Robert L. Bogue
MS MVP, MCSE, MCSA:Security
317-844-5310
Rob.Bogue@ThorProjects.com

Wrangling SharePoint Workflows with Visual Studio

Before I can really describe what SharePoint and Workflow are together it's important to understand what is meant when I talk about workflow. Its most basic components are a stimulus—an event—and a response. The response can continue until the workflow is completed. This is what people think of when they visualize a flowchart. Alternatively, a workflow may wait for the next stimulus to cause a transition from one state to another.

An example of a flowchart-like, or sequential, workflow, might be a set of approvals, beginning with one manager and moving up the chain of command. Regardless of what is being approved, the workflow transitions through a set of actions waiting intermittently for new input. The flow is largely linear. Workflows don't go back from the "VP approval" state to the "manager approval" state. They simply progress through the process from one approval to the next.

Alternatively, you might have a process such as a contract negotiation which alternates between states where both parties may: counter offer, quit negotiating, or accept the last offer. The number of rounds in this sort of a workflow may be many or few. Ultimately, however, the stimulus (event) indicates the state that the item should next enter. These types of workflows are called queue-based or state machine workflows.

Both of these types of workflows—each of these views of how workflows work—is something that SharePoint supports through the use of the Windows Workflow Foundation. And Visual Studio is the most powerful environment for creating workflows in SharePoint; it's a fitting place to start.

Workflow Foundation

SharePoint's workflow capabilities are based on the Workflow Foundation, which is a part of .NET 3.0. Microsoft has a whole site dedicated to the Workflow Foundation, what it does, and how it works at. The Workflow Foundation is designed to make it easier for developers to integrate workflow into their applications in a consistent way.

Workflow Foundation (WF) supports both sequential (flowchart-like) and state machine (queue-based) workflows. This provides a great deal of flexibility in terms of how you define the workflow you're trying to automate.

Also included with Workflow Foundation is a set of activities. Within workflow these are the components—or shapes—in the flow chart or in the state machine workflow. Each activity performs a different function including looping, branching, and waiting on external events. This means that SharePoint needs only to add the additional activities that are specific to SharePoint features.

Why Visual Studio?

One might ask why use Visual Studio when you could just as easily—or perhaps more easily—use SharePoint Designer? The short answer is reusability. SharePoint Designer workflows, in addition to being more limited, are always associated with just one list. Although you can template that list and get copies of the workflow, fundamentally the workflows are not reusable. Visual Studio workflows can be reused without limitation.

Another valid question is "What about InfoPath?" InfoPath can be a part of the workflow solution—for those with Microsoft Office Sharepoint Server (MOSS) Enterprise, which is out of reach for many developers. However, the techniques you'll learn here are applicable to workflows that include InfoPath as well.

Creating a Workflow

Enough of the theory and background, it's time to start developing a workflow.

In this example you're going to build a workflow that tracks the grading of homework assignments. The grading workflow is designed to activate whenever a document is uploaded to a specific document library. There is also a task list that informs the workflow when it has new work to do. The workflow can add to this list because it runs as a special system account. Instructors and assistants subscribe to alerts on the task list and are thus informed that a new assignment is ready to be graded.

When the instructor or assistant has reviewed the assignment they enter a numeric score as well as notes on the score into the task. This is copied back into the document in a set of fields that cannot be modified. (In a real life situation you might also integrate the workflow with a grade book program to add the grades automatically.) The grading example demonstrates a number of core workflow concepts:

- Content Types—You'll use content types to create a custom task type as well as to create fields that cannot be edited.
- Task Creation—Creating a task in a SharePoint workflow isn't as easy as dropping an activity on the design surface. Because this workflow relies upon tasks to take instructor input, you'll walk step-by-step through the creation of tasks.
- Property Modification—You'll see how to make changes to the workflow item while the workflow is running.

Content Types

Every piece of information in SharePoint has a content type. The content type defines the fields that are associated with the content. It is possible, as is the case in lists, to add additional fields beyond the base content type, however, the best way to ensure that a field is present is to define a new content type.

Content types can also inherit from other content types. For instance, the default workflow task, which is used to create tasks from a workflow, derives from the standard task content type. In this example, you're going to create content types derived from the workflow content type.

