A. Leonard, *Building Custom Tasks for SQL Server Integration Services*
https://doi.org/10.1007/978-1-4842-6482-9_11

# 11. Minimal Coding for the Complex Editor

Andy Leonard[1]
(1) Farmville, VA, USA

This chapter continues the goals of this section of the book, which are

- Surface a more "SSIS-y" experience for users of the task including a "pretty" editor with General and Settings pages "views"



ing existing ExecuteCatalogPackageTask code and replacing the existing task editor project with a new version.

In this chapter, we begin minimally coding the ExecuteCatalogPackageTaskComplexUI by creating the editor form and adding General and Settings views to the View ➤ Node ➤ Property Category ➤ Property hierarchy for SSIS tasks.

## Updating DtsTask Interface Members

The current code in the ExecuteCatalogPackageTaskComplexUI class, reordered, is shown in Listing 11-1:

```
public class ExecuteCatalogPackageTaskComplexUI : IDtsTaskUI
{
  public void Initialize(TaskHost taskHost, IServiceProvider serviceProv
  {


    throw new NotImplementedException();
  }
  public ContainerControl GetView()
  {
    throw new NotImplementedException();
  }
  public void New(IWin32Window parentWindow)
  {
    throw new NotImplementedException();
  }
  public void Delete(IWin32Window parentWindow)
  {
    throw new NotImplementedException();
  }
}
```

***Listing 11-1*** ExecuteCatalogPackageTaskComplexUI current code, reordered

The new order sorts well with my CDO (that's "OCD" with the letters in the proper order).

In the sections that follow, we build out the "SSIS-y" editor by adding members (internal variables), interface member methods, and views (which will translate to "pages" in the editor).

## Adding Internal Variables

Begin coding by adding two variables just after the declaration of the ExecuteCatalogPackageTaskComplexUI class, listed in Listing 11-2:

```
private TaskHost taskHost = null;
private IDtsConnectionService connectionService = null;
```

***Listing 11-2*** Declare the taskHost and connectionService internal variables

Once added, your code should appear as shown in Figure 11-1:

```
public class ExecuteCatalogPackageTaskComplexUI : IDtsTaskUI
{
    private TaskHost taskHost = null;
    private IDtsConnectionService connectionService = null;
```

***Figure 11-1*** Adding the internal variables: taskHost and connectionService

The taskHost and connectionService variables are used to pass values between the ExecuteCatalogPackageTaskComplexUI editor class and the ExecuteCatalogPackageTask class.

## Coding Interface Member Methods

In the Initialize method, replace the line of code auto-generated in the previous chapter – `throw new NotImplementedException();` – with the code listed in Listing 11-3:

```
this.taskHost = taskHost;
this.connectionService =
 serviceProvider.GetService(typeof(IDtsConnectionService))
  as IDtsConnectionService;
```

***Listing 11-3*** Initializing taskHost and connectionService

When finished, your code should appear as shown in Figure 11-2:

When finished, your code should appear as shown in Figure 11-2:

```
public void Initialize(TaskHost taskHost, IServiceProvider serviceProvider)
{
    this.taskHost = taskHost;
    this.connectionService = serviceProvider.GetService(typeof(IDtsConnectionService)) as IDtsConnectionService;
}
```

***Figure 11-2*** The taskHost and connectionService variables, initialized

In the GetView method, replace the line of code auto-generated in the previous chapter – `throw new NotImplementedException();` – with the code that instantiates a new instance of the editor form listed in Listing 11-4:

```
return new ExecuteCatalogPackageTaskComplexUIForm(taskHost, connectionSe
```

***Listing 11-4*** Code to instantiate a new instance of the editor form

When finished, your code should appear as shown in Figure 11-3:

```
public ContainerControl GetView()
{
    return new ExecuteCatalogPackageTaskComplexUIForm(taskHost, connectionService);
}
```

***Figure 11-3*** Instantiate a new instance of the editor form

The red squiggly line beneath `ExecuteCatalogPackageTaskComplexUIForm` informs us there's an issue with this line of code. Click Quick Actions gives more information, as shown in Figure 11-4:

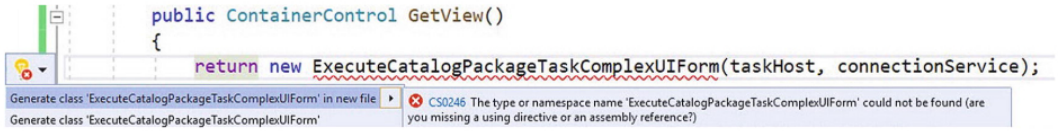



***Figure 11-4*** ExecuteCatalogPackageTaskComplexUIForm does not yet exist

Comment out the throw statements in the `New` and `Delete` interface member methods,

editing the code in each method to match Listing 11-5:

```
// throw new NotImplementedException();
```

***Listing 11-5*** Throw statements, commented out

The `New` and `Delete` interface member methods should appear as shown in Figure 11-5:

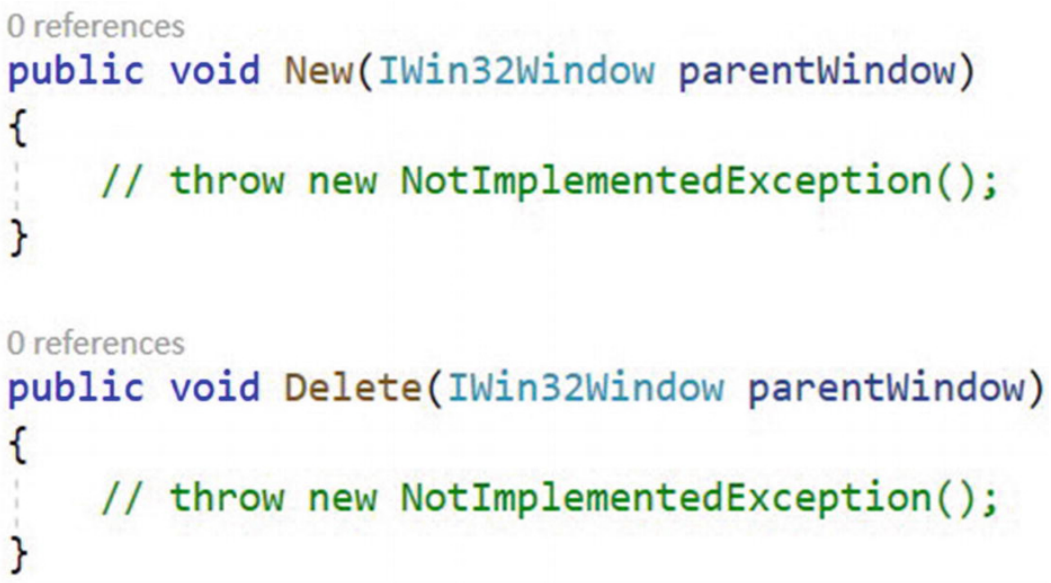



***Figure 11-5*** New and Delete member methods

The next step is to create a form for the editor.

## Creating the Editor Form

The editor form provides an interface for developers to view and configure task properties. Earlier in this book, we built a custom editor. In this section, we add a more “SSIS-y” editor.

Begin adding the editor form by adding a new class. To add a new class, right-click the ExecuteCatalogPackageTaskComplexUI project in Solution Explorer, hover over the Add menu item, and then click Class, as shown in Figure 11-6:

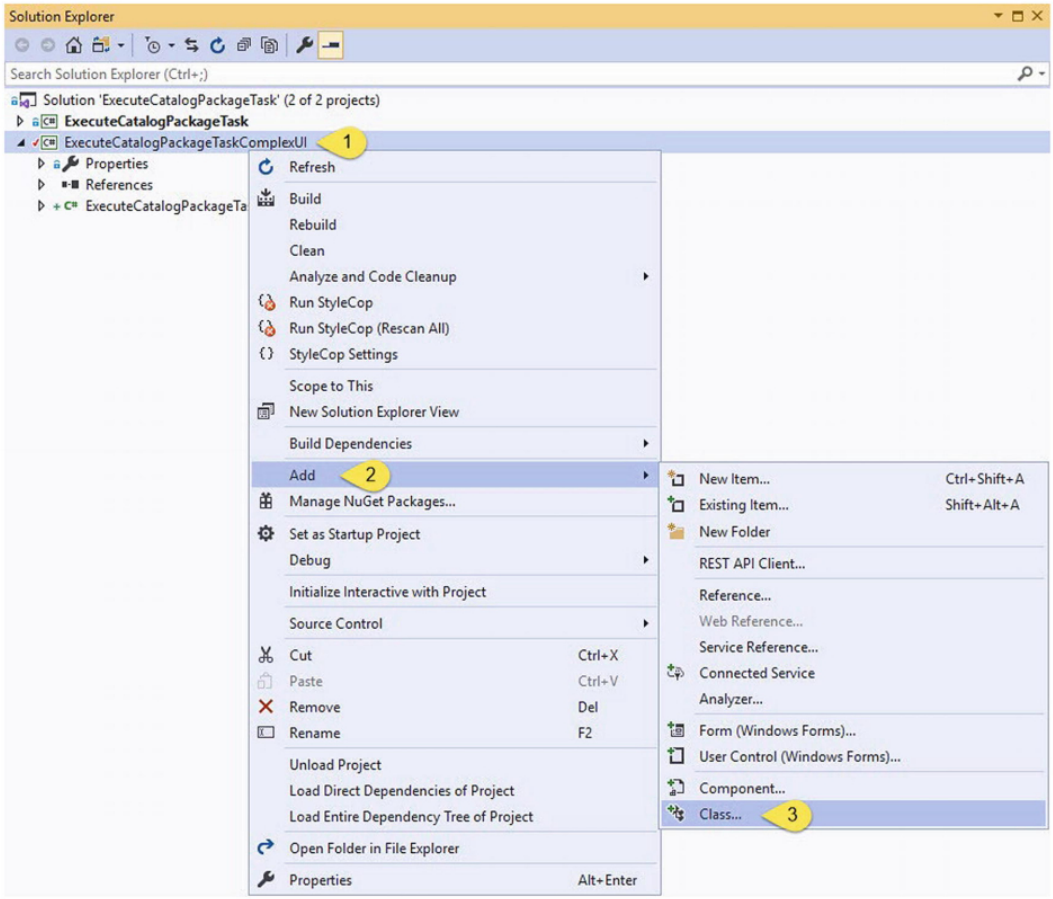
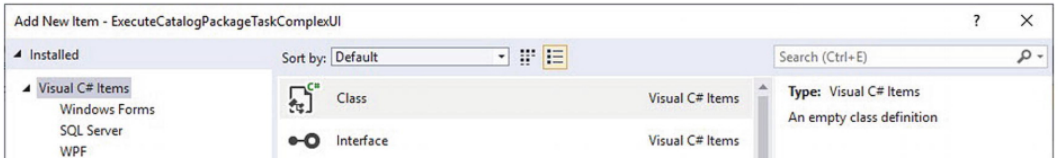



