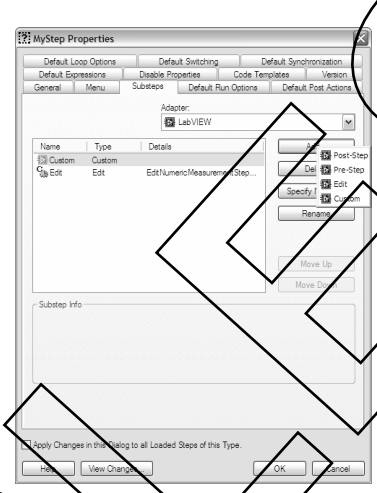
NATIONAL INSTRUMENTS | ni.com/training

Custom Step Types – Step Type Settings

Step types have a number of settings you can configure to customize the type. Right-click a step and select **Properties** to open the step Properties dialog box where you can specify step type settings. Use the step type settings to specify the menu location, specify the icon and step description as well as configure code module settings, define substeps, and create templates. You can use the step type settings to set default values for any of the standard step properties. You can also disable any of the standard properties so that a test developer cannot override the default values.

Custom Step Types – Using Substeps

- Substeps are code modules that are called when certain events occur on the step
- Substeps cannot use a relative path from a sequence file
  - Store code modules in a search path directory
  - `<TestStand>\Components\User\StepTypes` is the standard location
- You can use the substep modules from built-in step types
  - `<TestStand>\Components\NI\StepTypes`



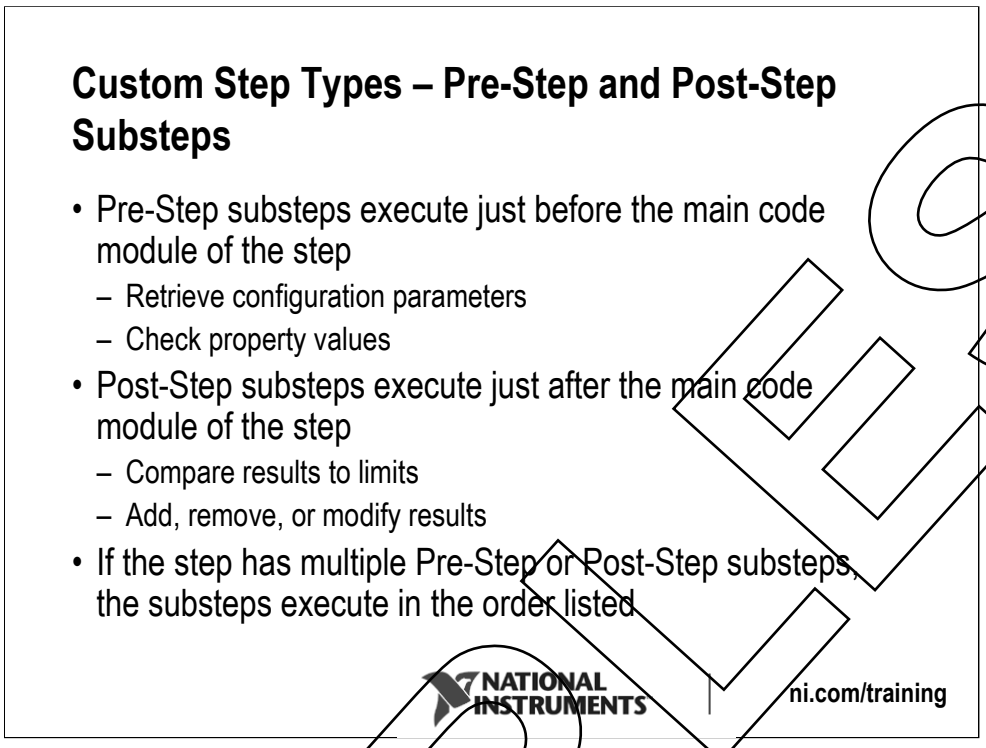
NATIONAL INSTRUMENTS | ni.com/training

Custom Step Types – Using Substeps

You can use a substep to call a code module when certain events occur for an instance of the custom step type. There are four types of substeps. Pre-Step and Post-Step substeps influence the run-time behavior of the step. Edit substeps define configuration options for the step. Custom substeps are called when you create a new step or when you specify a special event from a user interface.

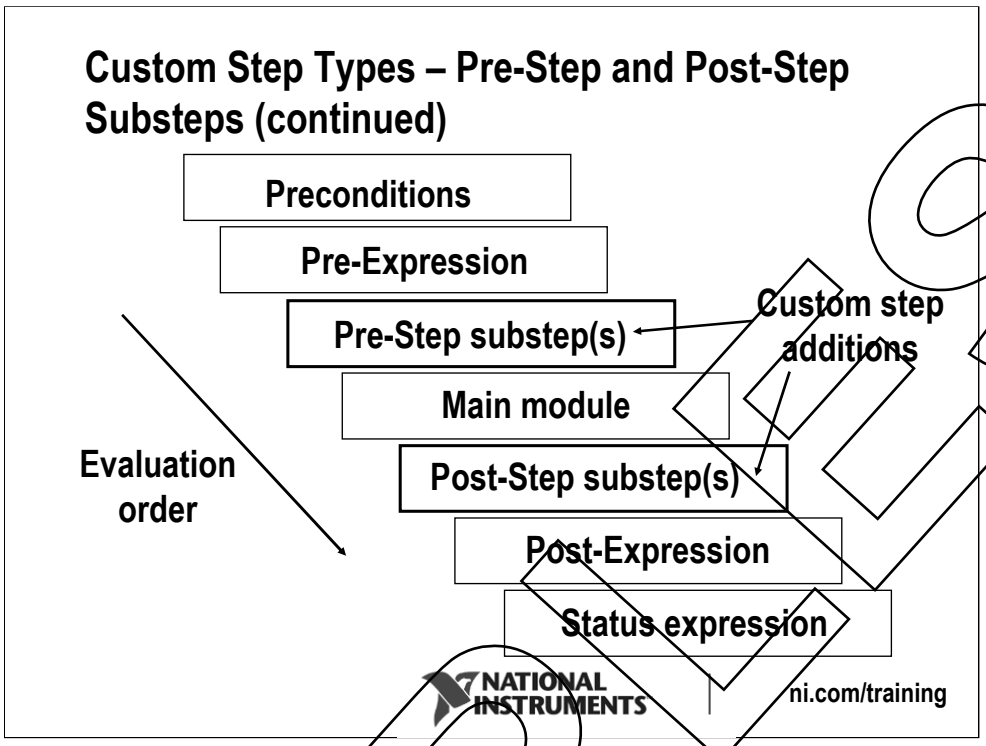
Substeps can only call code modules and cannot use the sequence adapter. Therefore, you must write any substeps in external languages. You can reuse code for existing substeps by locating the appropriate code module in the `<TestStand>\Components\NI\StepTypes` folder.

