A. Leonard, *Building Custom Tasks for SQL Server Integration Services*

https://doi.org/10.1007/978-1-4842-6482-9_13

13. Implement Views and Properties

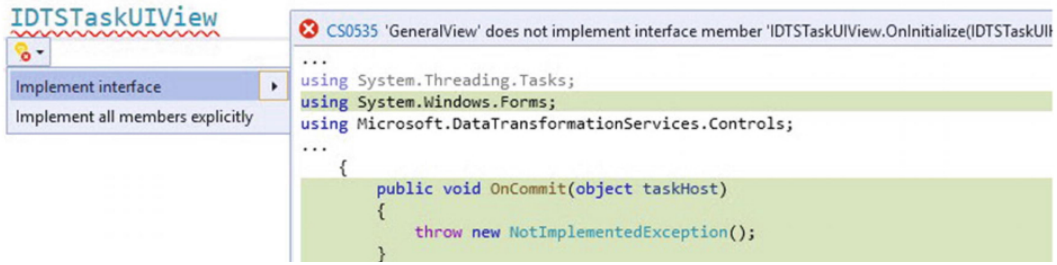
Andy Leonard[1]

(1) Farmville, VA, USA

In Chapter 12, we built and tested a minimally coded ExecuteCatalogPackageTask sporting a new “SSIS-y” editor named ExecuteCatalogPackageTaskComplexUI. In this chapter, we will implement the IDTSTaskUIView editor interfaces for GeneralView and SettingsView, add editor properties, and conduct even more tests.

Implementing the GeneralView IDTSTaskUIView Interface

In Chapter 11, we used some nifty functionality built into Visual Studio 2019 to implement the required IDTSTaskUIView interface methods for the GeneralView with a single click, as shown in Figure 13-1:



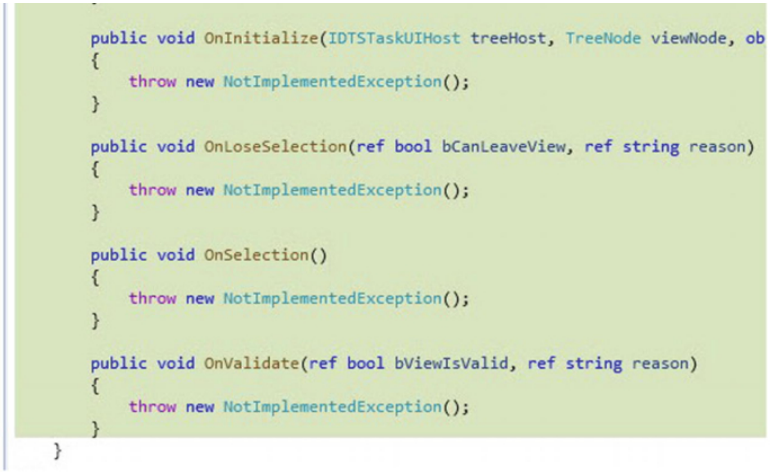


Figure 13-1 Implementing required IDTSTaskUIView interface methods for GeneralView

Near the end of the chapter, we commented out the `throw` statements so the ExecuteCatalogPackageTaskComplexUI project would build and pass some basic tests. The code now appears as shown in Listing **13-1**:

```
public void OnInitialize(IDTSTaskUIHost treeHost, TreeNode viewNode, obj
  {
    // throw new NotImplementedException();
  }
public void OnValidate(ref bool bViewIsValid, ref string reason)
  {
    // throw new NotImplementedException();
  }



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

```
    // throw new NotImplementedException();
  }
public void OnLoseSelection(ref bool bCanLeaveView, ref string reason)
  {
    // throw new NotImplementedException();
  }
```

Listing 13-1 IDTSTaskUIView interface methods for GeneralView, commented out

The code above has also been rearranged to more closely resemble an order I prefer (see the earlier note on "CDO").

The next step is to implement the `GeneralView OnInitialize` method.

Implementing GeneralView OnInitialize

Implement the `GeneralView OnInitialize` method by replacing the current `OnInitialize` method code with the code in Listing 13-2:

```
public virtual void OnInitialize(IDTSTaskUIHost treeHost
                                 , System.Windows.Forms.TreeNode viewNode
                                 , object taskHost
                                 , object connections)
{
  if (taskHost == null)
  {
throw new ArgumentNullException("Attempting to initialize the ExecuteCat
  }
  if (!((TaskHost)taskHost).InnerObject is ExecuteCatalogPackageTask.Ex
  {
  throw new ArgumentException("Attempting to initialize the ExecuteCatal
  }
  theTask = ((TaskHost)taskHost).InnerObject as ExecuteCatalogPackageTas
  this.generalNode = new GeneralNode(taskHost as TaskHost);
  generalPropertyGrid.SelectedObject = this.generalNode;
```