***Figure 11-6*** Adding a new class

When the Add New Item dialog displays, name the new class "ExecuteCatalogPackageTaskComplexUIForm," as shown in Figure 11-7:



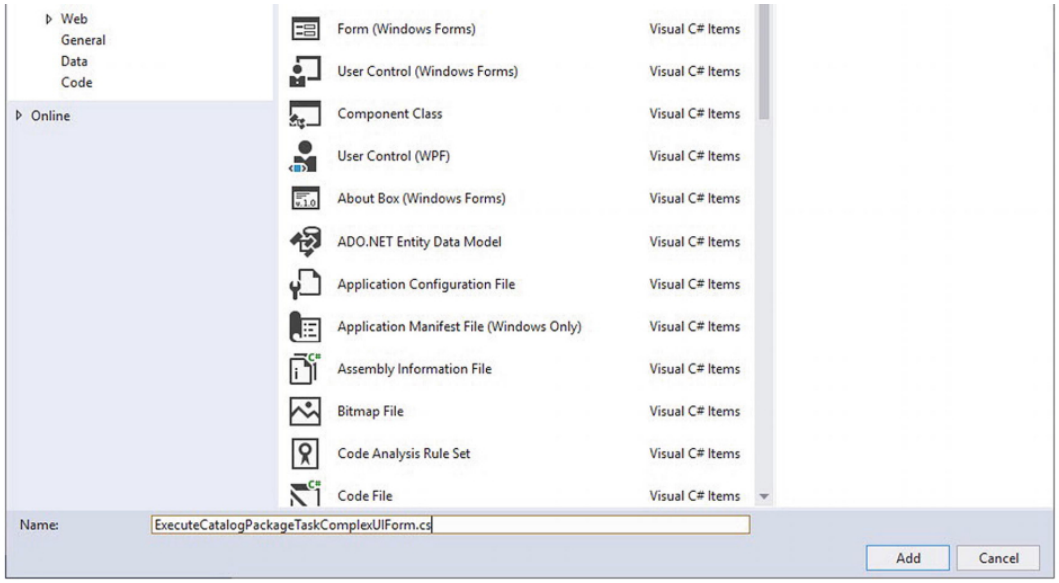



***Figure 11-7*** Add the ExecuteCatalogPackageTaskComplexUIForm class

The ExecuteCatalogPackageTaskComplexUIForm class will appear as shown in Figure 11-8:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ExecuteCatalogPackageTaskComplexUI
{
    1 reference
    class ExecuteCatalogPackageTaskComplexUIForm
    {
    }
}
```

***Figure 11-8*** The ExecuteCatalogPackageTaskComplexUIForm class

Modify the ExecuteCatalogPackageTaskComplexUIForm class declaration to read as shown in Listing **11-6**:

```
public partial class ExecuteCatalogPackageTaskComplexUIForm : DTSBaseTas
```

***Listing 11-6*** Modifying the ExecuteCatalogPackageTaskComplexUIForm class declara

The class declaration should appear as shown in Figure **11-9**:

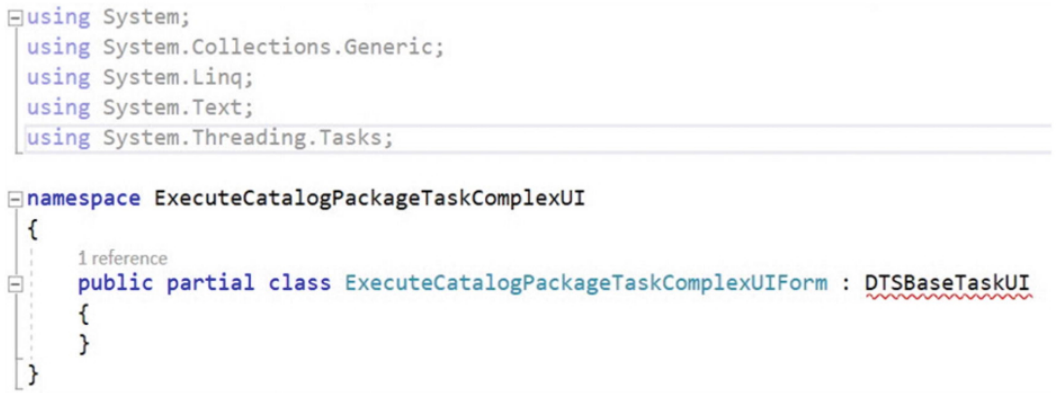



***Figure 11-9*** ExecuteCatalogPackageTaskComplexUIForm declaration, modified

The ExecuteCatalogPackageTaskComplexUIForm class inherits from DTSBaseTaskUI, an interface defined in the Microsoft.DataTransformationServices.Controls assembly. Although referenced in the ExecuteCatalogPackageTaskComplexUI project, the Microsoft.DataTransformationServices.Controls assembly is not listed in the collection of assemblies being *used* by the ExecuteCatalogPackageTaskComplexUIForm class. Remedy this (and eliminate the red squiggly line beneath the DTSBaseTaskUI inheritance implementation) by adding the using statement listed in Listing **11-7**:

```
using Microsoft.DataTransformationServices.Controls;
```

***Listing 11-7*** Adding a using statement

The ExecuteCatalogPackageTaskComplexUIForm class should now appear as shown in Figure 11-10:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.DataTransformationServices.Controls;

namespace ExecuteCatalogPackageTaskComplexUI
{
    1 reference
    public partial class ExecuteCatalogPackageTaskComplexUIForm : DTSBaseTaskUI
    {
    }
}
```

***Figure 11-10*** Microsoft.DataTransformationServices.Controls added

The next step is adding private members for `Title` and `Description` as string constants and a public static `taskIcon`. `Title`, `Description`, and `taskIcon` will be used to configure the base class in the next step.