Substeps cannot use a relative path to a sequence file to locate a code module because the step type can be used in any sequence file. Therefore, place code modules for substeps into a search directory. The `<TestStand>\Components\User\StepTypes` directory is automatically configured as a search directory and is set aside for supporting files for custom step types.



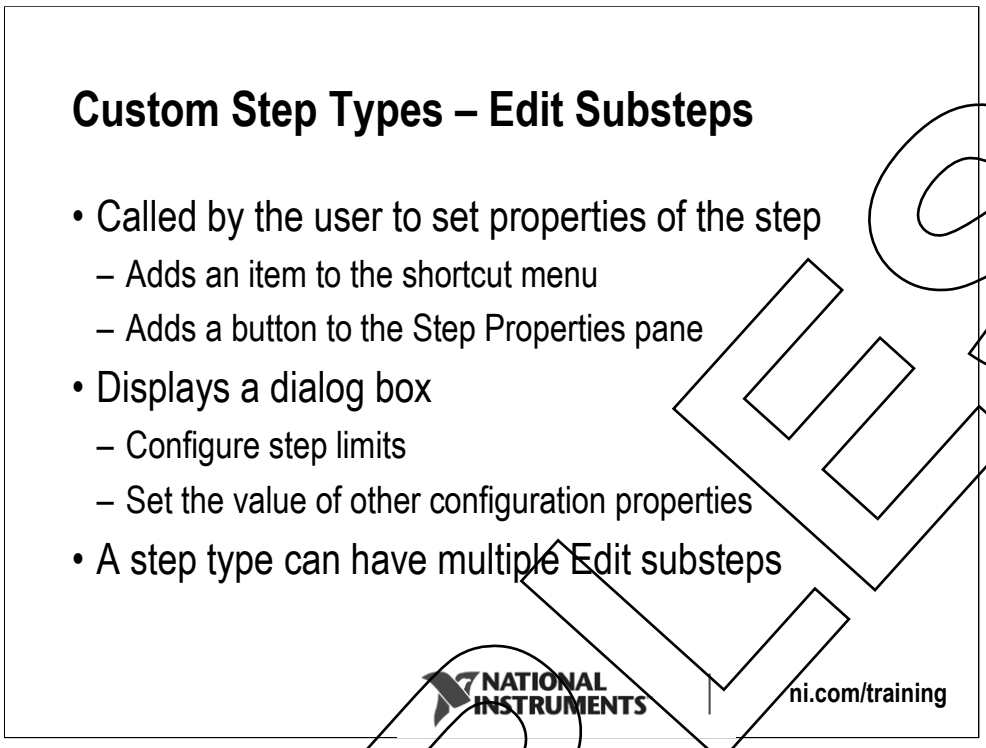
Custom Step Types – Pre-Step and Post-Step Substeps

Pre-Step and Post-Step substeps modify the run-time behavior of a custom step type. Pre-Step substeps execute just before the main code module for the step. Use Pre-Step substeps to retrieve configuration parameters, check property values, or otherwise prepare the step properties before executing the main code module. Post-Step substeps execute just after the main code module. Use Post-Step substeps to analyze the results of the main code module. Post-Step substeps may compare results of the code module to configured limits, determine the step status, add additional results to the results container, or otherwise modify the results of the step. A step can have multiple Pre-Step substeps. In this case, the substeps execute in the order they are listed. Multiple Post-Step substeps also execute in the order they are listed.



