Creating Custom Step Type Edit Tabs in the Sequence Editor

CONTENTS

NATIONAL INSTRUMENTS

Introduction

When developing a custom step type to meet the needs of your test system one of the essential tasks is to offer an edit-time experience that allows users of your step type to configure the individual properties of the step instance. This is often accomplished using an edit substep that launches a modal dialog that handles accepting user inputs and writing out the results to the properties of your step. However, you may be interested in offering an experience that is integrated into the TestStand Sequence Editor similar to that of included step types without the need for modal dialogs.

TestStand offers access to the same APIs used to develop our included step types with .NET languages. This allows you to develop an integrated edit tab panel as an alternative to an edit substep. If you are new to custom step type development, you should read the TestStand Advanced Architecture series topic, Best Practices for Custom Step Type Development, as the same design principles apply to custom step types that use an edit substep or an integrated edit tab panel. This document will cover the design pattern of the edit tab panel, requirements for developing a panel, and the necessary APIs used. While Edit tab panels can be developed and built with any environment that can compile against the .NET Framework, this document will focus on the use of C# with Visual Studio.

Edit Tab Panel Design Pattern

For proper development of the edit tab panel, it is important to understand the design pattern that the API conforms to. The overall architecture of a custom step type remains unchanged when using an edit tab panel compared to using an edit substep.

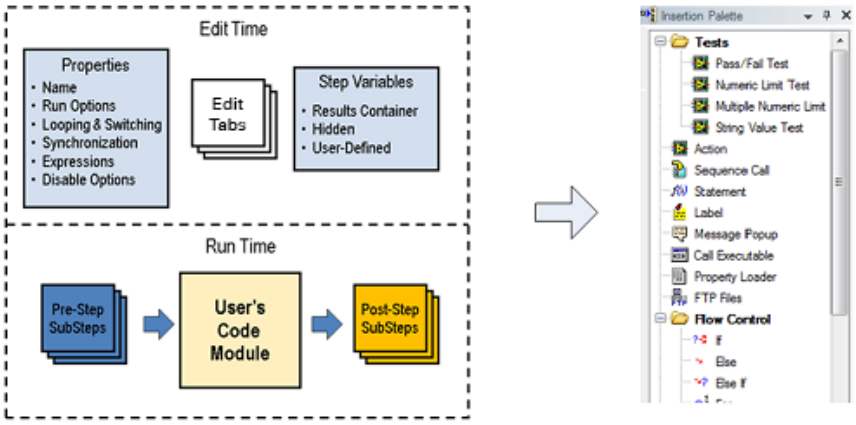

Figure 1 - Architecture of a Custom Step Type

The edit tab panel is used for handling edit-time requirements for your step type, primarily getting and setting configurable properties for step instances. Any work that needs to be done by a step type at run-time should be implemented in the step type sub-steps (e.g. Pre-Step and Post-Step) or in the code module of the step type.

The Sequence Editor will create a single instance of the edit tab panel for the lifetime of the Sequence Editor the first time that a step type using it is selected. The edit tab panel framework provides entry points to notify the panel when it needs to refresh and it provides a data structure for accessors to get and set properties of the steps the user has selected. More detail on this is provided in section 3 on the StepTypeSettingsPanel Base Class.

Preparing Visual Studio

the .NET Framework and Windows Forms controls is required for development of edit tab panels. TestStand ships with a set of special controls that are provided specifically for use with step settings panels. The functionality of the controls will be covered in more depth in section 4, Using the TestStand UI Controls. To use these controls in your project, you must first set up your Visual Studio environment to include them in the toolbox.

1. Open the Toolbox pane in Visual Studio.
2. Right-click in the Toolbox pane and select **Add Tab** menu item.
3. Name the new tab StepSettingsPaneControls.
4. Right-click on the new tab and select **Choose Items** menu item.
5. Click the Browse button and browse to %TestStandBin% and select TSDotNetSupport.dll
6. In the .NET Framework Components tab, sort the items by Assembly Name and scroll down to the assembly named TSDotNetSupport.
7. Check the following items in TSDotNetSupport and uncheck all others in that assembly:
   - ExpressionEdit (There are 2. Pick the one with the namespace NationalInstruments.TestStand.SeqEdit.Support)
   - PathControl
   - TSButton
   - TSCheckBox
   - TSComboBox
   - TSErrorProvider
   - TSFlowLayoutPanel
   - TSGroupBox
   - TSLabel
   - TSPanel
   - TSRadioButton
   - TSSplitContainer
   - TSTableLayoutPanel
   - TSTextBox

Setting Up the Edit Tab Project

With the toolbox properly setup, you will need to create the project for your edit tab panel. Each major version of TestStand will require its own build of the edit tab panel targeting the correct TestStand assembly and .NET Framework versions.