> *Inheritance*, *encapsulation*, and *polymorphism* make up the foundation of Object-Oriented Programming, or OOP. Inheritance allows developers to design a *base class* that is more generic and then *inherit* members in a more-specified class called a *derived class*. See docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/inheritance for more information.

Add private members for `Title` and `Description`, and a public static `taskIcon` declaration , by adding the code in Listing 11-8 just after the partial class declaration:

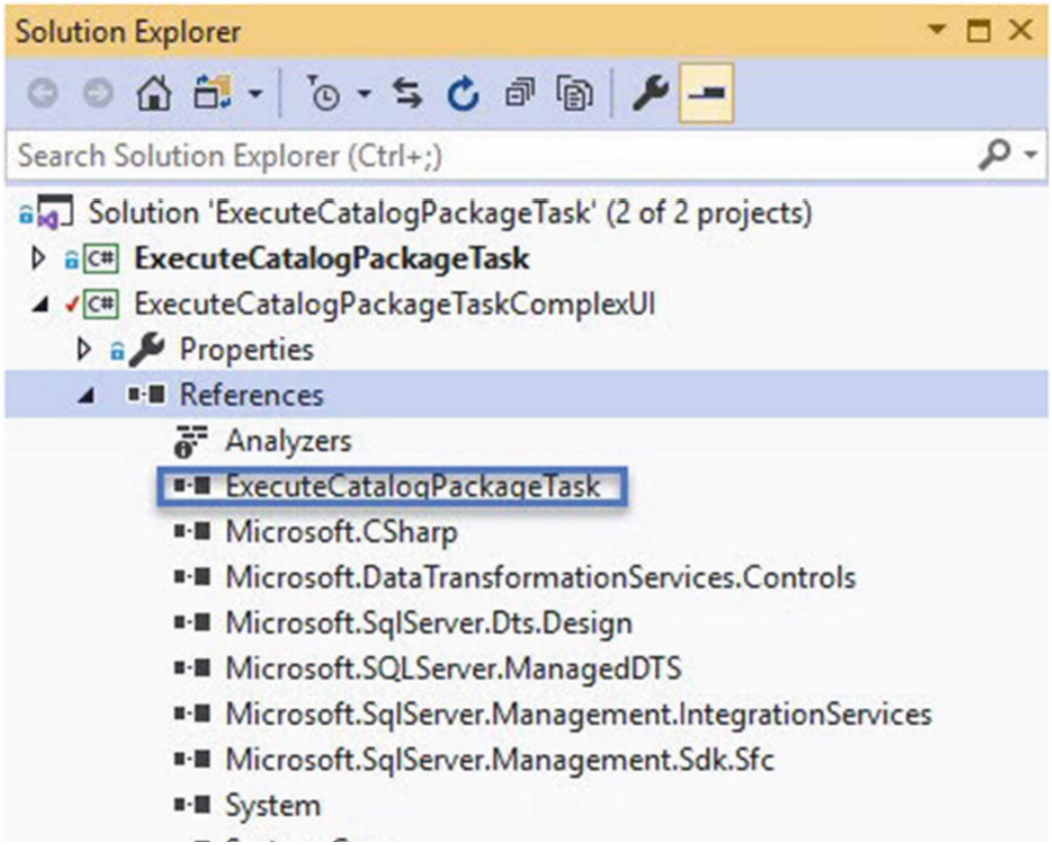
```
private const string Title = "Execute Catalog Package Complex Task Edito
private const string Description = "This task executes an SSIS package i
public static Icon taskIcon = new Icon(typeof(ExecuteCatalogPackageTask.
```

***Listing 11-8*** Adding Title, Description, and Icon

Members (or properties) in the base class can be set from the derived class if the base class allows.

### Adding References

The next step is to add missing references. In Solution Explorer, right-click the References virtual folder and click "Add Reference…." Expand the Projects node and check the checkbox beside ExecuteCatalogPackageTask to add a reference to the ExecuteCatalogPackageTask project. Expand Assemblies, select Framework, and then add the System.Drawing assembly. Click the OK button to close the Reference Manager dialog. The added references should now appear in the References virtual folder as shown in Figure 11-11:



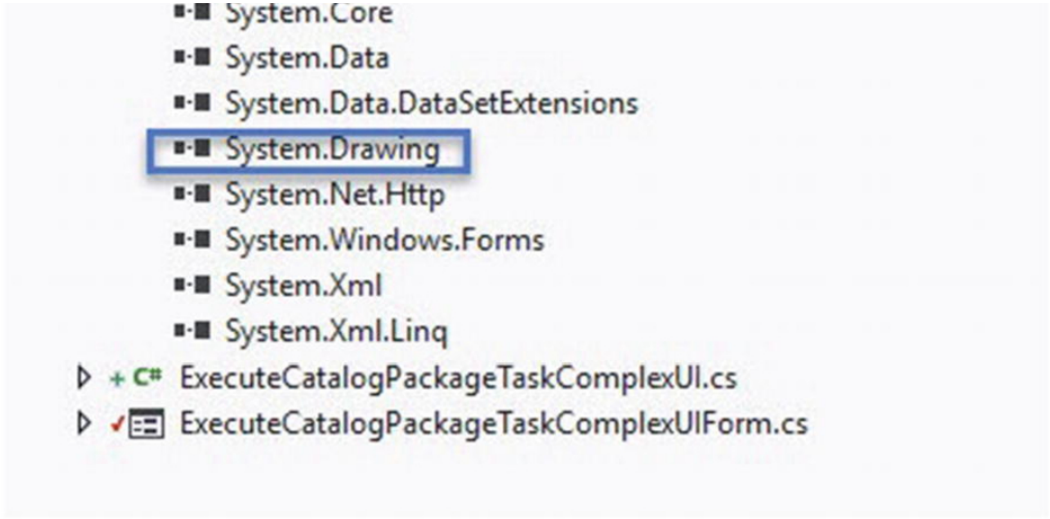



***Figure 11-11*** ExecuteCatalogPackageTask and System.Drawing references added

The next step is coding the form constructor.