```
    generalNode.Name = ((TaskHost)taskHost).Name;
}
```

Listing 13-2 Implementing OnInitialize for GeneralView

The `GeneralView OnInitialize` method appears as shown in Figure 13-2:

```
public virtual void OnInitialize(IDTSTaskUIHost treeHost
                                 , System.Windows.Forms.TreeNode viewNode
                                 , object taskHost
                                 , object connections)
{
    if (taskHost == null)
    {
        throw new ArgumentNullException("Attempting to initialize the ExecuteCatalogPackageTask UI with a null TaskHost.");
    }

    if (!(((TaskHost)taskHost).InnerObject is ExecuteCatalogPackageTask.ExecuteCatalogPackageTask))
    {
        throw new ArgumentException("Attempting to initialize the ExecuteCatalogPackageTask UI with a task that is not an ExecuteCatalogPackageTask.");
    }

    theTask = ((TaskHost)taskHost).InnerObject as ExecuteCatalogPackageTask.ExecuteCatalogPackageTask;

    this.generalNode = new GeneralNode(taskHost as TaskHost);

    generalPropertyGrid.SelectedObject = this.generalNode;

    generalNode.Name = ((TaskHost)taskHost).Name;
}
```

Figure 13-2 The GeneralView OnInitialize method

Note `TaskHost` and `GeneralNode` are marked with red squiggly lines to indicate an issue with each. Hover over `TaskHost`, click the dropdown, and then click "using Microsoft.SqlServer.Dts.Runtime;" as shown in Figure 13-3:

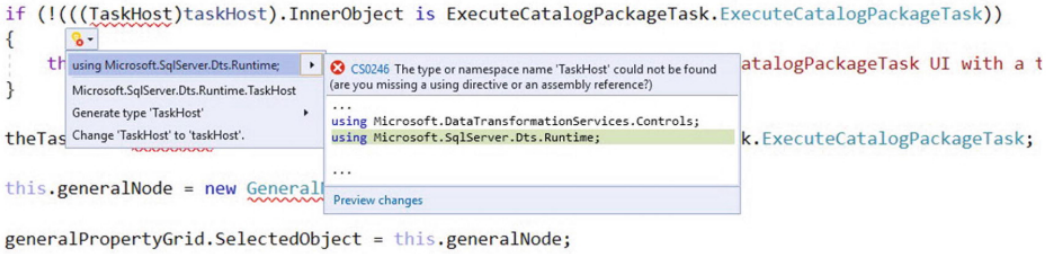


Figure 13-3 Adding "using Microsoft.SqlServer.Dts.Runtime;"

This Visual Studio automation adds a `using` directive for the `Microsoft.SqlServer.Dts.Runtime` assembly reference, clearing a number of red

squiggly lines in the `GeneralView OnInitialize` method, as shown in Figure 13-4:

```
59  public virtual void OnInitialize(IDTSTaskUIHost treeHost
60                                  , System.Windows.Forms.TreeNode viewNode
61                                  , object taskHost
62                                  , object connections)
63  {
64      if (taskHost == null)
65      {
66          throw new ArgumentNullException("Attempting to initialize the ExecuteCatalogPackageTask UI with a null TaskHost");
67      }
68
69      if (!((TaskHost)taskHost).InnerObject is ExecuteCatalogPackageTask.ExecuteCatalogPackageTask))
70      {
71          throw new ArgumentException("Attempting to initialize the ExecuteCatalogPackageTask UI with a task that is not an ExecuteCatalogPackageTask.");
72      }
73
74      theTask = ((TaskHost)taskHost).InnerObject as ExecuteCatalogPackageTask.ExecuteCatalogPackageTask;
75
76      this.generalNode = new GeneralNode(taskHost as TaskHost);
77
78      generalPropertyGrid.SelectedObject = this.generalNode;
79
80      generalNode.Name = ((TaskHost)taskHost).Name;
81  }
```

Figure 13-4 The using directive clears most design-time issues

Visual Studio line numbers are included in Figure 13-4 to aid in code functionality discussion. Lines 59–63 are the `GeneralView OnInitialize` method's declaration and arguments.

Lines 64–67 check to see if the `taskHost` member is null and, if `taskHost` is null, throw an `ArgumentNullException` that includes the message: "Attempting to initialize the ExecuteCatalogPackageTask UI with a null TaskHost."

Lines 69–72 test whether the `GeneralView taskHost` member's `InnerObject` may *not* be cast to an instance of the `ExecuteCatalogPackageTask` type. If the `GeneralView taskHost` member's `InnerObject` cannot be cast to an `ExecuteCatalogPackageTask`, the code throws an `ArgumentException` that includes the message: "Attempting to initialize the ExecuteCatalogPackageTask UI with a task that is not an ExecuteCatalogPackageTask."

If the previous "type test" succeeds, the `GeneralView theTask` member's `InnerObject` is assigned to the `ExecuteCatalogPackageTask` object on line 74.

On line 76, the `GeneralView generalNode` member is initialized (the code here is

On line 76, the `GeneralView generalNode` member is initialized (the code here is currently broken, but we will fix it soon).

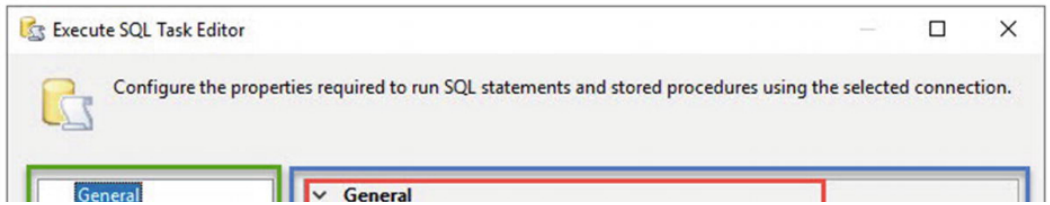
On line 78, the `GeneralView generalPropertyGrid` member is initialized as the `generalNode`.

Finally, the `generalNode.Name` is initialized as the value of the `taskHost.Name` property value on line 80. The taskHost may be thought of as the "task on the SSIS package Control Flow." This line of code helps keep the name of the ExecuteCatalogPackageTask in sync with the `taskHost.Name` property value displayed one the SSIS package Control Flow when the Execute Catalog Package Task editor is closed.

There are a lot of moving parts in the `GeneralView OnInitialize` method. A lot of what is occurring in the `OnInitialize` method is a *knitting together* of the SSIS task editor View ➤ Node ➤ Property Category ➤ Property hierarchy we originally discussed in Chapter 10.

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 13-5 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 13-5:



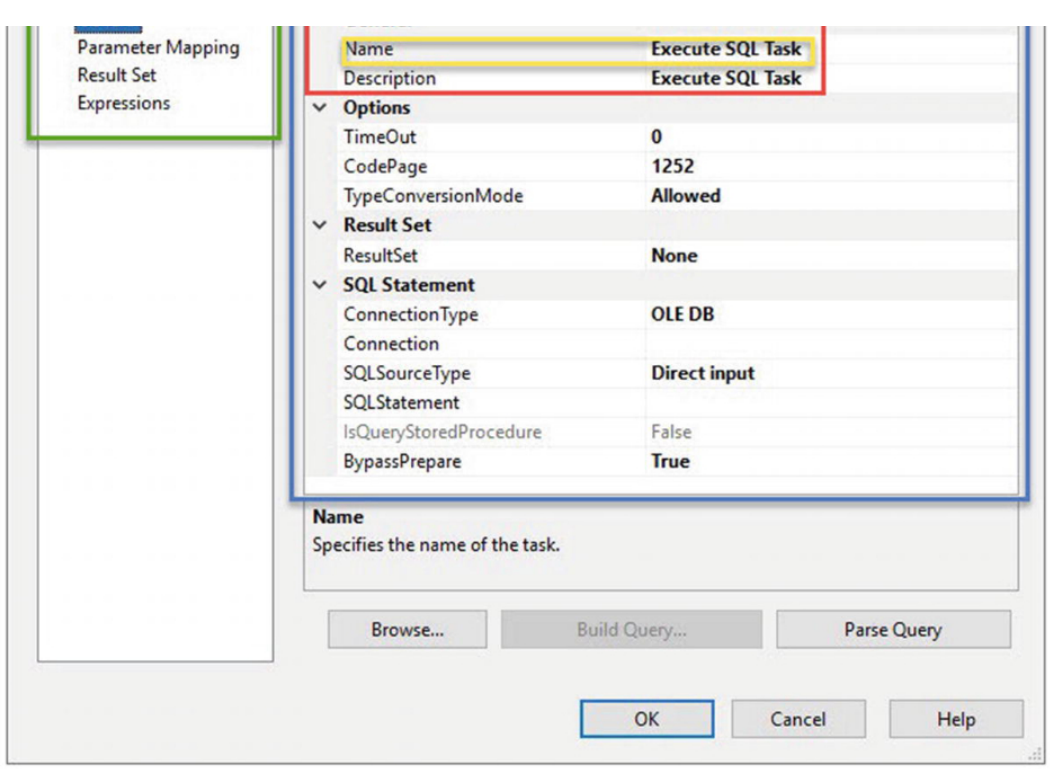


Figure 13-5 Visualizing View ➤ Node ➤ Property Category ➤ Property

Applied to the View ➤ Node ➤ Property Category ➤ Property hierarchy, the names of the entities for the Execute SQL Task Editor – shown in Figure 13-5 – are the following: 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.

The new `GeneralView OnInitialize` method is coupling the `ExecuteCatalogPackageTaskComplexUI` to the `ExecuteCatalogPackageTask` using the `GeneralView taskHost` member. Once that succeeds – and the code will throw an exception if the coupling is not possible – the `GeneralView`'s `generalNode` member is initialized (to a new `GeneralNode` object) and the `GeneralView`'s `generalPropertyGrid` is, in turn, initialized to the (new) `generalNode`.

Again, there are a *lot* of moving parts in the `GeneralView OnInitialize` method.

Implementing GeneralView OnCommit

Implement the `GeneralView OnCommit` method by replacing the current `OnCommit` method code with the code in Listing **13-3**:

```
public virtual void OnCommit(object taskHost)
{
  TaskHost th = (TaskHost)taskHost;
  th.Name = generalNode.Name.Trim();
  th.Description = generalNode.Description.Trim();
  theTask.TaskName = generalNode.Name.Trim();
  theTask.TaskDescription = generalNode.Description.Trim();
}
```

Listing 13-3 Implementing OnCommit for GeneralView

The `GeneralView OnCommit` method appears as shown in Figure **13-6**:

```
public virtual void OnCommit(object taskHost)
{
    TaskHost th = (TaskHost)taskHost;

    th.Name = generalNode.Name.Trim();
    th.Description = generalNode.Description.Trim();

    theTask.TaskName = generalNode.Name.Trim();
    theTask.TaskDescription = generalNode.Description.Trim();
}
```

Figure 13-6 The GeneralView OnCommit method

When the `Name` and `Description` members are added to the `GeneralNode` class, the red squiggly lines in Figure **13-6** – along with the errors the indicate – will resolve. Add the

`TaskName` and `TaskDescription` properties to the ExecuteCatalogPackageTask using the code in Listing 13-4 to resolve the red squiggly lines beneath the `TaskName` and `TaskDescription` properties to the ExecuteCatalogPackageTask:

```
public string TaskName { get; set; } = "Execute Catalog Package Task";
public string TaskDescription { get; set; } = "Execute Catalog Package T
```

Listing 13-4 Adding the TaskName and TaskDescription properties to the ExecuteCat

Once added, the code appears as shown in Figure 13-7:

```
public string TaskName { get; set; } = "Execute Catalog Package Task";
1 reference
public string TaskDescription { get; set; } = "Execute Catalog Package Task";
```

Figure 13-7 Adding the TaskName and TaskDescription properties

Coding the GeneralNode

`GeneralNode` contains the `ExecuteCatalogPackageTask` properties surfaced on the General page of the Execute Catalog Package Task (complex) editor. Begin by adding members using the code in Listing 13-5:

```
internal TaskHost taskHost = null;
private ExecuteCatalogPackageTask.ExecuteCatalogPackageTask task = null;
```

Listing 13-5 Add taskHost and task members to GeneralNode

The `GeneralNode` class appears as shown in Figure 13-8:

```
internal class GeneralNode
{
    internal TaskHost taskHost = null;
    private ExecuteCatalogPackageTask.ExecuteCatalogPackageTask task = null;
}
```

Figure 13-8 The GeneralNode class members

Add a constructor to the `GeneralNode` class by adding the code in Listing 13-6:

```
public GeneralNode(TaskHost taskHost)
{
  this.taskHost = taskHost;
  this.task = taskHost.InnerObject as ExecuteCatalogPackageTask.ExecuteC
}
```

Listing 13-6 Adding a GeneralNode class constructor

Once the constructor is added, the `GeneralNode` class appears as shown in Figure 13-9:

```
internal class GeneralNode
{
    internal TaskHost taskHost = null;
    private ExecuteCatalogPackageTask.ExecuteCatalogPackageTask task = null;

    1 reference
    public GeneralNode(TaskHost taskHost)
    {
        this.taskHost = taskHost;
        this.task = taskHost.InnerObject as ExecuteCatalogPackageTask.ExecuteCatalogPackageTask;
    }
}
```

Figure 13-9 GeneralNode class constructor

After the `GeneralNode` constructor is implemented, the design-time error on line 76 clears, as shown in Figure 13-10:

```
76    this.generalNode = new GeneralNode(taskHost as TaskHost);
```

Figure 13-10 No more error

The next step is to code the GeneralNode properties.

Coding the GeneralNode Properties

In this section, we reach the Property Category ➤ Property portion of the View ➤ Node ➤ Property Category ➤ Property hierarchy. Properties of nodes are decorated to indicate property category and description. In the editor, the value of the property is paired with the property name, and this value is passed to the instantiation of the task. The pairing may be thought of as similar to a key-value mapping where the key is composed of the View ➤ Node ➤ Property Category ➤ Property hierarchy and the value is the value supplied by the SSIS developer during task (instance) edit. An example of a value is circled in Figure 13-11:

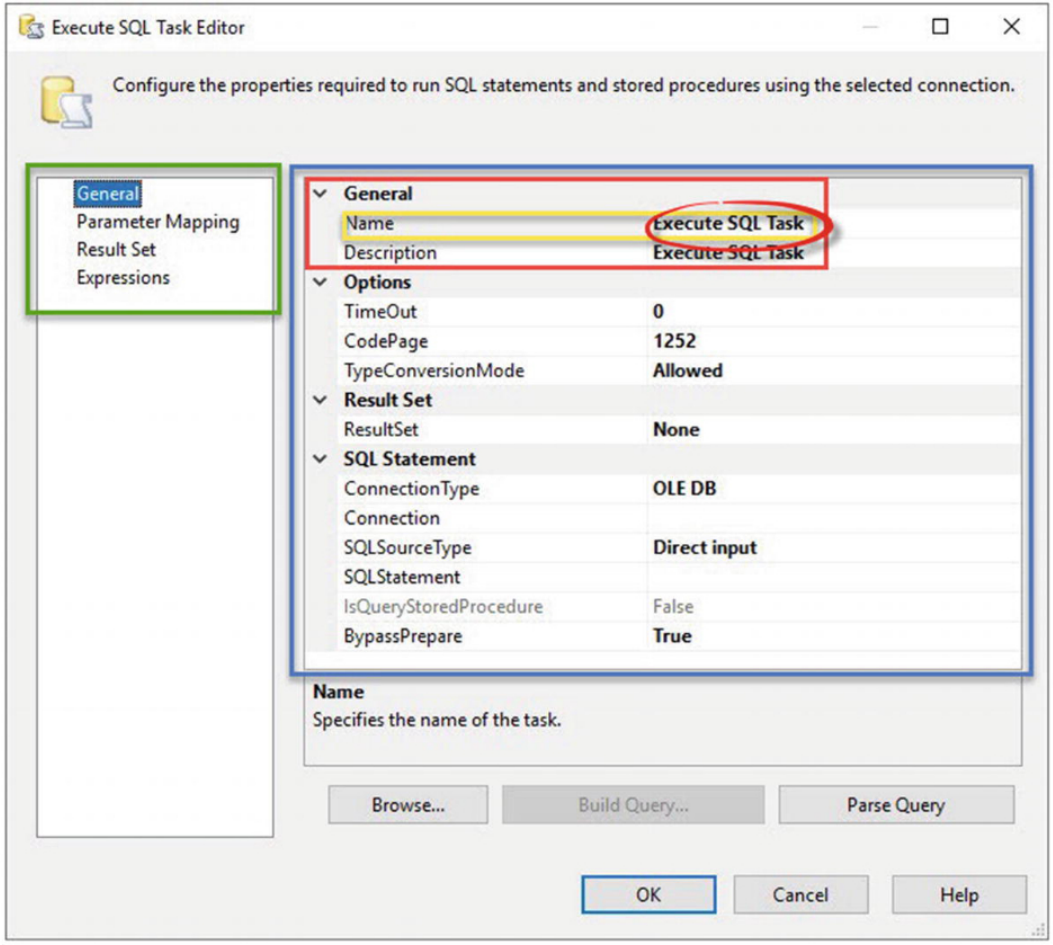


Figure 13-11 The value of the key-value pair

In the SSIS package, the value is configured during editing by an SSIS developer. When

In the SSIS package, the value is configured during editing by an SSIS developer. When the SSIS developer clicks the OK button on the editor, the `OnCommit` method fires and stores the key-value property configurations in the SSIS package's XML, as shown in Figure 13-12:

```
<DTS:Executables>
  <DTS:Executable
    DTS:refId="Package\Execute SQL Task"
    DTS:CreationName="Microsoft.ExecuteSQLTask"
    DTS:Description="Execute SQL Task"
    DTS:DTSID="{B12AF18C-77FD-4164-9BA4-681FBA10D49D}"
    DTS:ExecutableType="Microsoft.ExecuteSQLTask"
    DTS:LocaleID="-1"
    DTS:ObjectName="Execute SQL Task"
    DTS:TaskContact="Execute SQL Task; Microsoft Corporation;
    DTS:ThreadHint="0">
    <DTS:Variables />
```

Figure 13-12 XML for an SSIS Execute SQL Task

In our sample Execute SQL Task, the `Name` property value is "Execute SQL Task." The property value is stored in an attribute named `DTS:ObjectName`. At runtime, SSIS XML is loaded, interpreted, and executed, but it all starts with the SSIS developer opening the task editor and then configuring the value portion of key-value pairs, in which the keys are visually surfaced in editor views that make up the View ➤ Node ➤ Property Category ➤ Property hierarchy.

To add the Name property of our custom SSIS task, add the code in Listing 13-7 to the GeneralNode:

```
[
  Category("General"),
  Description("Specifies the name of the task.")
]
public string Name {
```

```
    get { return taskHost.Name; }
    set {
      if ((value == null) || (value.Trim().Length == 0))
      {
        throw new ApplicationException("Task name cannot be empty");
      }
      taskHost.Name = value;
    }
  }
```

Listing 13-7 Adding the Name property to the GeneralNode

When added, the code appears as shown in Figure **13-13**:

```
[
Category("General"),
Description("Specifies the name of the task.")
]
1 reference
public string Name {
    get { return taskHost.Name; }
    set {
        if ((value == null) || (value.Trim().Length == 0))
        {
            throw new ApplicationException("Task name cannot be empty");
        }

        taskHost.Name = value;
    }
}
```

Figure 13-13 Adding the Name property

The red squiggly lines under the decorations indicate an issue. Hovering over the code presents options to address the issue, as shown in Figure **13-14**:

```
[
Category("General"),
D  ription("Specifies the name of the task.")
```

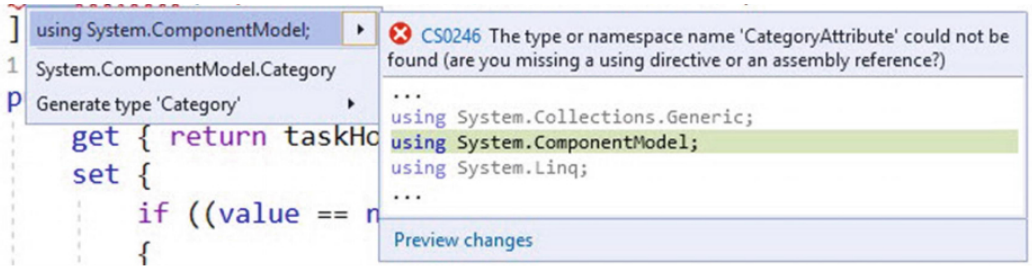


Figure 13-14 Addressing the decoration issue

Adding the `using` directive resolves the issue (you have to love Visual Studio 2019!), as shown in Figure **13-15**:

```
117  [
118  Category("General"),
119  Description("Specifies the name of the task.")
120  ]
     1 reference
121  public string Name {
122      get { return taskHost.Name; }
123      set {
124          if ((value == null) || (value.Trim().Length == 0))
125          {
126              throw new ApplicationException("Task name cannot be empty");
127          }
128
129          taskHost.Name = value;
130      }
131  }
```

Figure 13-15 The Name property with issues resolved

The Category and Description decorations shown on lines 117-120 are used by the propertygrid control for Property Category – which may hold one or more properties – and the Property Description displayed at the bottom of the editor when the property is selected during edit. The Property Category is set to `General` in this case; the Property Description is set to `Specifies the name of the task`. The Property Name is derived from the name of the view member – `Name` in this case.

Line 122 contains a `get` statement that returns the value of the `Name` property from the `taskHost.Name` property, or the `Name` property of the `taskHost`. While the statement is relatively short, there are a few moving parts. The `GeneralNode taskHost` member points back to the `ExecuteCatalogPackageTask` class. The `taskHost.Name` property value that the `GeneralNode Name` property is `get`ting here is the `ExecuteCatalogPackageTask.Name` property value. Where is the `ExecuteCatalogPackageTask.Name` property value stored? In the SSIS package XML.

Lines 123–130 contain the `set` functionality for the property. Lines 124–127 check to see if the property value is null or empty and, if so, throws an `ApplicationException` that returns the message "`Task name cannot be empty`".

If there is no exception, the value of the taskHost.Name is set on Line 129.

Add the Description property to the `GeneralNode` using the code in Listing **13-8**:

```
[
  Category("General"),
  Description("Specifies the description for this task.")
]
public string Description {
  get { return taskHost.Description; }
  set { taskHost.Description = value; }
}
```

Listing 13-8 Adding the GeneralNode Description property

When added, the code appears as shown in Figure **13-16**:

```
]
1 reference
public string Description {
    get { return taskHost.Description; }
    set { taskHost.Description = value; }
}
```

Figure 13-16 The Description property

When the Name and Description properties are added, errors in the OnCommit method clear (compare to Figure **13-6**), as shown in Figure **13-17**:

```
public virtual void OnCommit(object taskHost)
{
    TaskHost th = (TaskHost)taskHost;

    th.Name = generalNode.Name.Trim();
    th.Description = generalNode.Description.Trim();

