A. Leonard, *Building Custom Tasks for SQL Server Integration Services*
https://doi.org/10.1007/978-1-4842-6482-9_12

12. Editor Integration

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

In Chapter 11, we minimally coded the ExecuteCatalogPackageTaskComplexUI. This chapter focuses on the operations required to couple the new complex editor (ExecuteCatalogPackageTaskComplexUI) with the ExecuteCatalogPackageTask, similar to the coupling we achieved with the previous editor in Chapters 5 and 6.

11. Minimal Coding for the Complex Editor
12. Editor Integration
Building Custom Tasks for SQL Server Integration Services: The Power of .NET for ETL for SQL Server 2019 and Beyond
13. Implement Views and Properties

Chapter 5 walks through the process of signing an assembly. In order to use the assembly in the Global Assembly Cache (GAC), the assembly must be signed. The process described in this section is nearly identical to the process described in Chapter 5.

Begin by adding a new text file named Notes.txt to the ExecuteCatalogPackageTaskComplexUI project. Right-click the ExecuteCatalogPackageTaskComplexUI project in Solution Explorer, hover over Add, and then click "New Item…." When the Add New Item dialog displays, click the General category and then click "Text File." Rename "TextFile1.txt" to "Notes.txt," as shown in Figure 12-1:

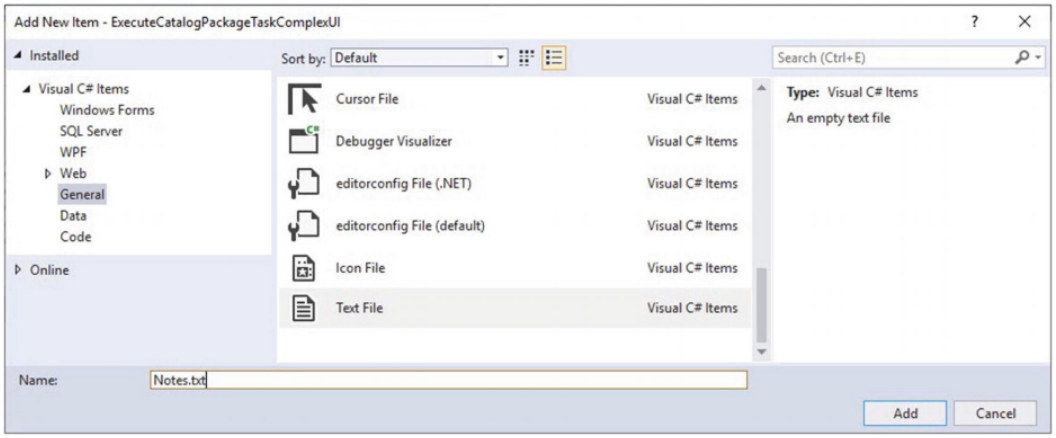


Figure 12-1 Adding the Notes.txt file

Click the Add button to add the Notes.txt file to the ExecuteCatalogPackageTaskComplexUI project. In my solution, I expanded the ExecuteCatalogPackageTask project in Solution Explorer, opened the Notes.txt file contained therein, selected all contents, copied the contents, and then pasted the clipboard to the new Notes.txt file in the ExecuteCatalogPackageTaskComplexUI project. I then edited the pasted contents, replacing "ExecuteCatalogPackageTask" with "ExecuteCatalogPackageTaskComplexUI," as shown in Figure 12-2:

```
-- key generation
"C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\sn.exe" -k key.snk
"C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\sn.exe" -p key.snk public.out
"C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\sn.exe" -t public.out

 -- register
 "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\gacutil.exe" -if "E:\Program Files (x86)\Microsoft SQL Server\150\DTS\Tasks\ExecuteCatalogPackageTaskComplexUI.dll"
  -- unregister
"C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\gacutil.exe" -u ExecuteCatalogPackageTaskComplexUI

 -- build output path
E:\Program Files (x86)\Microsoft SQL Server\150\DTS\Tasks\
```

Figure 12-2 Copy, paste, edit Notes.txt from the ExecuteCatalogPackageTask project

Creating the Key

You will need to locate sn.exe on your server to complete the next three steps.