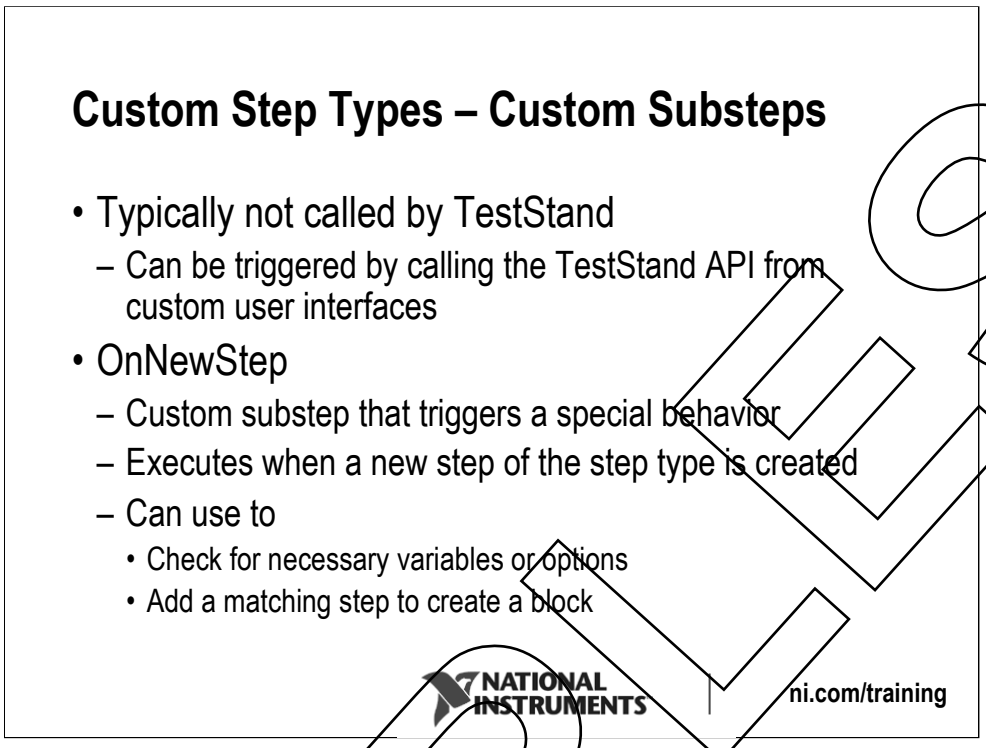
Custom Step Types – Pre-Step and Post-Step Substeps (continued)

This figure above shows each operation that takes place as a step executes and how Pre-Step and Post-Step substeps fit into the order of execution.



Custom Step Types – Edit Substeps

Edit substeps add an item to the context (right-click) menu for the step type and a button to the Step Properties pane. When the user clicks the button or selects the menu item, TestStand executes the code module you configure. Edit substeps call code modules that display a dialog box to configure custom properties of the step. A single custom step type can have multiple Edit substeps to change different categories of settings or perform other operations. Each Edit substep creates a button and a context menu item.



Custom Step Types – Custom Substeps

Custom step types are generally not called directly by TestStand. Use a custom step type to specify a code module that can be called by executing the Step.ExecuteSubstep method of the TestStand API. Use custom substeps to update a step based on changes in a user interface. For example, use a custom substep to tie controls on the user interface to configuration properties in the custom step type. Custom substeps can also be called by other entities such as the process model or another code module.

The OnNewStep substep is a special type of custom substep called directly by TestStand. If you create a custom substep named OnNewStep, TestStand calls the code module associated with the step each time you create a new instance of the step type. Use an OnNewStep substep to check for and/or create any variables or other components expected by the custom step type. You can also use OnNewStep substeps to automatically create matching sets of steps, similar to the the Flow Control step types, which have a configured step and an End step to indicate the end of a block of steps.

Exercise 3-1: Creating a Custom Step Type

GOAL

Create a custom step type to perform error handling.

NATIONAL INSTRUMENTS | ni.com/training

NI TestStand™ II: Customization Exercises

Course Software Version 4.0
July 2007 Edition
Part Number 324607A-01

Contents

Lesson 6
Customizing Database Logging

3

Custom Steps

Exercise 3-1 Creating a Custom Step Type

Goal

Create a custom step type to perform error handling.

Scenario

When a code module returns an error, you do not always need to shut down the test routine. You can ignore or correct many errors without restarting the test. Develop a custom step type that you can configure to handle specific errors. If a module called by the step returns an error, the step should compare the error to a configured list of errors and, if the error is in the list, handle it in one of two ways:

- Ignore the error, which clears the error and continues the sequence. The custom step should record the code of the error that was ignored.
- Retry the error, which executes the code module again. When retrying, the custom step should also be capable of displaying a dialog box to prompt the user to change aspects of the test environment to remedy the error. The step should also have a configurable maximum number of times to retry the error to avoid getting stuck in an infinite loop.

Design

A custom step is a good choice for this type of error handler because it allows you to use substeps to configure the errors to handle and to process the error output from a code module and determine the correct actions to take. Although you could write specific code to handle errors in your sequences, creating a custom step type gives you a reusable, self-contained component that you can use in multiple projects.

Available Components

Configure the errors to handle, the action to take, and options for each error though a dialog box. The provided Error Prompt VI displays a dialog box where the user configures the error information. The dialog

box returns an array of containers. Each item in the array represents one error and the options for handling the error. Call the Error Prompt VI using an Edit substep.

The provided Error Processor VI compares an error to the error handling array configured by the Error Prompt VI. The Error Processor VI clears errors, displays dialog boxes when necessary, returns information about any errors cleared, and indicates whether the step should loop and execute the code module again. Call the Error Processor VI using a Post-step substep.

Step Data

You must store multiple pieces of data in order to correctly handle errors. Your custom step should include the following properties in addition to the default step properties.

- ErrorsToHandle—An array of containers, with each item in the array representing one error and the options for handling it. The container type is defined by the `<Exercises>\TestStand II\Error Handler Step Type\Substeps\Error Options.ctl` control.
- ErrorToCheck—A storage location to pass the error from the code module to the Post-step substep for processing. Do not directly assign the error from the code module to Step.Result.Error because it will prematurely trigger the TestStand error handling mechanisms. Use the Error container for the data type of this property.
- Loop—A Boolean property indicating whether the step should loop and retry the code module. This value is set by the Post-step substep and used by the step looping options. The default value, True, indicates that the code module should execute in the first iteration of the step.
- CurrentIteration—An iteration counter to track the number of loops processed. The Post-step substep uses this value to determine whether looping should continue.
- ErrorCleared—If the Post-step substep ignores an error, this variable stores the code of the ignored error. Because this value might not always appear in the report or database, do not store it in the results container.