## Coding the Form Constructor

When an object is loaded into memory – or *instantiated* – a *constructor* initializes the state of the object by loading and initializing object members (properties). The screenshot recorded in Figure **11-12** (originally shown in Figure **11-3**) contains a call to the ExecuteCatalogPackageTaskComplexUIForm class, which calls the constructor for the ExecuteCatalogPackageTaskComplexUIForm class. The call displays an error, as shown in Figure **11-12**:

```
public ContainerControl GetView()
{
    return new ExecuteCatalogPackageTaskComplexUIForm(taskHost, connectionService);
}
```

***Figure 11-12*** Calling ExecuteCatalogPackageTaskComplexUIForm

The error remains because there is no constructor for the ExecuteCatalogPackageTaskComplexUIForm class that accepts two arguments. Add the code in Listing **11-9** to create a constructor for the

ExecuteCatalogPackageTaskComplexUIForm class that accepts two arguments:

```
public ExecuteCatalogPackageTaskComplexUIForm(TaskHost taskHost, object
        base(Title, taskIcon, Description, taskHost, connections)
       { }
```

***Listing 11-9*** Coding the ExecuteCatalogPackageTaskComplexUIForm constructor

Once added, the ExecuteCatalogPackageTaskComplexUIForm constructor should appear as shown in Figure **11-13**:


***Figure 11-13*** Adding the ExecuteCatalogPackageTaskComplexUIForm constructor code

The `TaskHost` object is not initialized because the class header is missing a `using` statement for the `Microsoft.SqlServer.Dts.Runtime` assembly. Add the using statement shown in Listing **11-10**:

```
using Microsoft.SqlServer.Dts.Runtime;
```

***Listing 11-10*** Adding using Microsoft.SqlServer.Dts.Runtime

`TaskHost` is no longer in error

Add two constants to initialize the task Title and Description properties and a static Icon to use with the editor form using the code in Listing **11-11**:

```
private const string Title = "Execute Catalog Package Task Editor";
private const string Description = "This task executes an SSIS package i
public static Icon taskIcon = new Icon(typeof(ExecuteCatalogPackageTask.
```

***Listing 11-11*** Adding Title, Description, and taskIcon

Once added, the code appears as shown in Figure 11-14:

```
using Microsoft.DataTransformationServices.Controls;
using System.Drawing;
using Microsoft.SqlServer.Dts.Runtime;

namespace ExecuteCatalogPackageTaskComplexUI
{
    2 references
    public partial class ExecuteCatalogPackageTaskComplexUIForm : DTSBaseTaskUI
    {
        private const string Title = "Execute Catalog Package Task Editor";
        private const string Description = "This task executes an SSIS package in an SSIS Catalog.";
        public static Icon taskIcon = new Icon(typeof(ExecuteCatalogPackageTask.ExecuteCatalogPackageTask), "ALCStrike.ico");

        1 reference
        public ExecuteCatalogPackageTaskComplexUIForm(TaskHost taskHost, object connections) :
         base(Title, taskIcon, Description, taskHost, connections)
        { }
    }
```

***Figure 11-14*** TaskHost no longer in error

Also, the call to the ExecuteCatalogPackageTaskComplexUIForm constructor found in the ExecuteCatalogPackageTaskComplexUI GetView method is no longer in error, as shown in Figure 11-15:

```
public ContainerControl GetView()
{
    return new ExecuteCatalogPackageTaskComplexUIForm(taskHost, connectionService);
}
```

***Figure 11-15*** No error in the ExecuteCatalogPackageTaskComplexUI GetView method

The next step is to add views to the ExecuteCatalogPackageTaskComplexUIForm class.

## Calling the Views

An SSIS task editor surfaces properties in a hierarchy: View ➤ Node ➤ Property Category ➤ Property. *Views* are “pages” on an SSIS task editor. The following section appeared in Chapter 10. We present it here as a review.

Figure 11-16 visualizes the hierarchy for the Execute SQL Task Editor. As mentioned earlier, the General view (in the green box on the left) displays the General node – represented by the propertygrid control (in the blue box on the right). The property category named "General" is shown in the red box. The Name property is shown enclosed in the yellow box, as shown in Figure 11-16:

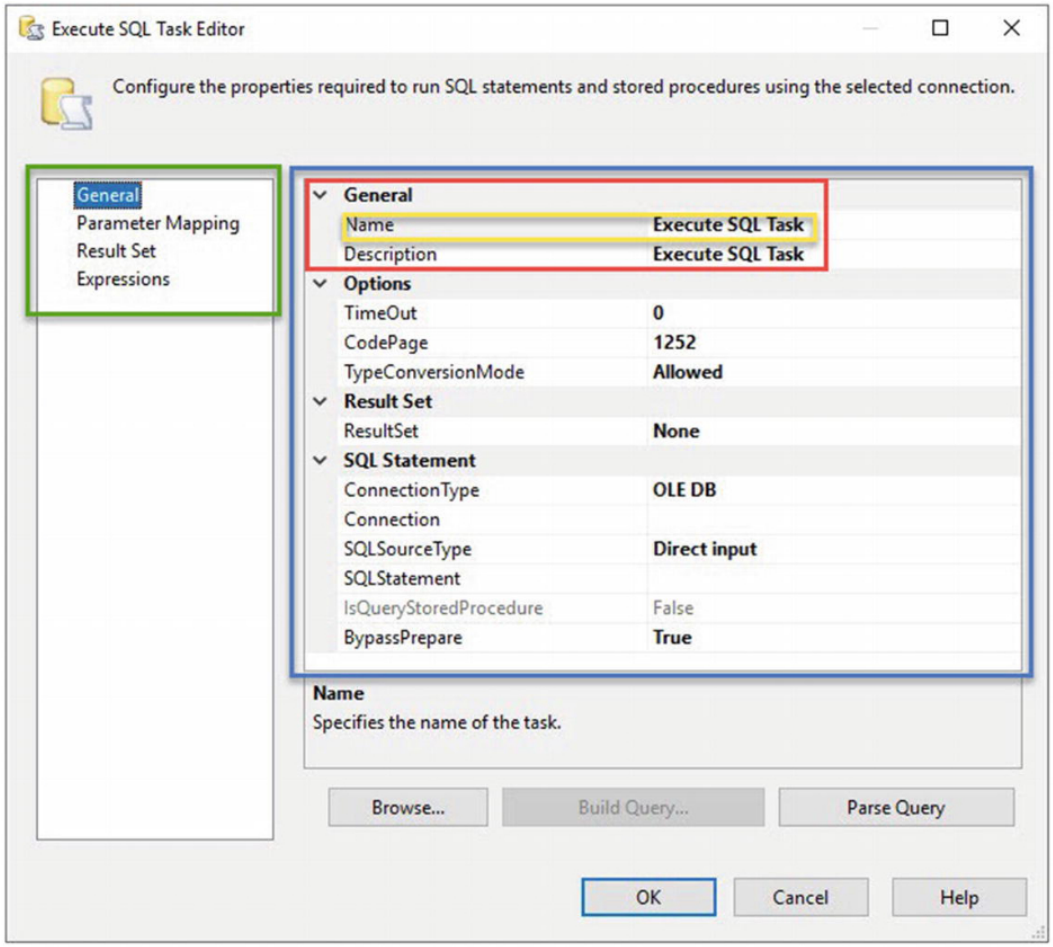



***Figure 11-16*** Visualizing View ➤ Node ➤ Property Category ➤ Property

Applied to the View ➤ Node ➤ Property Category ➤ Property hierarchy, the names of the entities for the Execute SQL Task Editor – shown in Figure 11-16 – are as follows: General (View) ➤ General (Node) ➤ General (Property Category) ➤

Name (Property). That's a lot of General's, and one – the General node – is hiding beneath the propertygrid control.

Add the code in Listing 11-12 to the ExecuteCatalogPackageTaskComplexUIForm class's constructor to call the General and Settings views:

```
// Add General view
GeneralView generalView = new GeneralView();
this.DTSTaskUIHost.AddView("General", generalView, null);
// Add Settings view
SettingsView settingsView = new SettingsView();
this.DTSTaskUIHost.AddView("Settings", settingsView, null);
```

***Listing 11-12*** Adding calls to the General and Settings views

Once added, the code should appear as shown in Figure 11-17:

```
public ExecuteCatalogPackageTaskComplexUIForm(TaskHost taskHost, object connections) :
 base(Title, taskIcon, Description, taskHost, connections)
{

    // Add General view
    GeneralView generalView = new GeneralView();
    this.DTSTaskUIHost.AddView("General", generalView, null);

    // Add Settings view
    SettingsView settingsView = new SettingsView();
    this.DTSTaskUIHost.AddView("Settings", settingsView, null);

}
```

***Figure 11-17*** ExecuteCatalogPackageTaskComplexUIForm class's constructor with calls to the General and Settings views

The GeneralView and SettingsView classes do not yet exist, hence the red squiggly lines beneath each. The next step is to create and code the view classes, starting with the GeneralView.

### Coding the GeneralView Class

## Coding the GeneralView Class

The code in this section builds part of the *View* portion of the View ➤ Node ➤ Property Category ➤ Property hierarchy. It is important to note the GeneralView class is actually a `System.Windows.Form` object, and this is why the implementation automatically added the statement `using System.Windows.Forms`. Later in this chapter, we will add standard and typical `Form` methods manually. Why not simply add GeneralView as a form? This book is written for people who are not professional software developers. Adding a class instead of adding a form is one way to emphasize the fact that forms *are* classes.

In Solution Explorer, right-click the ExecuteCatalogPackageTaskComplexUI project, hover over Add, and then click "Class…." When the Add New Item dialog displays, change "Class1.cs" to "GeneralView.cs," as shown in Figure 11-18:

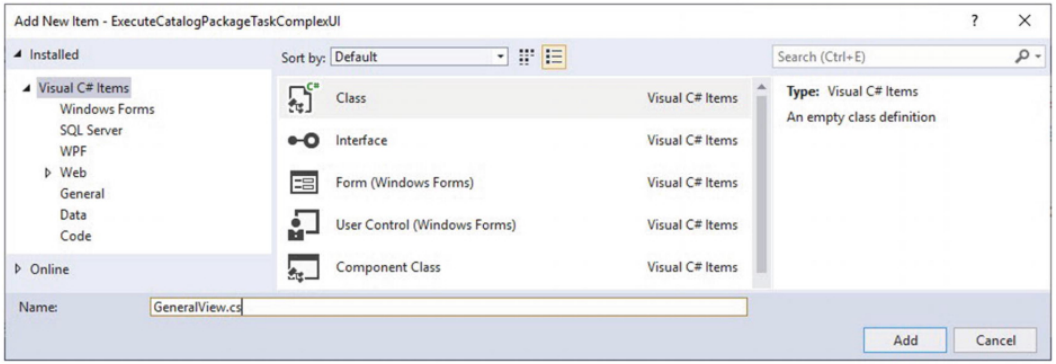



***Figure 11-18*** Adding the GeneralView class

Click the Add button to add the new GeneralView class.