TestStand Version	.NET Framework Version
2014, 2014 SP1	4.0
2016, 2016 SP1	4.6
2017	4.6

1. In Visual Studio, create a new **Windows Forms Controls Library** project, targeting the appropriate .NET framework, and title it "MyStepTypePanel" for example (after going through this tutorial, your actual project can be named whatever you like).
2. Add references to the following TestStand assemblies to the MyStepTypePanels project:
   a. Choose the **Add Reference** menu item (right-click on References in Solution, or use Project menu).
   b. Choose the **Browse** tab.
   c. Browse to the %TestStandBin% folder and select the following 3 assemblies:
      - TSDotNetSupport.dll
      - NationalInstruments.TestStand.Interop.Controls.AxControls.dll
      - NationalInstruments.TestStand.Interop.Controls.dll

d. Click the OK button.
e. Choose the **Add Reference** menu item.
f. Choose the **.NET** tab and select the following 4 assemblies:
    - TestStand <version> Adapter API Primary Interop Assembly
    - TestStand <version> API Primary Interop Assembly
    - TestStand <version> UI ActiveX Interop Assembly
    - TestStand <version> UI Primary Interop Assembly
g. Click the OK button.

3. Change the Embed Interop Types setting on all references.
    h. Select all 7 assembly references that you added to the project.
    i. In the **Properties** pane, change the Embed Interop Types option to False.