Implementation

1. Copy substep modules into the user components directory.

   ❑ Open a Windows Explorer window and browse to the `<TestStand>\Components\User\StepTypes` directory.

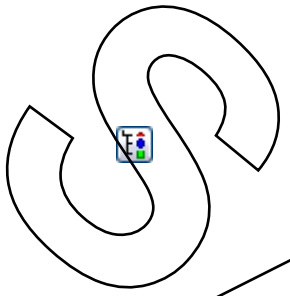
- ❑ Create a new folder called `ErrorHandler`.
- ❑ Open another Windows Explorer window and browse to the `<Exercises>\TestStand II\Error Handler Step Type\Substeps` folder.
- ❑ Copy the contents of the `<Exercises>\TestStand II\Error Handler Step Type\Substeps` folder and paste them in the `<TestStand>\Components\User\StepTypes\ErrorHandler` directory.

2. Create the step type.

- ❑ Close any open sequence files.
- ❑ Click the **Types** button.
- ❑ Select `MyTypes.ini`.
- ❑ Right-click in the Step Types container and select **Insert Step Type**.
- ❑ Name the new step `ErrorHandler`.

3. Create an edit substep to allow the step user to specify the errors to handle.

- ❑ Right-click the ErrorHandler step and select **Properties**.
- ❑ Click the **Substeps** tab.
- ❑ Select **LabVIEW** from the **Adapter** pull-down menu.
- ❑ Click **Add** and select **Edit** from the pull-down menu.
- ❑ Select the Edit substep.
- ❑ Enter `"Edit Errors to Handle"` in the Menu Item Name Expression textbox.
- ❑ Click **Specify Module**.
- ❑ Select `<TestStand>\Components\User\StepTypes\ErrorHandler\Error Prompt.vi` as the VI to call.



- ❑ Click **OK**.
- ❑ Click the **Create Custom Data Type From Cluster** button next to the Error Handling Info In parameter.
- ❑ Leave the default values in the Create Custom Data Type From Cluster dialog box and click **Create**.
- ❑ Click **OK** to exit the Edit LabVIEW VI Call dialog box.
- ❑ Click **OK** to exit the ErrorHandler Properties dialog box.
- ❑ Click **Ignore Error** to close the Warning dialog box.

Note TestStand warns that you have not specified values for the module parameters. You must create the properties to pass to the parameters before you can use them. You will return to specify the parameters later.

4. Define a step property to store an array of information about how to handle each error.
   - ❑ Expand the ErrorHandler step under the Step Types container.
   - ❑ Right click on the <Right-click to insert Field> item and select **Insert Field»Array of»Type» Error_Options**.
   - ❑ Enable the **Empty** checkbox in the Array Bounds dialog box.
   - ❑ Click **OK** to exit the Array Bounds dialog box.
   - ❑ Name the new field `ErrorsToHandle`.
5. Define a step property to store the error that should be checked against the errors to handle.
   - ❑ Right-click the <Right click to insert Field> item and select **Insert Field»Type»Error**.
   - ❑ Name the new field `ErrorToCheck`.
6. Define a step property to store whether to retry the step by looping.
   - ❑ Right-click the <Right click to insert Field> item and select **Insert Field»Boolean**.

- ❑ Name the new field `Loop`.
- ❑ Set the value of Loop to True.

7. Define a step property to store the current loop iteration.
   - ❑ Right-click the <Right-click to insert Field> item and select **Insert Field»Number**.
   - ❑ Name the new field `CurrentIteration`.
8. Define a step property to store the value of any error cleared by the function.
   - ❑ Right-click the <Right click to insert Field> item and select **Insert Field»Number**.
   - ❑ Name the new field `ErrorCleared`.
   - ❑ Verify your ErrorHandler step type properties match Figure 3-1.

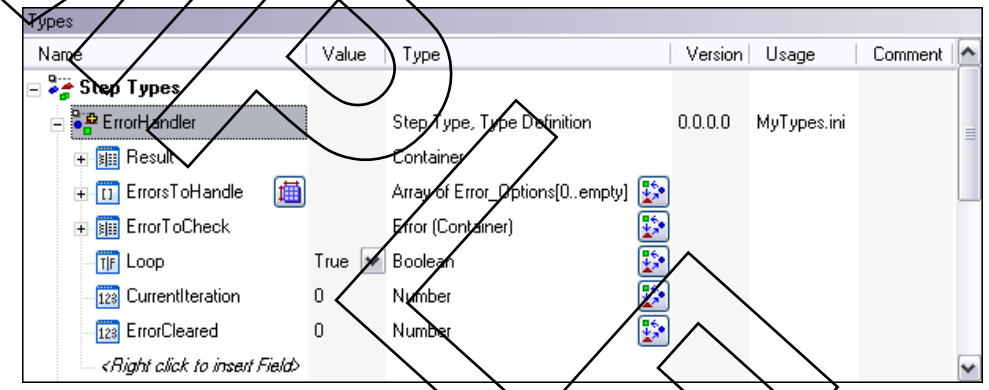


Figure 3-1. ErrorHandler Step Type Properties

9. Use the new properties in the Edit substep.
   - ❑ Right-click the ErrorHandler step type and select **Properties**.
   - ❑ Click the **Substeps** tab.
   - ❑ Select the Edit substep.

- ❑ Click **Specify Module**.
- ❑ Disable the **Default** checkbox for the Error Handling Info In parameter and enter `Step.ErrorsToHandle` for the value.
- ❑ Enter `Step.ErrorsToHandle` for the value of the Error Handling Info Out parameter.