Edit the class declaration using the code in Listing 11-13:

```
public partial class GeneralView : System.Windows.Forms.UserControl, IDT
```

***Listing 11-13*** GeneralView class declaration

Once edited, the code should appear as shown in Figure 11-19:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ExecuteCatalogPackageTaskComplexUI
{
    2 references
    public partial class GeneralView : System.Windows.Forms.UserControl, IDTSTaskUIView
    {
    }
}
```

***Figure 11-19*** GeneralView class declaration, edited

GeneralView is declared inheriting two interfaces:

- System.Windows.Forms.UserControl
- IDTSTaskUIView

Add a `using` statement for the `Microsoft.DataTransformationServices.Controls` assembly – in Listing 11-14 – to implement the IDTSTaskUIView interface:

```
using Microsoft.DataTransformationServices.Controls;
```

***Listing 11-14*** Using statement for IDTSTaskUIView

Adding the `using` statement clears the immediate error in the GeneralView class, but another error exists. Hovering over the red squiggly underlined IDTSTaskUIView text, the potential fixes "helper" icon displays, as shown in Figure 11-20:


***Figure 11-20*** IDTSTaskUIView interface is recognized but not yet implemented

Click the dropdown and then click "Implement interface" as shown in Figure 11-21:


***Figure 11-21*** Displaying potential fixes

Implementing the interface adds the code in Listing 11-15, and shown in Figure 11-22:

```
using System.Windows.Forms;
.
.
.
namespace ExecuteCatalogPackageTaskComplexUI
{
    public partial class GeneralView : System.Windows.Forms.UserControl,
```

```
    {
        public void OnCommit(object taskHost)
        {
            throw new NotImplementedException();
        }




        public void OnInitialize(IDTSTaskUIHost treeHost, TreeNode viewN
        {
            throw new NotImplementedException();
        }
        public void OnLoseSelection(ref bool bCanLeaveView, ref string r
        {
            throw new NotImplementedException();
        }
        public void OnSelection()
        {
            throw new NotImplementedException();
        }
        public void OnValidate(ref bool bViewIsValid, ref string reason)
        {
            throw new NotImplementedException();
        }
    }
```

***Listing 11-15*** IDTSTaskUIView implementation code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.DataTransformationServices.Controls;

namespace ExecuteCatalogPackageTaskComplexUI
{
    2 references
```

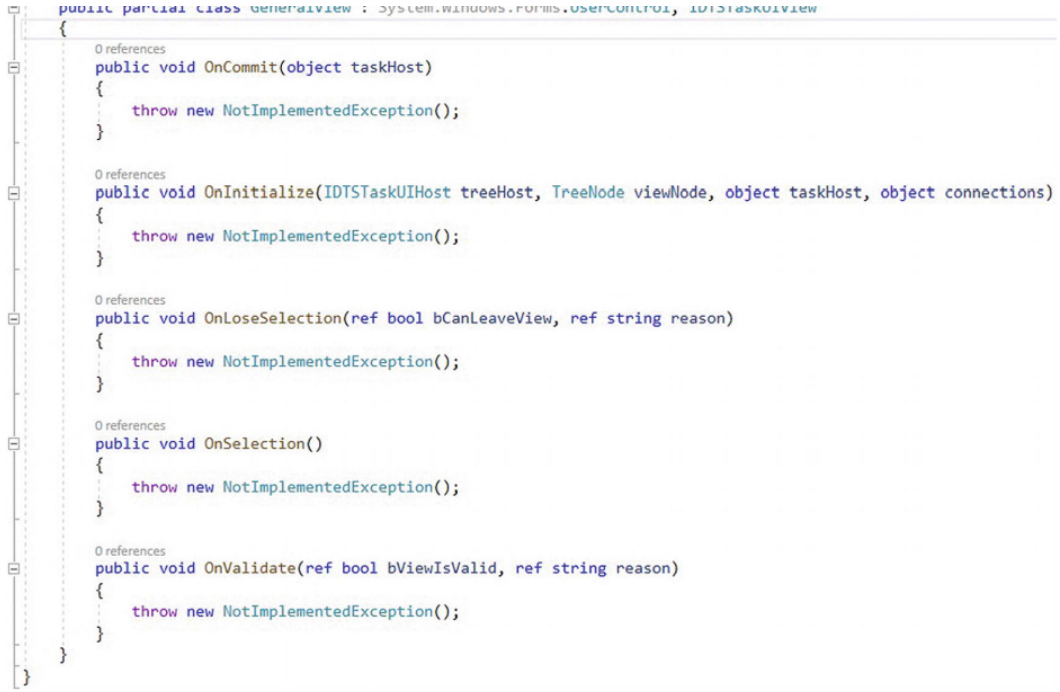



***Figure 11-22*** IDTSTaskUIView interface, implemented for GeneralView

The next step is to build out a GeneralView *Node*, the first of a few internal members (properties), a constructor, and to override the default interface implementation code.