Content types comprise three basic components: field definitions, field references, and the feature that you deploy them in. Let's start by defining two fields in XML (you'll use these later in this article):

```
<Field ID="{0116A5FF-6FB7-43e1-B3E3-30A58B40349C}"
     Name="GradingScore" Group="Grading" DisplayName="Score"
     Type="Number" Sealed="FALSE" ReadOnly="FALSE" Hidden="FALSE"
     DisplaceOnUpgrade="TRUE"/>
<Field ID="{81792580-6F97-4960-84C2-7A8A926D1DCE}"
     Name="GradingNotes" Group="Grading" DisplayName="Notes"
     Type="Text" Sealed="FALSE" ReadOnly="FALSE" Hidden="FALSE"
     DisplaceOnUpgrade="TRUE"/>
```

Let's walk through each of the attributes from the field definitions above:

- ID—A unique GUID for the field. You can generate a GUID from Visual Studio by selecting Tools->Create GUID. The Create GUID application will launch. From there you can click the Copy button to copy the GUID. The GUIDs above were created in precisely this way.
- Name—This is the internal name of the field in SharePoint. Care must be taken when naming fields so as not to collide with an internal name already used by SharePoint or with the names of other fields on the system. Note that the fields above were prefixed with Grading for this reason. These names should not contain spaces or special punctuation.
- Group—This is the grouping that will be used to organize fields in the user interface. This name can be anything that makes sense for the application.
- DisplayName—This is the name that will be displayed to the user when they see the field. This field may contain spaces and the special punctuation to make the field easy for the user to understand.
- Type—This is the type of the field to be created. Here we're using the Number and Text values but Boolean, Choice, Computer, Currency, DateTime, URL, and other field types can be used as described at http://msdn2.microsoft.com/en-us/library/ms437580.aspx .
- Sealed—If True then no one will be able to change the field in list settings. By setting them to False, the fields are modifiable.

- ReadOnly—The field itself can be written to. If you don't set this to False (or omit it to accept the default) the field won't show up in the user interface.
- Hidden—You might hide a field if you want to use it internally but you don't want users to be able to see it.
- DisplaceOnUpgrade—This setting indicates that if the field definition already exists we want to overwrite it with the values in this XML fragment.

With the fields created, the next step is to create your content type and add references to the fields. In order to add the content types, you need to know how to derive from an existing content type. There's a base content type hierarchy in MSDN at http://msdn2.microsoft.com/en-us/library/ms452896.aspx . This shows the content IDs for existing types. Unlike other IDs in SharePoint, which are either an integer or a GUID, the content type Id is a special hexadecimal string that represents not only the content type itself but also its parent—sort of like a human's surname except in more detail.

There are two mechanisms for creating your content type. The first mechanism is appending a two-digit ID after the existing content type. The two-digit ID can be anything except 00. This is generally reserved for content types derived from your own content types because the potential for collision is so high (1 in 256 odds).

The other mechanism, which is recommended, is 00 followed by a GUID that has had all of its punctuation removed. This creates a longer content type ID so it's recommended only for the first level that you derive from a system type. Thereafter you should use the two-digit mechanism in order to minimize the overall length of the content type, which is capped at 512 bytes—in other words 1,024 hexadecimal characters. (You can learn more about content type IDs in the Content Type IDs entry in MSDN available at http://msdn2.microsoft.com/en-us/library/aa543822.aspx .)

The content type for the grades workflow task looks like this:

```
<ContentType ID="0x01080100B7336179CFFE43e59B86E241C767010E"
     Name="GradingTask" Group="Grading" Description="Grading Task"
     Version="0" Hidden="FALSE" >
     <FieldRefs>
     <FieldRef ID="{0116A5FF-6FB7-43e1-B3E3-30A58B40349C}"
          Name="GradingScore" DisplayName="Score" />
     <FieldRef ID="{81792580-6F97-4960-84C2-7A8A926D1DCE}"
          Name="GradingNotes" DisplayName="Notes"/>
     </FieldRefs>
</ContentType>
```

Here the "00+Guid" method of creating the content type was used because it's the first level from a system content type. The workflow task content type is 0x010801 thus that is how this content type begins. The XML fragment also contains <FieldRef> nodes in that correspond to the fields defined above—both the ID and the Name fields match exactly.

The next content type is the content type for the document which will have the same score and notes fields but this time they won't be editable. The document content type as shown below:

```
<ContentType ID="0x010100ADEC125AC3C74c1bA7E3A50731525809"
     Name="GradingDocument" Group="Grading" Description="Grading Document"
     Version="0" Hidden="FALSE" >
     <FieldRefs>
     <FieldRef ID="{0116A5FF-6FB7-43e1-B3E3-30A58B40349C}"
          Name="GradingScore" DisplayName="Score" ShowInDisplayForm="TRUE"
          ShowInFileDlg="FALSE" ShowInListSettings="TRUE"
          ShowInEditForm="FALSE"
          ShowInNewForm="FALSE" ReadOnlyClient="TRUE" />
     <FieldRef ID="{81792580-6F97-4960-84C2-7A8A926D1DCE}"
          Name="GradingNotes" DisplayName="Notes" ShowInDisplayForm="TRUE"
          ShowInFileDlg="FALSE" ShowInListSettings="TRUE"
          ShowInEditForm="FALSE"
          ShowInNewForm="FALSE" ReadOnlyClient="TRUE" />
     </FieldRefs>
```

```
    </ContentType>
```