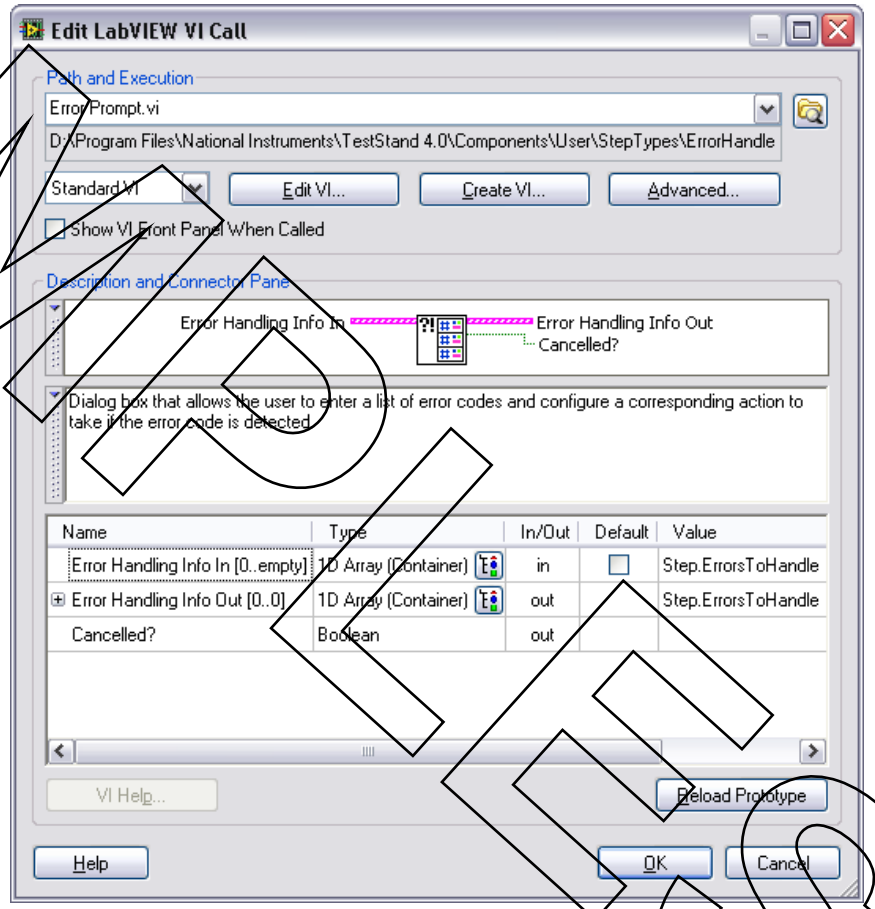


Figure 3-2. Edit Substep Module Settings

- ❑ Click **OK** to exit the Edit LabVIEW VI call dialog box.

10. Create a post substep to check an error against the errors to handle.

    ❑ On the Substeps tab of the ErrorHandler properties dialog box, select **LabVIEW** from the Adapter pull-down menu.

    ❑ Click **Add** and choose **Post-Step** from the pull-down menu.

    ❑ Click **Specify Module**.

    ❑ Select `<TestStand>\Components\User\StepTypes\ErrorHandler\Error Processor.vi` as the VI to call.

❑ Specify the parameter values as shown in Figure 3-3.

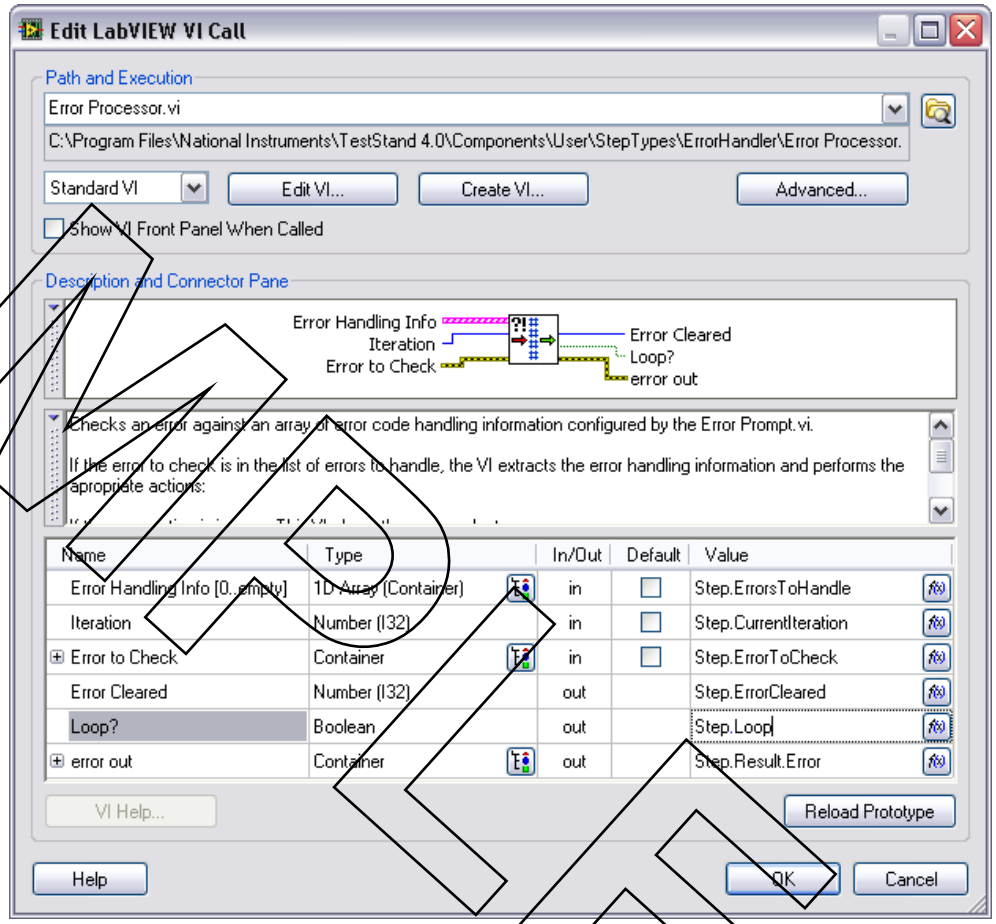


Figure 3-3. Post-Step Substep Module Settings

❑ Click **OK** to exit the Edit LabVIEW VI Call dialog box.