You have options when designing the GeneralNode class. You can implement the class in a separate ".cs" file, or you may declare the class within the GeneralView.cs file. For the purposes of this example, we add the GeneralNode class to the GeneralView.cs file by adding the code in Listing **11-16**:

```
internal class GeneralNode { }
```

***Listing 11-16*** Declaring the GeneralNode class

Once added, the code should appear as shown in Figure **11-23**:


***Figure 11-23*** Declaring the GeneralNode class

Once the GeneralNode class is declared, add an initialize a private member to the GeneralView class using the code in Listing **11-17**:

```
private GeneralNode generalNode = null;
```

***Listing 11-17*** Declaring a private GeneralNode member in GeneralView

Once added, the code should appear as shown in Figure **11-24**:

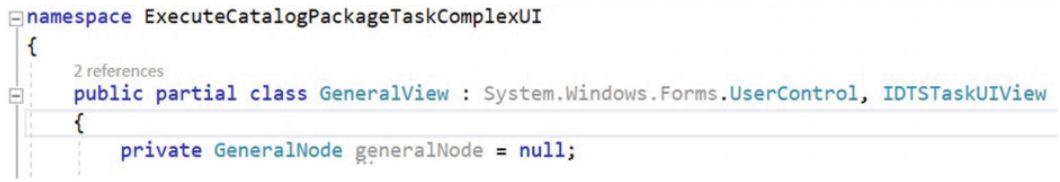



***Figure 11-24*** Declare and initialize a GeneralNode member

Declare three additional members in GeneralView using the code in Listing **11-18**:

```
private System.Windows.Forms.PropertyGrid generalPropertyGrid;
private ExecuteCatalogPackageTask.ExecuteCatalogPackageTask theTask = nu
private System.ComponentModel.Container components = null;
```

***Listing 11-18*** Declaring the PropertyGrid, ExecuteCatalogPackageTask, and Containe

GeneralView should appear as shown in Figure **11-25**:

```
private ExecuteCatalogPackageTask.ExecuteCatalogPackageTask theTask = null;
private System.ComponentModel.Container components = null;
```

***Figure 11-25*** PropertyGrid, ExecuteCatalogPackageTask, and Container members

The next step is to add a constructor and a `Dispose` method to the GeneralView class using the code in Listing **11-19**:

```
public GeneralView()
  {
    InitializeComponent();
  }
  protected override void Dispose(bool disposing)
  {
    if (disposing)
    {
      if (components != null)
      {
        components.Dispose();
      }
    }
    base.Dispose(disposing);
  }
```

***Listing 11-19*** GeneralView Constructor and Dispose

Once added, the code should appear as shown in Figure **11-26**:

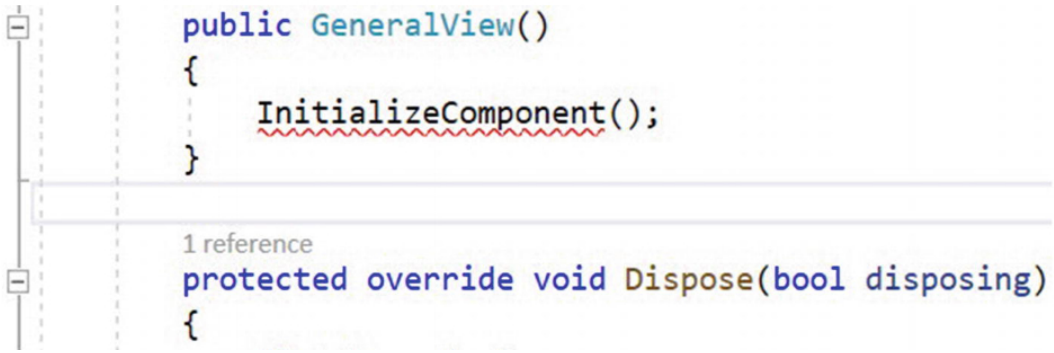



***Figure 11-26*** Adding the constructor and Dispose methods to the GeneralView

The constructor – named the same as the class (`GeneralView`) – calls `InitializeComponent`, which is not yet implemented. The `Dispose` method overrides the `base.Dispose` method, cleaning up any lingering `components` objects before disposing of the base object.

> The `InitializeComponent` method contains information about controls and objects implemented on a form. If a new form is added to a .Net Framework solution, an `InitializeComponent` method is automatically included.

The next step is to implement the `InitializeComponent` method using the code in Listing **11-20**:

```
private void InitializeComponent()
  {
    // generalPropertyGrid
    this.generalPropertyGrid = new System.Windows.Forms.PropertyGrid();
    this.SuspendLayout();
    this.generalPropertyGrid.Anchor = ((System.Windows.Forms.AnchorStyle
      | System.Windows.Forms.AnchorStyles.Bottom)
      | System.Windows.Forms.AnchorStyles.Left)
      | System.Windows.Forms.AnchorStyles.Right)));
```

```
        this.generalPropertyGrid.Location = new System.Drawing.Point(3, 0);
        this.generalPropertyGrid.Name = "generalPropertyGrid";
        this.generalPropertyGrid.PropertySort = System.Windows.Forms.Propert
        this.generalPropertyGrid.Size = new System.Drawing.Size(387, 360);
        this.generalPropertyGrid.TabIndex = 0;
        this.generalPropertyGrid.ToolbarVisible = false;
        // GeneralView
        this.Controls.Add(this.generalPropertyGrid);
        this.Name = "GeneralView";
```