    theTask.TaskName = generalNode.Name.Trim();
    theTask.TaskDescription = generalNode.Description.Trim();
}
```

Figure 13-17 OnCommit errors cleared

Testing GeneralView

Build the solution and then open a test SSIS project. Add an Execute Catalog Package Task to the control flow and open the editor. Observe the General page, as shown in Figure **13-18**:

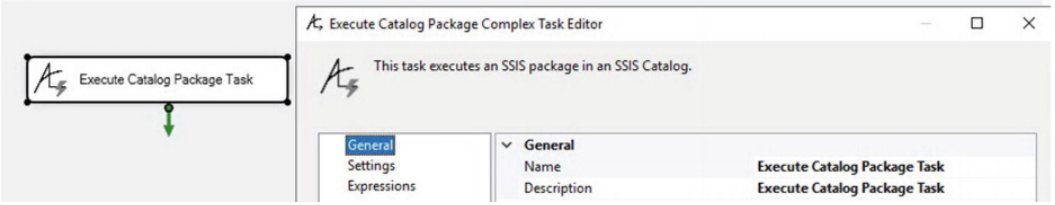


Figure 13-18 The GeneralView in action

The next step is to code the SettingsView.

Implementing the SettingsView IDTSTaskUIView Interface

In Chapter 11, we used the same nifty functionality built into Visual Studio 2019 to implement the required IDTSTaskUIView interface methods for the SettingsView as we used to implement the GeneralView interface, as shown in Figure 13-19:

Figure 13-19 Implementing required IDTSTaskUIView interface methods for SettingsView

Near the end of the chapter, we commented out the `throw` statements so the ExecuteCatalogPackageTaskComplexUI project would build and pass some basic tests. The code – rearranged to suit my CDO – now appears as shown in Listing 13-9:

```
public void OnInitialize(IDTSTaskUIHost treeHost, TreeNode viewNode, obj
  {
    // throw new NotImplementedException();
  }
public void OnValidate(ref bool bViewIsValid, ref string reason)
  {
    // throw new NotImplementedException();
  }
public void OnCommit(object taskHost)
  {
    // throw new NotImplementedException();
  }
public void OnSelection()
  {
    // throw new NotImplementedException();
  }
public void OnLoseSelection(ref bool bCanLeaveView, ref string reason)
  {



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

Listing 13-9 IDTSTaskUIView interface methods for SettingsView, commented out

The next step is to implement the `SettingsView OnInitialize` method.

Implementing SettingsView OnInitialize

Implement the `SettingsView OnInitialize` method by replacing the current `OnInitialize` method code with the code in Listing **13-10**:

```
public virtual void OnInitialize(IDTSTaskUIHost treeHost
```

```
                                , System.Windows.Forms.TreeNode viewNode
                                , object taskHost
                                , object connections)
    {
      if (taskHost == null)
        {
          throw new ArgumentNullException("Attempting to initialize the Ex
        }
        if (!(((TaskHost)taskHost).InnerObject is ExecuteCatalogPackageTas
        {
          throw new ArgumentException("Attempting to initialize the Execut
        }
        theTask = ((TaskHost)taskHost).InnerObject as ExecuteCatalogPackag
        this.settingsNode = new SettingsNode(taskHost as TaskHost, connect
        settingsPropertyGrid.SelectedObject = this.settingsNode;
    }
```

Listing 13-10 Implementing OnInitialize for SettingsView

As when coding the `GeneralView`, repair the missing `using` directive to clear the `TaskHost` design-time warning (see Figure 13-3). We will clear the `SettingsNode` design-time warning – the red squiggly line – later.

The `SettingsView OnInitialize` method should now appear as shown in Figure 13-20:

```
59  public virtual void OnInitialize(IDTSTaskUIHost treeHost
60                                  , System.Windows.Forms.TreeNode viewNode
61                                  , object taskHost
62                                  , object connections)
63  {
64      if (taskHost == null)
65      {
66          throw new ArgumentNullException("Attempting to initialize the ExecuteCatalogPackageTask UI with a null TaskHost");
67      }
68
69      if (!(((TaskHost)taskHost).InnerObject is ExecuteCatalogPackageTask.ExecuteCatalogPackageTask))
70      {
71          throw new ArgumentException("Attempting to initialize the ExecuteCatalogPackageTask UI with a task that is not a ExecuteCatalogPackageTask.");
72      }
73
74      theTask = ((TaskHost)taskHost).InnerObject as ExecuteCatalogPackageTask.ExecuteCatalogPackageTask;
75
76      this.settingsNode = new SettingsNode(taskHost as TaskHost, connections);
77
78      settingsPropertyGrid.SelectedObject = this.settingsNode;
79  }
```

Figure 13-20 The SettingsView OnInitialize method

Visual Studio line numbers are included in Figure 13-20 to aid in code functionality discussion. Lines 59–63 are the `SettingsView OnInitialize` method's declaration and arguments.

Lines 64–67 check to see if the `taskHost` member is null and, if `taskHost` is null, throw an `ArgumentNullException` that includes the message: "Attempting to initialize the ExecuteCatalogPackageTask UI with a null TaskHost."

Lines 69–72 test whether the `SettingsView taskHost` member's `InnerObject` may *not* be cast to an instance of the `ExecuteCatalogPackageTask` type. If the `SettingsView taskHost` member's `InnerObject` cannot be cast to an `ExecuteCatalogPackageTask`, the code throws an `ArgumentException` that includes the message: "Attempting to initialize the ExecuteCatalogPackageTask UI with a task that is not an ExecuteCatalogPackageTask."

If the previous "type test" succeeds, the `SettingsView theTask` member's `InnerObject` is assigned to the `ExecuteCatalogPackageTask` object on line 74.