Your Substeps tab should resemble Figure 3-4.

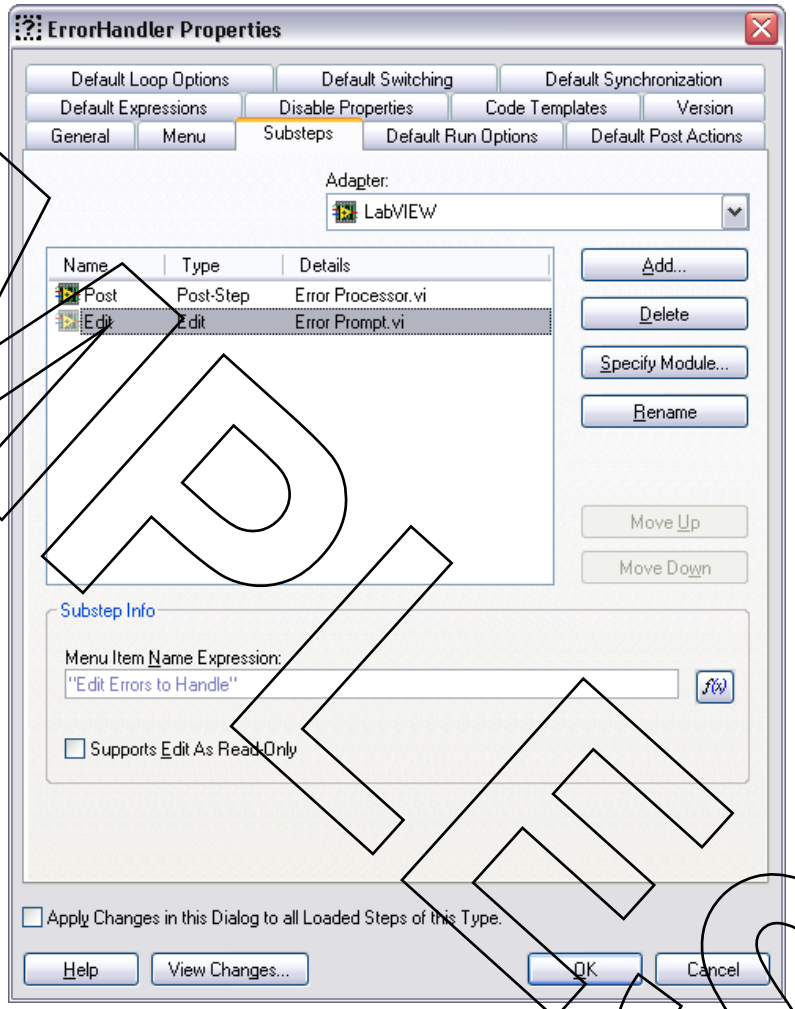


Figure 3-4. Substeps Tab

11. Configure loop settings to retry errors.

    ❑ Select the **Default Loop Options** tab.

    Configure the loop options as shown in Figure 3-5.