Open a command prompt window and enter the code in Listing 12-1 (which should appear near the top of your Notes.txt file):

```
"C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tool
```

Listing 12-1 Creating the key.snk file

Once executed, the results should appear similar to those shown in Figure 12-3:

Figure 12-3 Creating the key.snk file

To write the public key to a file named "public.out," paste the next line of code from the Notes.txt file, which is Listing 12-2:

```
"C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tool
```

Listing 12-2 Writing the public key to public.out

The output should appear as shown in Figure 12-4:

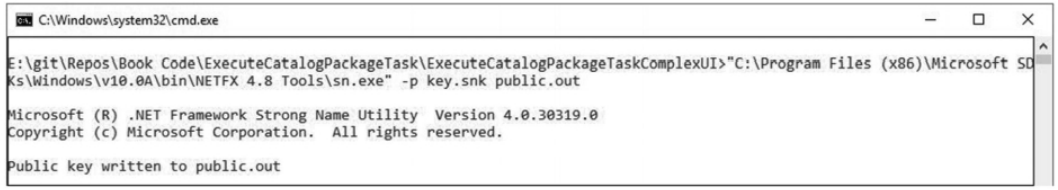


Figure 12-4 Writing the public key to public.out

Extract the public key token from the public.out file by executing the code in Listing 12-3:

```
"C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tool
```

Listing 12-3 Extracting the public key token

The output should appear similar to Figure 12-5:

Figure 12-5 Public key token, extracted

Select the public key token and copy the selection to the clipboard. Paste the clipboard results – along with some text to help identify the public key token – in the ExecuteCatalogPackageTaskComplexUI class, as shown in Figure 12-6:

```
namespace ExecuteCatalogPackageTaskComplexUI
{
    0 references
    public class ExecuteCatalogPackageTaskComplexUI : IDtsTaskUI
    {
        // Public key token: a68173515d1ee3e3
```

Figure 12-6 Storing the public key token in a code comment

The next step is to sign the ExecuteCatalogPackageTaskComplexUI assembly.

Sign the ExecuteCatalogPackageTaskComplexUI Assembly

In Solution Explorer, double-click the ExecuteCatalogPackageTaskComplexUI Properties folder. Click on the Signing page, check the “Sign the assembly” checkbox, and then browse for the key.snk file, as shown in Figure 12-7:

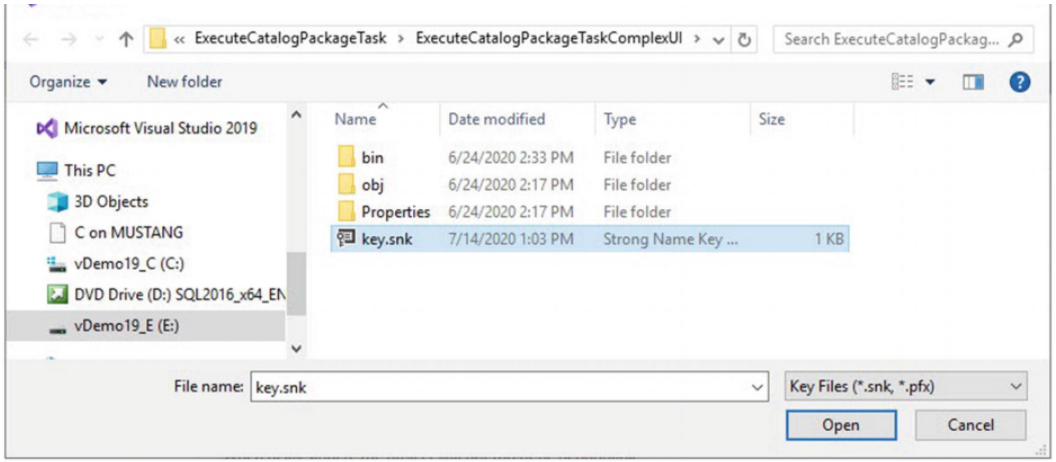


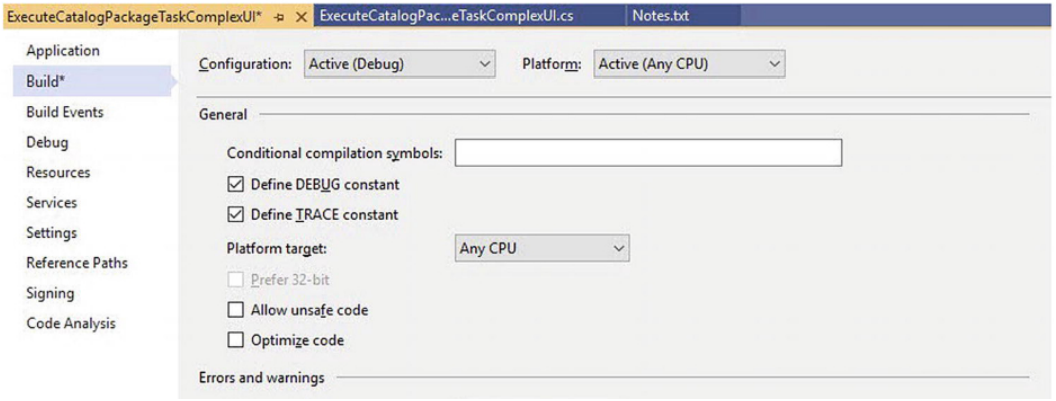
Figure 12-7 Browsing for the key.snk file

Click the Open button to use the key.snk file to sign the ExecuteCatalogPackageTaskComplexUI assembly.

The next step is to configure the Build output path location.

Configure the Build Output Path

Return to the Notes.txt file and copy the "build output path" location to the clipboard. Click the Build page on the ExecuteCatalogPackageTaskComplexUI properties page, and then paste the clipboard contents into the Output path textbox, as shown in Figure 12-8:



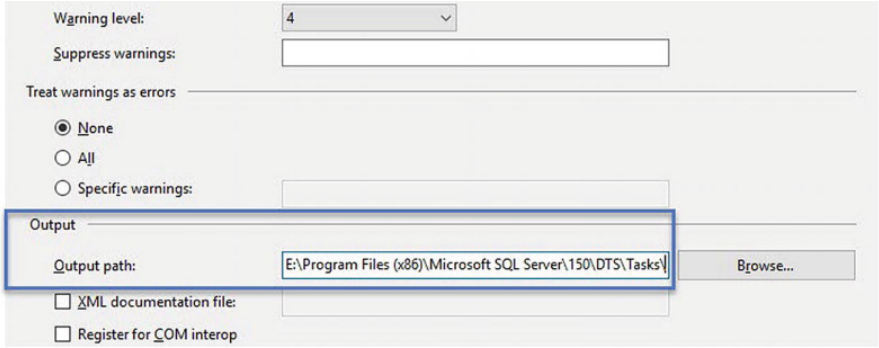


Figure 12-8 Configuring the Build Output path property

The next step is to configure build events for the ExecuteCatalogPackageTaskComplexUI assembly.

Configure Build Events

You will need to locate gacutil.exe on your server to complete the next two steps.

Return to the Notes.txt file and copy the “unregister” command to the clipboard. Click the Build Events page on the ExecuteCatalogPackageTaskComplexUI properties page, and then click the “Edit Pre-build…” button to open the Pre-build Event Command Line dialog. Paste the clipboard contents into the Pre-build Event Command Line textbox, as shown in Figure 12-9:

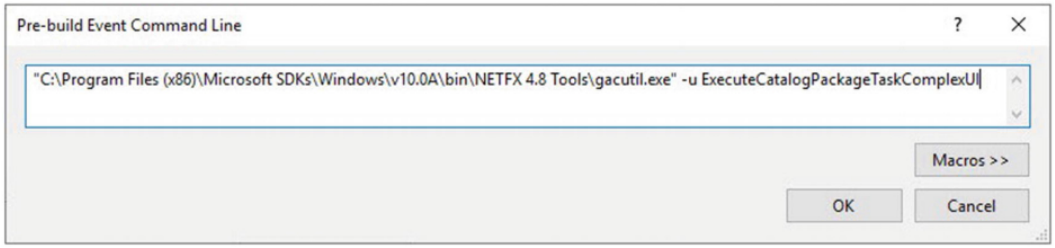


Figure 12-9 Pre-build event command line

Click the OK button to close the Pre-build Event Command Line dialog. Return to the Notes.txt file and copy the “register” command to the clipboard. Click the Build Events page

on the ExecuteCatalogPackageTaskComplexUI properties page, and then click the "Edit Post-build..." button to open the Post-build Event Command Line dialog. Paste the clipboard contents into the Post-build Event Command Line textbox, as shown in Figure 12-10:

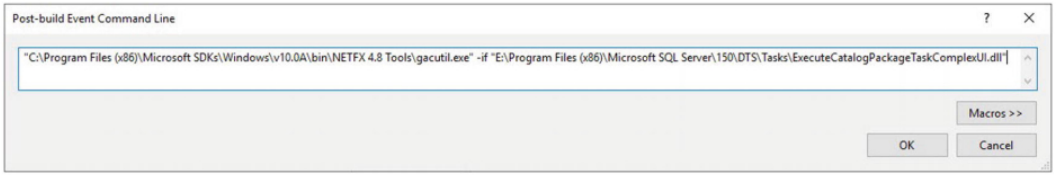


Figure 12-10 Post-build event command line

The next step is to edit the ExecuteCatalogPackageTask to use the newly signed assembly.

Editing the ExecuteCatalogPackageTask

Copy the public key token for the ExecuteCatalogPackageTaskComplexUI assembly onto the clipboard. In Solution Explorer, expand ExecuteCatalogPackageTask and open the ExecuteCatalogPackageTask.cs assembly file. Edit the DtsTask decoration for the ExecuteCatalogPackageTask constructor, replacing *your* public key token with the public key token I show in Listing 12-4:

```
[DtsTask(
  TaskType = "DTS150"
, DisplayName = "Execute Catalog Package Task"
, IconResource = "ExecuteCatalogPackageTask.ALCStrike.ico"
, Description = "A task to execute packages stored in the SSIS Catalog."
, UITypeName = "ExecuteCatalogPackageTaskComplexUI.ExecuteCatalogPackage
, TaskContact = "ExecuteCatalogPackageTask; Building Custom Tasks for SQ
```

Listing 12-4 Editing the DtsTask decoration for the ExecuteCatalogPackageTask construc

Once edited, the DtsTask decoration for the ExecuteCatalogPackageTask constructor should appear similar to Figure 12-11:

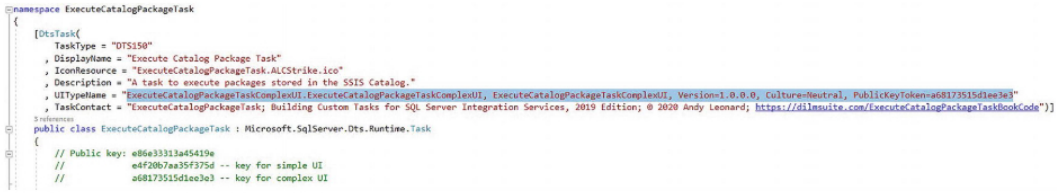


Figure 12-11 Editing the DtsTask decoration for the ExecuteCatalogPackageTask constructor

The next step is to build and test the custom task solution

Build the Solution

Click the Build menu in Visual Studio, and then click "Build Solution," as shown in Figure 12-12:

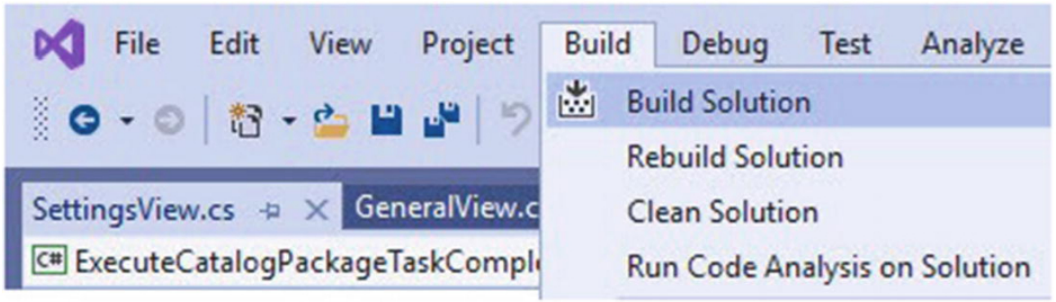
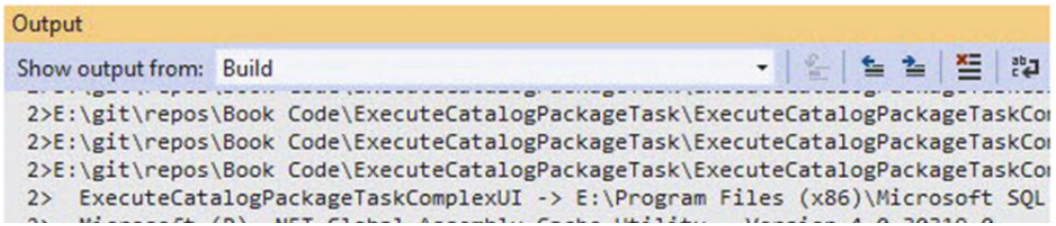