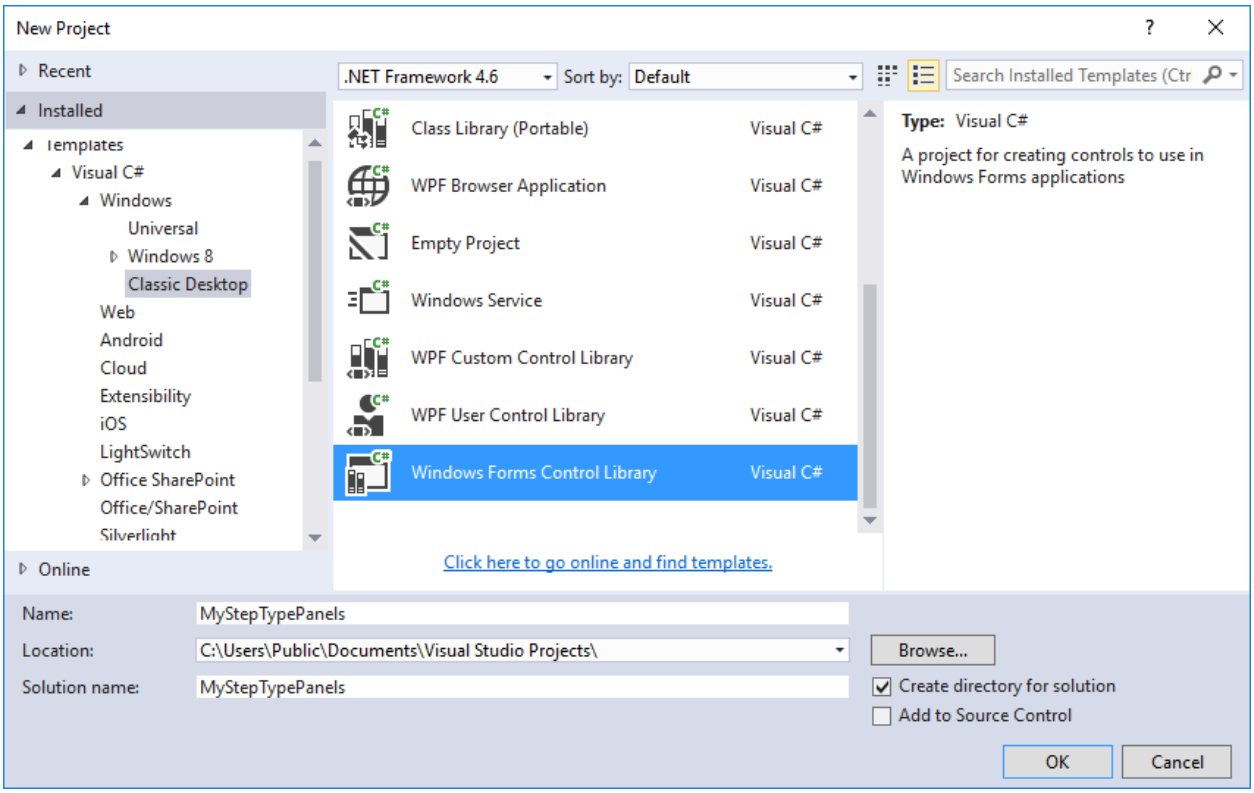

Figure 2 - Create Project Dialog in Visual Studio 2015

StepTypeSettingsPanel Base Class

TestStand edit tab panels all inherit from the `StepTypeSettingPanel` base class defined in TSDotNetSupport.dll. The `StepTypeSettingsPanel` base class contains the required properties and entry points that the TestStand Sequence editor expects. Visual Studio created a user control when the Windows Forms Control Library project was created. User control needs to be configured to inherit from `StepTypeSettingsPanel` instead of the default `UserControl`.

1. Close all Designer windows.
2. Add the following lines to the top of your Windows Control source file:

```
using NationalInstruments.TestStand.SeqEdit.Support;
using NationalInstruments.TestStand.Interop.API;
```

3. Change the base type from `UserControl` to `StepTypeSettingsPanel`

```
public partial class MyStepTypePanel : StepTypeSettingsPanel
```

4. Build the solution to verify that there are no errors.

Your user control now inherits the required data structures and entry points for use with the Sequence Editor. Your edit tab panel now will have a single instance for the lifetime of the Sequence Editor, which is constructed the first time that a step is selected.

> **Note:** Since the edit tab panel is constructed when a step is selected by the user, it is possible that a sequence file containing your step type may be run without the edit panel code ever being executed on subsequent loads of the file. This could occur because the file is being run in an operator interface, or is run immediately after being loaded in the Sequence Editor. It is important that all the state information required for your step type to operate is stored in step properties of the sequence file and does not require the edit panel code to be run to get it in a known state.

SelectedSteps Property

All instances of your step type will share a single instance of the edit tab panel. Therefore, the `StepTypeSettingsPanel` base class exposes a data structure called `SelectedSteps` which is a representation of the set of steps that are currently selected in the Steps view, either of a sequence file document or of an execution document. The `SelectedSteps` property is of the type `StepCollection`. Some other important objects provided by the `SelectedSteps` are objects referencing the `Adapter` and `StepType` properties of the current steps. Note that the `Adapter` and `StepType` properties may be null if the selected steps do not use a common adapter or have different step types. For `StepTypeSettingsPanel` classes, `StepType` will be guaranteed to be non-null when TestStand calls `RefreshContents()`. The data structure also offers access to the `EditManager` which contains the `EditContext` object for accessing data within TestStand.

Property Specifiers and Property Value Accessors

The other main component of the `SelectedSteps` object are property specifiers and property value accessors. Property specifiers act as a replacement for TestStand lookup strings, defining how to get and set and property value on a single property object. You can define custom property specifiers by implementing the `IPropertySpecifier` interface or you can create property specifiers for generic TestStand properties by calling the `GetPropertySpecifier` static method on the `StepCollection` class. For instance, to get a property specifier for the `Step.Result.Status` TestStand property, you call `StepCollection.GetPropertySpecifier("Result.Status")`.

Property value accessors are classes that allow you to get or set values of the same property on all the currently selected steps. You use the `GetPropertyAccessor()` method on the `SelectedSteps` property to get a property value accessor. This method takes either a TestStand lookup string or a property specifier object. For instance, to get a property value accessor for the `Step.Result.Status` TestStand property, you call `SelectedSteps.GetPropertyAccessor("Result.Status")`.