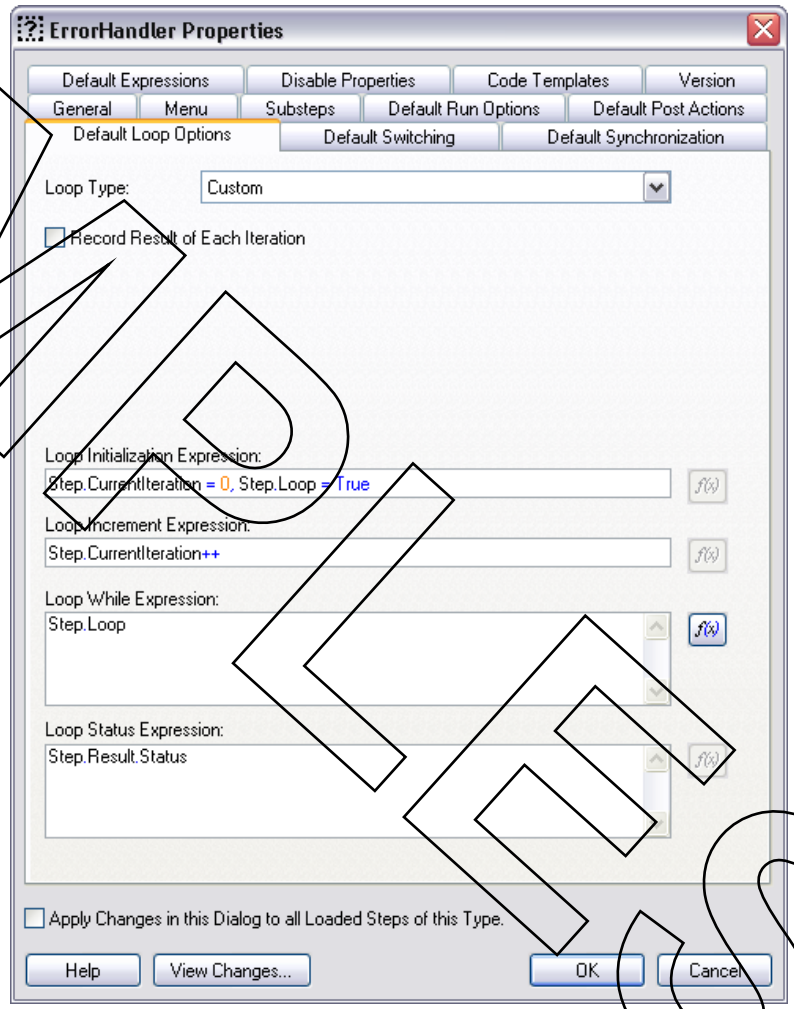


Figure 3-5. Loop Options Tab

12. Configure the step settings.
    - ❑ Select the **Menu** tab.
    - ❑ Enter `"Basic Error Handler"` in the Item Name Expression textbox.
    - ❑ Enter `Error Handling Steps` in the Group textbox.

❑ Select the **Disable Properties** tab.

Enable the **Loop Type**, **Loop Initialization Expression**, **Loop Increment Expression**, and **Loop While Expression** checkboxes, as shown in Figure 3-6.

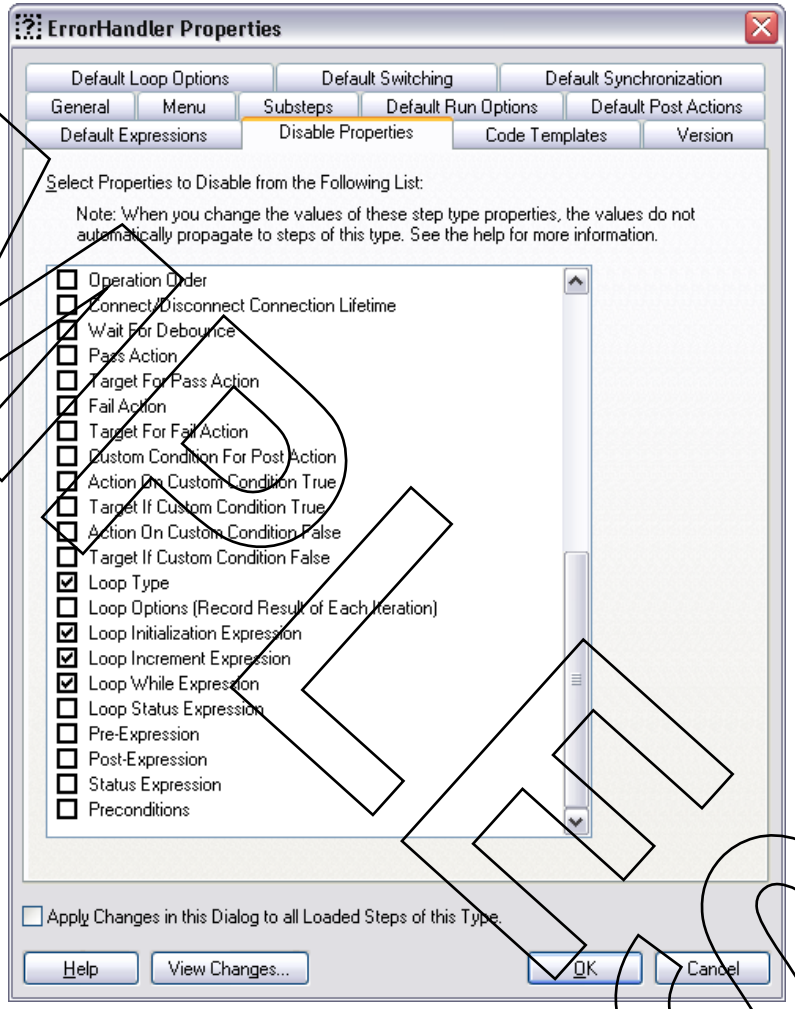


Figure 3-6. Disable Properties Tab

- ❑ Click **OK**.
- ❑ Select **File»Save** to save `MyTypes.ini`.
- ❑ Select **Do Not Increment Type Versions** in the Warning dialog box that appears.
- ❑ Enable the **Remove Modified Mark From Types** checkbox.
- ❑ Click **OK** to exit the Warning dialog box.

Testing

1. Create a sequence.
   - ❑ Select **File»New Sequence File**.
   - ❑ Save the sequence file as `<Exercises>\TestStand II\Error Handler Step Type\Test Sequence.seq`.
   - ❑ Expand the Error Handling Steps folder in the Insertion Palette.
   - ❑ Select the **LabVIEW** Adapter from the Selected Adapter pull-down menu.
   - ❑ Add a Basic Error Handler step to the Main step group of the MainSequence.
   - ❑ Name the step `Simulate Error`.

❑ Configure the Module for the step as shown in Figure 3-7.