Figure 12-12 Building the ExecuteCatalogPackageTask solution

If all goes as planned, the Visual Studio Output window should display similar to Figure 12-13:



```
2>  Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.0
2>  Copyright (c) Microsoft Corporation.  All rights reserved.
2>
2>  Assembly successfully added to the cache
========== Build: 2 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Figure 12-13 Build succeeded

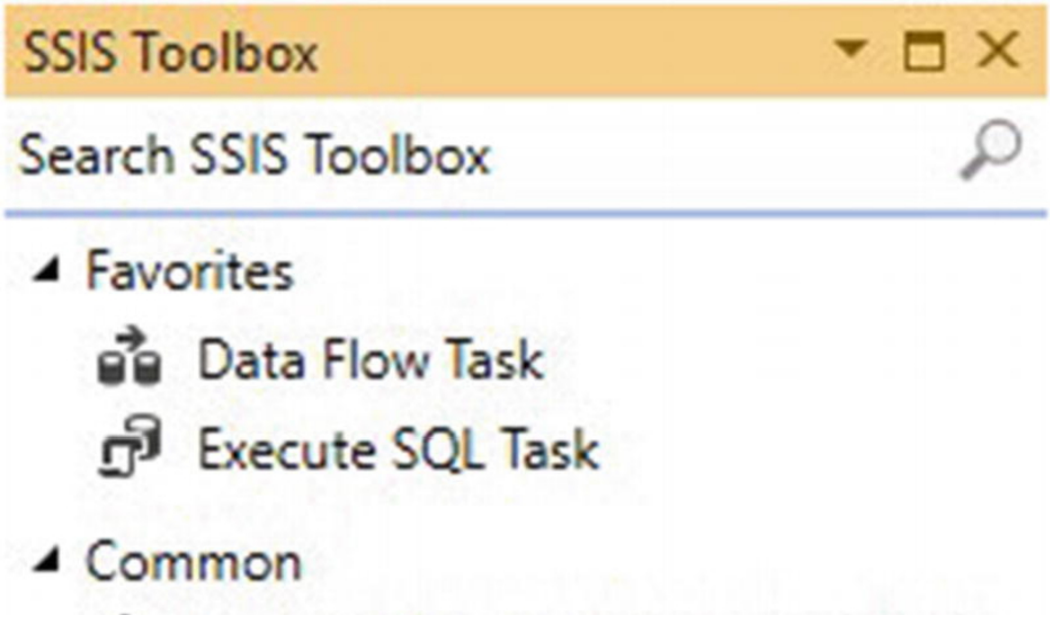
The very first test of a .Net solution is: Does it build? This solution builds.

Test the Custom SSIS Task

Test the solution by creating (or opening) a test SSIS project. The tests are

1. Does the custom SSIS task show up in the SSIS Toolbox?
2. Can an SSIS developer add the custom task to a package without error?

When an SSIS package is open in the Visual Studio IDE (Integrated Development Environment), the SSIS Toolbox may be opened to check for condition 1, as shown in Figure 12-14:



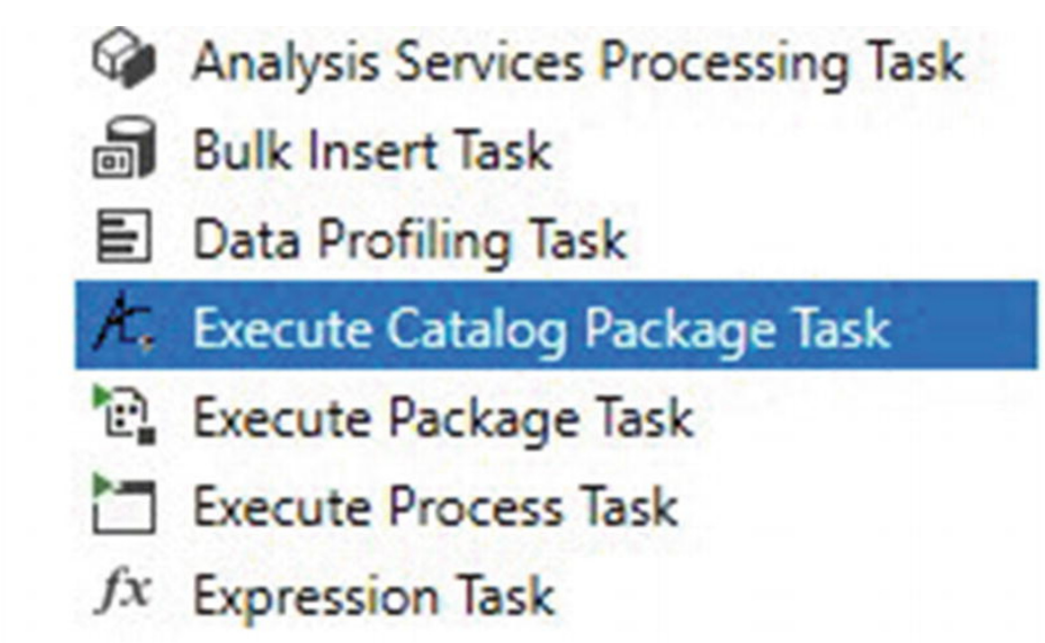


Figure 12-14 The Execute Catalog Package Task shows up in the SSIS Toolbox