Use the `Value` property or the `TryGetValue()` method to get the common value of the property on all selected steps. The obtained value is null if not all the selected steps have the same property value. To get individual property values for the selected steps call the `GetValues()` method. Use the `SetValue()` method to set the property value on all the selected steps.

The property value accessors handle many of the actions required for editing steps in the step settings pane, including the following:

- Determines whether the sequence file is editable before making changes. For example, the sequence file might be read-only, the current user might not have permissions to edit sequence files or the sequence file might be executing.
- Automatically checks out the sequence file from source code control, if necessary.
- Records information to support undo and redo commands.

When using TestStand UI controls, property specifier will need to be specified and the control will get the property value accessor internally to handle the getting & setting of the property.

RefreshContents Entry Point

The primary entry point that handles the getting of TestStand properties is the `RefreshContents` method defined in the `StepTypeSettingsPanel` class. The Sequence Editor will call the `RefreshContents()` method each time a step is selected or when the step settings panel tab is selected. You will want to override this method for your panel. This method is where you can do the bulk of the work of interpreting the user's input. If you are using the TestStand UI controls then the work of handling user input has already been handled for you and you can use this method to call the `RefreshControlValue()` method on each of the controls to have their connected properties updated in TestStand.

There will be instances where your step types may need more information than just the currently selected steps. For example, the included TestStand Wait step needs a list of sequence call steps that start new executions so that it can fill a combobox with a list of steps to wait on. There are two general strategies that you can use to get this information. The first strategy is to recompute the information each time `RefreshContents()` is called. This is the most common way and has the advantage of not requiring additional events nor the caching of information, which can become outdated. However, you will also want to make sure that your `RefreshContents()` method can exit quickly so it does not hang the TestStand Sequence Editor. If the generation of this additional information takes too much time, then you can have the step settings panel create an instance of another object that maintains the state separately from the step settings panel and the panel just queries this object for the information in `RefreshContents()`.

Using the TestStand UI Controls

TestStand ships with a set of UI controls specifically designed for use with step type edit panels. These controls are the same set of controls used by included step types in TestStand, offering a consistent UI between first and third-party step types. The controls are also designed to handle TestStand-specific functionality such as handling expressions, localization with resource strings, integration with the undo & redo menu, and the setting of step type properties. The following panel controls are available:

Panel Controls	Panel Containers
TSTextBox, TSComboBox, TSExprTextBox, TSRadioButton, TSCheckBox, TSButton, TSLabel, TSToolbar, TSToolbarButton	TSGroupBox, TSSplitContainer, TSPanel, TSTableLayoutPanel, TSFlowLayoutPanel

Many of these controls are extensions of Windows Forms controls like TSTextBox or TSLabel and offer the same functionality as their Windows forms counterparts. Other controls are specific to TestStand, like TSExprTextBox, offering the same expression editing experience as is available elsewhere in the Sequence Editor.