On line 76, the `SettingsView generalNode` member is initialized (the code here is currently broken, but we will fix it soon).

Finally, the `SettingsView settingsPropertyGrid` member is initialized as the `settingsNode`.

As with the `GeneralView`, there are a lot of moving parts in the `SettingsView OnInitialize` method. The new `SettingsView OnInitialize` method is coupling the `ExecuteCatalogPackageTaskComplexUI` to the `ExecuteCatalogPackageTask` using the `SettingsView taskHost` member. Once that succeeds – and the code will throw an exception if the coupling is not possi-

ble – the `SettingsView`'s `settingsNode` member is initialized (to a new `SettingsNode` object), and the `SettingsView`'s `settingsPropertyGrid` is, in turn, initialized to the (new) `settingsNode`.

Adding ExecuteCatalogPackageTask Properties

Before implementing `SettingsView OnCommit`, we need to add a few properties to the `ExecuteCatalogPackageTask` class . Open the ExecuteCatalogPackageTask.cs file in the ExecuteCatalogPackageTask project, and then add the member declarations in Listing 13-11:

```
public string ConnectionManagerName { get; set; } = String.Empty;
public bool Synchronized { get; set; } = false;
public bool Use32bit { get; set; } = false;
public string LoggingLevel { get; set; } = "Basic";
```

Listing 13-11 Adding ExecuteCatalogPackageTask members

When added, the new `ExecuteCatalogPackageTask` members appear as shown in Figure 13-21:

```
public class ExecuteCatalogPackageTask : Microsoft.SqlServer.Dts.Runtime.Task
{
    // Public key: e86e33313a45419e
    //             e4f20b7aa35f375d -- key for simple UI
    //             a68173515d1ee3e3 -- key for complex UI

    18 references
    public string TaskName { get; set; } = "Execute Catalog Package Task";
    3 references
    public string TaskDescription { get; set; } = "Execute Catalog Package Task";
    5 references
    public string ConnectionManagerName { get; set; } = String.Empty;
    6 references
    public bool Synchronized { get; set; } = false;
    3 references
    public bool Use32bit { get; set; } = false;
    3 references
    public string LoggingLevel { get; set; } = "Basic";
    2 references
```

```
public string ReferenceName { get; set; } = "NULL (-1)";
4 references
public long ReferenceID { get; set; } = -1;
22 references
public string ServerName { get; set; } = String.Empty;
12 references
private Server catalogServer { get; set; } = null;
```

Figure 13-21 New ExecuteCatalogPackageTask members

`SettingsView` surfaces the `ExecuteCatalogPackageTask` properties configured on the Settings page of the Execute Catalog Package Task (complex) editor. In this section, we introduce complexity into the complex editor, so we will take an incremental approach to coding, building, and testing. We begin by adding the FolderName, ProjectName, and PackageName properties.

Implementing SettingsView OnCommit for FolderName, ProjectName, and PackageName Properties

Return to the ExecuteCatalogPackageTaskComplexUI project and implement the `SettingsView OnCommit` method by replacing the current `OnCommit` method code with the code in Listing **13-12**:

```
public virtual void OnCommit(object taskHost)
  {
    theTask.PackageFolder = settingsNode.FolderName;
    theTask.PackageProject = settingsNode.ProjectName;
    theTask.PackageName = settingsNode.PackageName;
  }
```

Listing 13-12 Implementing OnCommit for SettingsView

The `SettingsView OnCommit` method appears as shown in Figure **13-22**:

```
public virtual void OnCommit(object taskHost)
{
```

```
    theTask.PackageFolder = settingsNode.FolderName;
    theTask.PackageProject = settingsNode.ProjectName;
    theTask.PackageName = settingsNode.PackageName;
}
```

Figure 13-22 The SettingsView OnCommit method for FolderName, ProjectName, and PackageName Properties

The next step is to code `SettingsNode` for the FolderName, ProjectName, and PackageName properties.

Coding SettingsNode for FolderName, ProjectName, and PackageName Properties

Begin by adding FolderName, ProjectName, and PackageName members to SettingsNode using the code in Listing **13-13**:

```
internal ExecuteCatalogPackageTask.ExecuteCatalogPackageTask _task = nul
private TaskHost _taskHost = null;
```

Listing 13-13 Add members to SettingsNode

The `SettingsNode` class appears as shown in Figure **13-23**:

```
internal class SettingsNode
{
    internal ExecuteCatalogPackageTask.ExecuteCatalogPackageTask _task = null;
    private TaskHost _taskHost = null;
}
```

Figure 13-23 The SettingsNode class members

Add a constructor to the `SettingsNode` class by adding the code in Listing **13-14**:

```
public SettingsNode(TaskHost taskHost
```

```
                    , object connections)
    {
      _taskHost = taskHost;
      _task = taskHost.InnerObject as ExecuteCatalogPackageTask.ExecuteCat
    }
```

Listing 13-14 Adding a SettingsNode class constructor