Drag the Execute Catalog Package Task onto the SSIS package's Control Flow to see if the Execute Catalog Package Task may be added to an SSIS package without error, as shown in Figure 12-15:

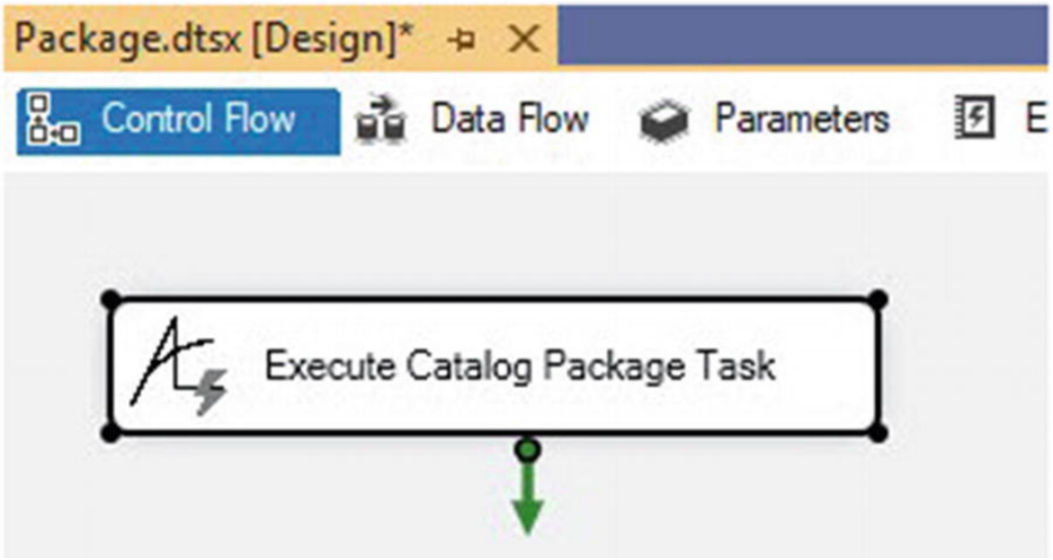


Figure 12-15 The Execute Catalog Package Task may be added to an SSIS package without error

Finally, double-click the Execute Catalog Package Task to see if it is possible to open the editor without error, as shown in Figure 12-16:

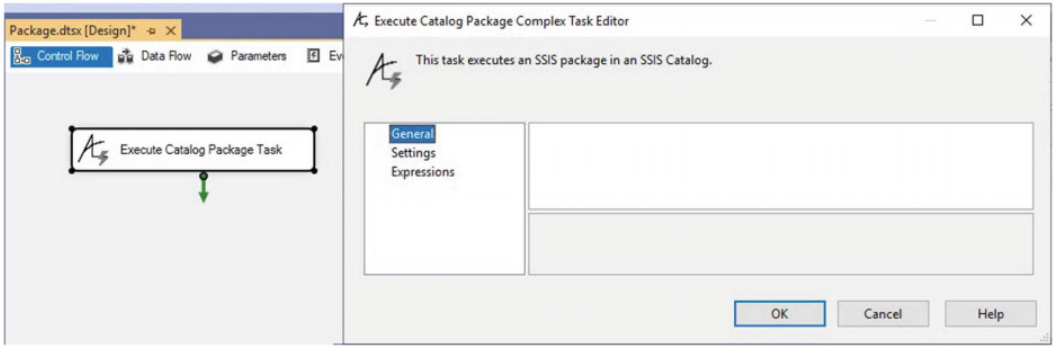
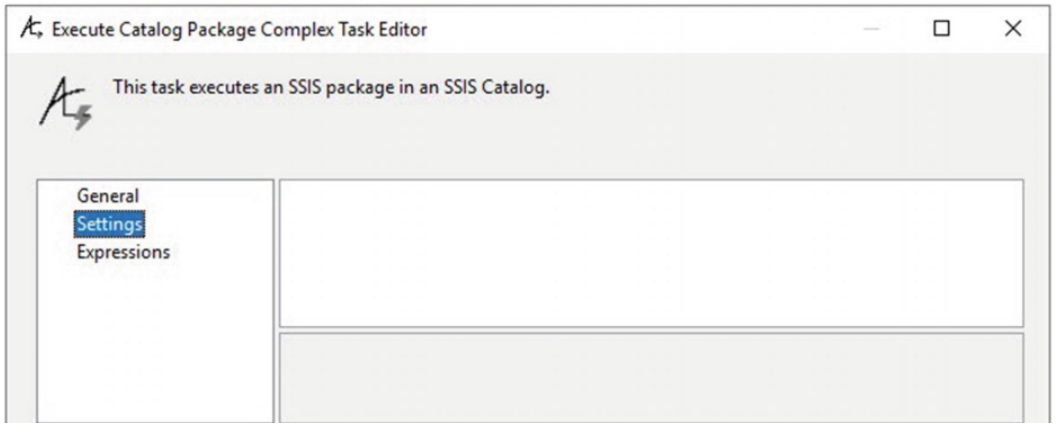


Figure 12-16 The Execute Catalog Package Task Editor may be opened without error

The tests pass! We now have a minimally coded Execute Catalog Package Task with a shiny, new editor.

The GeneralView and SettingsView are empty, as seen in Figure 12-16 and Figure 12-17:



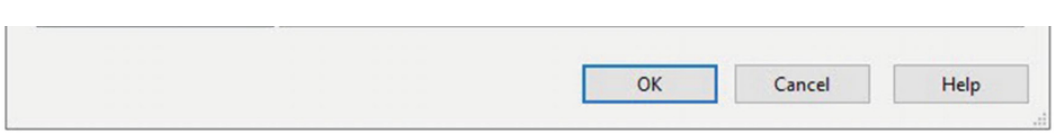


Figure 12-17 SettingsView is empty

Click the Expressions page, click inside the Expressions property value textbox, and then click the ellipsis to open the Property Expressions Editor, as shown in Figure 12-18:

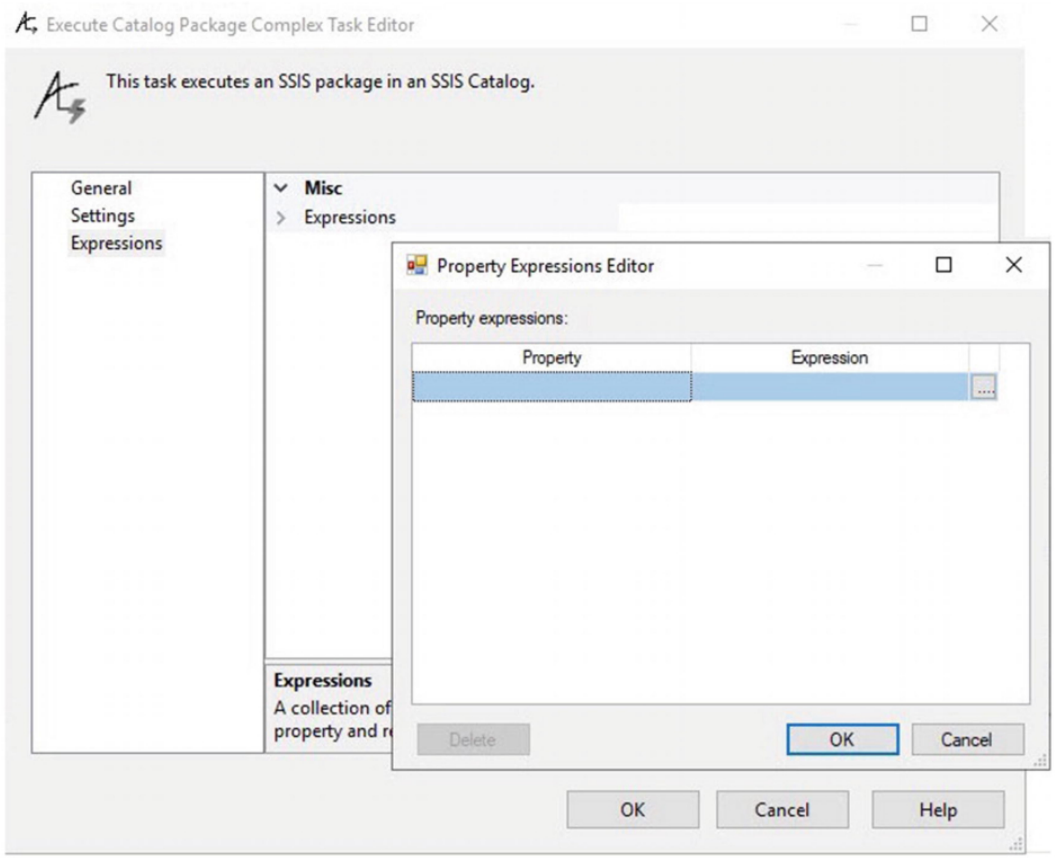


Figure 12-18 Expressions

Recall 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”

- Enable the use of SSIS Expressions
- Add design-time validation of task settings
- Surface more SSIS Catalog execution properties in the editor

In this section, we begin to see the second bullet realized – enabling the use is SSIS expressions – as Figure 12-18 shows.

We added no code to create an “ExpressionsView,” so where did this editor page come from? The ExpressionsView is inherited from the `Microsoft.DataTransformationServices.Controls` assembly. You may not believe it at this time, but the ability to manipulate SSIS custom task properties using SSIS expressions is worth all the work required to add an “SSIS-y” editor.

Conclusion

In this chapter, we focused on operations required to couple the new complex editor (ExecuteCatalogPackageTaskComplexUI) with the ExecuteCatalogPackageTask. Once coupled, we were able to test the minimally coded ExecuteCatalogPackageTask in a test SSIS package.

Now would be an excellent time to check in your code.

In the next chapter, we add properties.