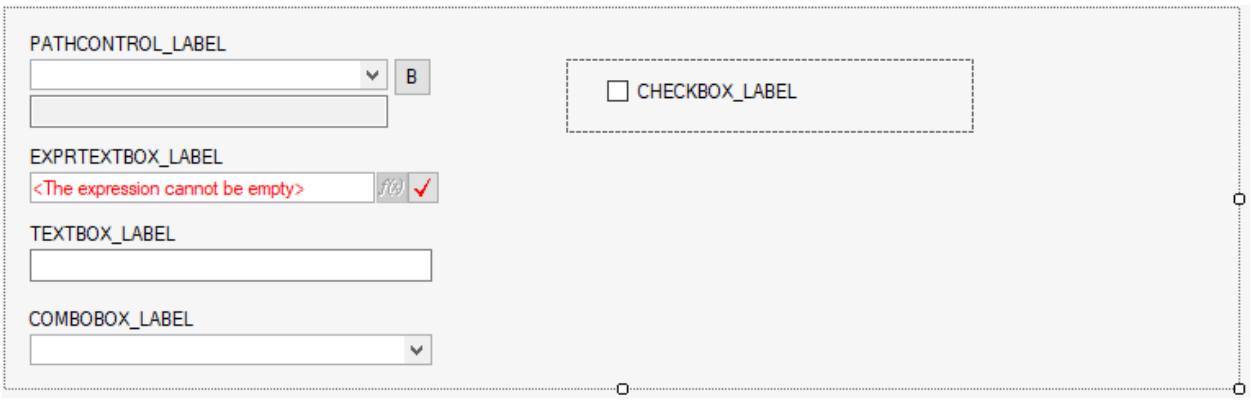

Figure 3 - TestStand UI Controls in a Windows Control

Connecting to Step Properties

One of the primary advantages of using the TestStand UI controls over standard .NET controls is the ability to connect the controls directly to step properties in TestStand. This is handled using `PropertySpecifier` objects to define what step property the control should write its value to. Typically, you will want to set the `PropertySpecifier` for the control during the initialization of your step type panel, which can be done by overriding the Initialize method of the `StepTypeSettingsPanel` base class. To get the property specifier, you need to access the static class `StepCollection` and call `GetPropertySpecifier()`. The necessary lookup string to pass, `GetPropertySpecifier()` should be relative to the "Step" property in TestStand. Once you have this object you can set it to the `PropertySpecifier` property of the control.

For instance, you may have an `Initialize` method similar to the following:

```
protected override void Initialize()
{
	base.Initialize();
	exampleTextBox.PropertySpecifier =
	StepCollection.GetPropertySpecifier("Result.Status");
}
```

With the controls connected to the necessary step properties, you will need to add the code to your `RefreshContents()` method to handle updating the value of the TestStand property with the control value. The TestStand UI controls offer the method `RefreshControlValue()` that will write the current value of the control to the TestStand property.

Your resulting RefreshContents() method should be similar to the following:

```
protected override void RefreshContents(RefreshReasons refreshReason)
{
	//additional RefreshContents work
	exampleTextBox.RefreshControlValue();
}
```

Additional Features

The TestStand UI controls support connecting to the TestStand localization feature. Localization is handled just like in TestStand using Resource Strings. For your panel you will need to create a new Resource String Category that contains all the necessary resource strings that your panel will use. It is

recommended that you create a new INI file that encapsulates this functionality, to better support deploying your custom step type independently of other TestStand resource strings.

To use your localized resource strings, you will first need to set the resource string category used by your panel. One of the properties inherited by the `StepTypeSettingsPanel` base class is the `ResourceStringCategory` property. You will need to set this property to your resource string category defined in your INI file. This property can be set by opening the designer view of your step type edit panel and selecting the background to view the properties of your panel, here you will find the `ResourceStringCategory` property to set. You can now use your resource strings to set localizable text for your UI controls. For any text field of your UI control, set the value to be the resource string to replace. Note that if that resource string is not found, then the text will be displayed as is.

Finally, there are several other kinds of TestStand functionality added to the UI controls. Some important properties to be aware of:

- **CommandDescription:** The text to display in Edit»Undo menu for this control after the user changes the control value. The text in this field supports localization.
- **HighlightNonDefaultValues:** If the value is not default and this is true, it will be shown in bold. The default value is determined by the value of the property on the step type.
- **Label**: The TSLabel object associated with this UI control. If the box is disabled, the label will be updated as well if linked.
- **Numeric**: For UI controls that support text input, like TSTextBox. Set to true if displaying a numeric value.

Accessing Data from TestStand

EditContext

There are instances that your step type edit panel needs to access data from TestStand, like in the case that you wish to populate a control with values pulled from your sequence file. In a code module, this kind of task would be accomplished by using the TestStand API on a `SequenceContext` object. The `SequenceContext` that is used in that case is an object that only exists at run-time to represent the state of the system. Since the step type edit panel code is executed at edit time, a similar object is available called `EditContext`. The `EditContext` is an object that is also of the `SequenceContext` type, but instead of representing the run-time state, it instead represents the best edit-time approximation of a `SequenceContext`.

To access the `EditContext` you will need to use the `SelectedSteps` data structure. One of the properties of `SelectedSteps` is an object called the `EditManager`. The `EditManager` provides access to several edit-time properties, the most important being `EditContext`. A call into the `EditContext` object should look like: `this.SelectedSteps.EditManager.EditContext`.