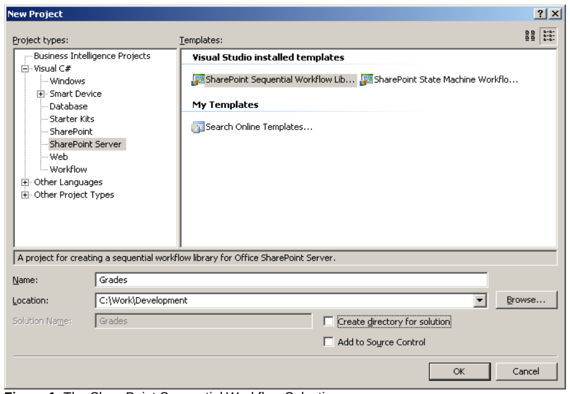
This document type is derived from a document (0x0101) with the same GUID method used above—but not the same GUID. The content type also includes the same fields as the task by creating <FieldRef> nodes, which match ID and Name attributes to the <Field> nodes created above. In this case, however, the fields should not be able to be written to. There is a ReadOnly attribute for the <FieldRef> but it causes the field to become hidden as well, so you can't use it. Instead, the attributes used here prevent the field from displaying on the new and edit forms. The new attributes are described below:

- ShowInDisplayForm—The display form is the form that is displayed when the user elects to view the item or view properties of a document. This value is set to True so that the field will be displayed.
- ShowInFileDlg—When an Office application saves the document does the field show? This is set to False since the user isn't supposed to manipulate the value directly.
- ShowInListSettings—This shows the field in the list settings. If False the field will not be visible when changing the list. This value is True so that the field will be displayed.
- ShowInEditForm—The edit form is the one that users get when they edit an item or edit properties on a document. Since the user shouldn't change the value the attribute is False.
- ShowInNewForm—The new form is the one that users get when they try to create a new item. Since the user shouldn't ever set this value the attribute is False.
- ReadOnlyClient—This attribute indicates that the client shouldn't be able to set the value of the field. This is True to prevent client side changes.

The content types are done. In order to deploy them a SharePoint feature is needed. Because the workflow will automatically create a feature, you'll use the workflow feature to deploy your content types at the same time.

Creating a Workflow Project in Visual Studio

Above, we mentioned that you would need to install a few components in order to develop workflows with SharePoint and Workflow Foundation. Once those pieces are installed, an option to create a SharePoint Sequential workflow will appear as shown in Figure 1. The Grades workflow will perform a few basic steps. It will create a task, wait for the task to change, change the document with some code, and delete the task.



A project for creating a sequential workflow library for Office SharePoint Server.

Name: Grades

Location: C:\Work\Development

Solution Name: Grades

Figure 1. The SharePoint Sequential Workflow Selection.

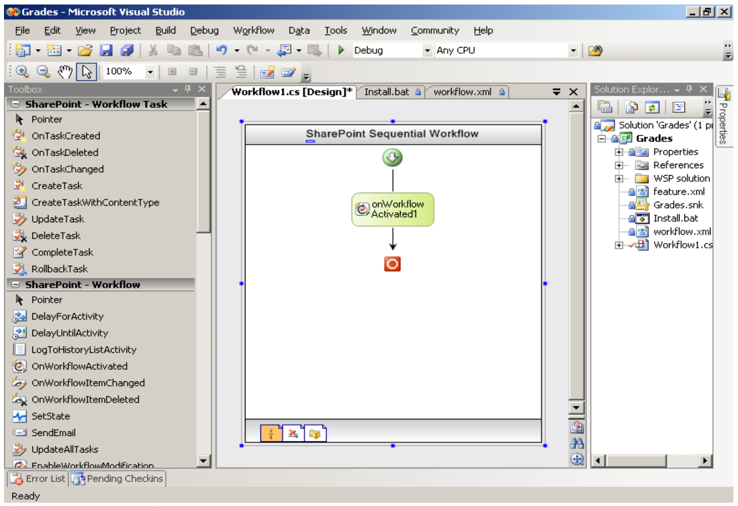


Figure 2. The blank workflow template.