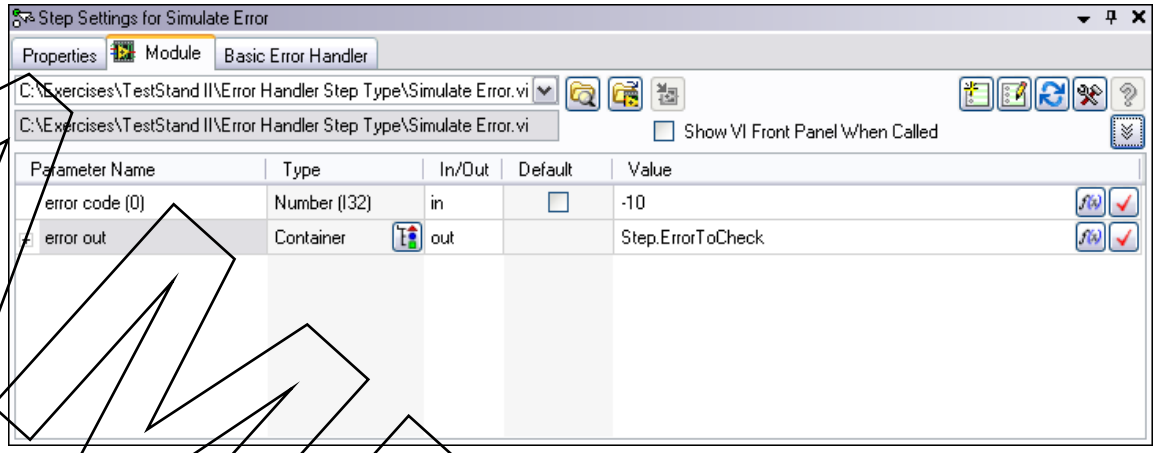


Figure 3-7. Simulate Error Module

2. Test the Edit substep.

❑ Select the **Basic Error Handler** tab in the Step Settings pane.

❑ Click **Edit Errors to Handle**. The button calls the edit substep you defined for the custom step type.

❑ When the Error Prompt.vi dialog box appears, enter `5000` in the **Error Code** textbox.

❑ Select **Ignore** as the Error Action. This instructs the error handler step to clear this error.

❑ Right-click `5000` in the Error List and select **Insert Code**.

❑ Enter `5001` in the Error Code textbox.

❑ Select **Retry** as the Error Action.

❑ Enter `4` for the Number of Retries.

This instructs the error handler step to attempt to execute the code module and check the error. If this error occurs, the error handler retries the module until the module does not return an error, a different error occurs, or the module has executed four times.

- ❑ Right-click the Error List step and select **Insert Code**.
- ❑ Enter `-10` in the Error Code textbox.
- ❑ Select **Retry** as the Error Action.
- ❑ Enable the **Infinite** checkbox.
- ❑ Enable the **Display Prompt?** checkbox.
- ❑ Enter `This will continue until stopped` in the Prompt Message textbox.

  This instructs the error handler step to attempt to execute the code module and check the error. If this error occurs, the error handler displays a message and retries the module. Retries continue until the module does not return an error, a different error occurs, or the operator clicks Stop on the dialog box.

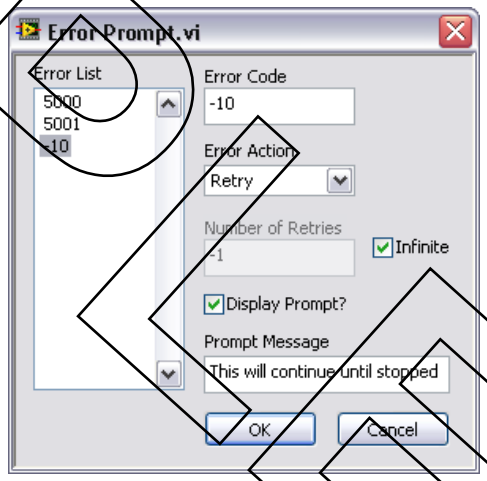


Figure 3-8. Error Prompt Dialog Box

- ❑ Click **OK** to exit the Error Prompt.vi dialog box.

3. Test the ignore error action.
   - ❑ Return to the Module tab and enter `5000` for the error code parameter.
   - ❑ Save the sequence file.
   - ❑ Select **Execute»Single Pass**.
   - ❑ Verify that the test report shows that the test passed, and that the Simulate Error step executed one loop and returned a status of **Done**.

Tip If your sequence does not work and you need to make changes to your custom step type, enable the **Apply Changes in this Dialog to all Loaded Steps of this Type** option in the ErrorHandler Properties dialog box so that you do not have to recreate and reconfigure the test step.

4. Test a limited number of retries.
   - ❑ Return to the Module tab of the Simulate Error step and enter `5001` for the error code parameter.
   - ❑ Save the sequence file.
   - ❑ Select **Execute»Single Pass**.
   - ❑ Verify that the test report shows that the test had an error status, and that the Simulate Error step executed five loops and returned an error status.

     The loop must execute again to check the loop condition, so there are five loops even though the code module only executed four times.

5. Test the prompt dialog box.
   - ❑ Return to the Module tab of the Simulate Error step and enter `-10` for the error code parameter.
   - ❑ Save the sequence file.
   - ❑ Select **Execute»Single Pass**.
   - ❑ When a dialog box appears, verify that the prompt message you entered for error `-10` displays.

- ❑ Click **Continue** in the dialog box.
- ❑ Click **Continue** a few more times. Count the number of times the dialog box displays.
- ❑ Click **Stop** from the dialog box.
- ❑ Verify that the test report shows the test had an error status, the Simulate Error step executed one more loop than the number of dialog boxes that displayed, and that it returned a status of `Error`.

6. Test a no error situation.
   - ❑ Return to the Module tab of the Simulate Error step and enter `0` for the error code parameter.
   - ❑ Save the sequence file.
   - ❑ Select **Execute»Single Pass**.
   - ❑ Verify that the test report shows that the test passed, and that the Simulate Error step executed one loop.
7. Test an error not handled by the step.
   - ❑ Return to the Module tab of the Simulate Error step and enter `5003` for the error code parameter.
   - ❑ Save the sequence file.
   - ❑ Select **Execute»Single Pass**.
   - ❑ Verify that the test report shows that the test had an error status, and that the Simulate Error step executed one loop and returned a status of **`Error`**.
8. Save and close the sequence file when you finish.