From there you can use the `EditContext` with the TestStand API to access data in the TestStand engine as you normally would in a code module with `SequenceContext`. This should only be used for reading data from TestStand, your step settings panel should not be changing properties other than step properties through UI controls and property value accessors.

Configure the Step Type to use the Step Settings Pane

Once your custom step type edit panel has been developed you will need to configure the step type to use the step settings pane that you just created. You will need to build a second class that exposes additional information about your step type to TestStand and then point the step type to your assembly and the new info class.

1. Create a new class that specifies the tab information for the step settings pane:
   a. In a new file or at the bottom of the MyStepTypePanel.cs file within the same namespace, insert a new class called `MyStepTypeTabInfo` that derives from the `StepTypeEditPanelTabInfo` base class.
   b. Add the `CreatePanel` method to create an instance of the MyStepTypePanel class.
   c. Add the `TextExpr` property to the class to specify an expression that evaluates to the text to display in the tab. The code below uses the ResStr expression function to set the tab text to a string from a resource file.
   d. Add the `IconName` property to the class to specify an expression that evaluates to the name of the icon to display in the tab. The code below sets the icon to the Goto step icon.

```
public class MyStepTypeTabInfo : StepTypeEditPanelTabInfo
{
        public override StepSettingsPanel CreatePanel() { return new
        MyStepTypePanel(); }
        public override string IconName { get { return "goto.ico"; } }
        public override string TextExpr { get { return "ResStr(\"GOTO_STEP_TYPE\",
        \"TAB_TEXT\")"; } }
}
```

2. Build the solution.

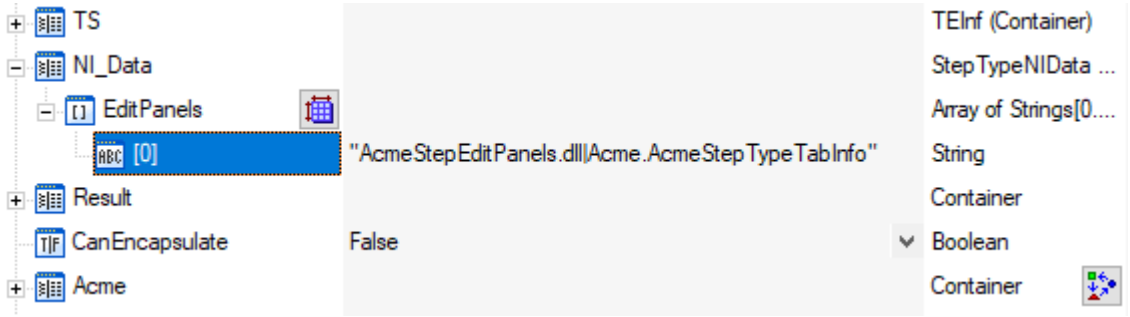

Figure 4 - Configuring the Edit Panels Property to Use a Custom Step Type Edit Tab

3. In TestStand, edit the step type to use the new step settings panel:
   a. Enable the Show Hidden Properties Station option (Configure»Station Options»Preferences tab).
   b. Expand the step type properties in the Types window in the Sequence Editor.
   c. Increase the size of the NI_Data.EditPanels array property to add a new element.
   d. Change the text for the item you added to be of the form "<dll name>|<namespace>.<TabInfo classname>". In this case, we will change the text to "MyStepTypePanels.dll|MyStepTypePanels.MyStepTypeTabInfo", where MyStepTypePanels.dll is the assembly that your project built and MyStepTypePanels.MyStepTypeTabInfo is the fully specified class name for the class you created above.
4. In TestStand, add the directory that contains the assembly to the TestStand search directories, or move the assembly to a directory in the TestStand search directories, such as %TestStandPublic%.

Additional Resources

Included with this document is an example step type that uses a custom step type edit panel. You can explore this example to see an implementation of the above concepts. When opening the solution, you will need to update the referenced TestStand assemblies to point to the correct version and locations on disk for your version of TestStand. The solution is currently configured to build for TestStand 2017 64-bit.

To learn more about custom step types in general, see the TestStand Custom Step Types help topic.

In addition, you should read the Best Practices for Custom Step Type Development topic of the TestStand Advanced Architecture Series.

If you have any feedback or questions about this content, please email ats.pse@ni.com.