To create the outline for the Grades workflow, follow these steps:

1. Create a sequential workflow by selecting the SharePoint (for WSS) or SharePoint Server (for MOSS) folder under Visual C#. Give the project a name of Grades, and select a folder to put the project in. Click the OK button.
2. Once the project has been created, double-click the Workflow1.cs file from the Solution Explorer to show the workflow that was automatically created. Figure 2 shows what the workflow looks like as soon as it's created. Visual Studio has automatically created the workflow and added the required onWorkflowActivated activity as the first activity.

3. From the toolbox drag a CreateTask activity, a OnTaskChanged activity, a Code activity, and a DeleteTask activity in sequence to the design surface as shown in Figure 3.

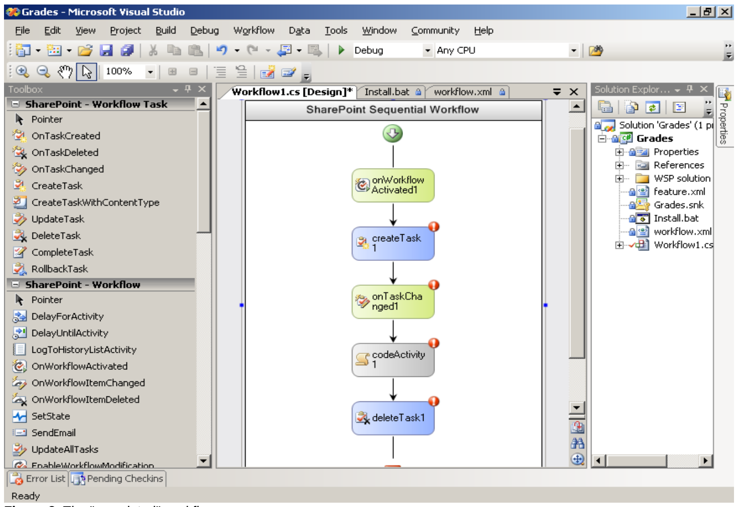


Figure 3. The "completed" workflow on screen.

You may notice that there are little red exclamation points next to each of the four activities that you added to the design surface. That is because they all failed validation. The design interface is warning you that, based on the current settings on the activities, they won't work. Next you're going to fix those errors and complete your workflow.

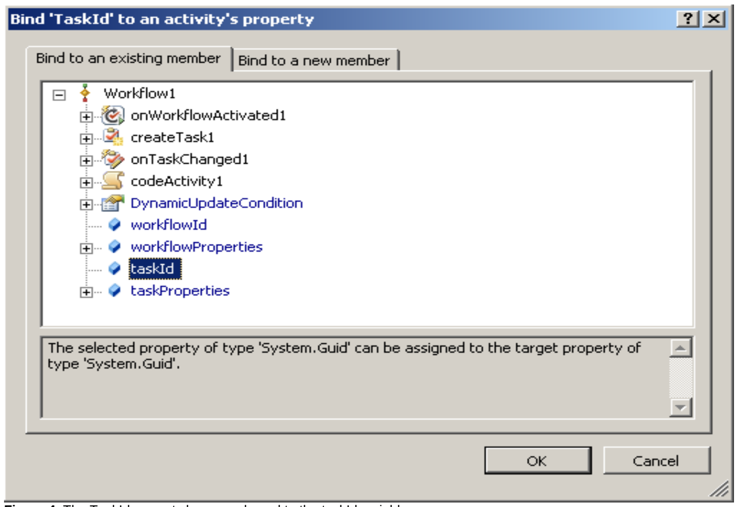
Completing the Create Task Activity

In order to get a CreateTask activity working, three things are needed: a correlation token, a task properties object, and a task GUID. The easiest of these to take care of is the correlation token, which can be created with the following process:

1. Click on the createTask1 activity, then right-click and select Properties.
2. In the Properties pane, select the CorrelationToken box and type in taskToken and press Enter.
3. Now click the plus sign to the left of CorrelationToken and in the OwnerActivityName line that appears select Workflow1. Visual Studio has created a correlation token for you. Note that each item that will receive events needs its own correlation token. Thus the task correlation token you created is different than the workflow correlation token that was created automatically when the project was created.
4. Defining the task properties and the GUID are a bit more challenging. For that you'll need to right-click the activity and select view code. You need to add two lines underneath the lines that define the workflowId and workflowProperties. The lines will define the taskId and taskProperties like this:

```
public Guid taskId = default(System.Guid);
public SPWorkflowTaskProperties taskProperties =
     new SPWorkflowTaskProperties();
```

5. Switch back to the design view, which appears in the tabs as Workflow1.cs [Design].
6. Click on the createTask1 activity, right-click and select Properties from the context menu.



The selected property of type 'System.Guid' can be assigned to the target property of type 'System.Guid'.

Figure 4. The TaskId property becomes bound to the taskId variable.

7. Select the taskId property and click the OK button. You've just bound the taskId Property to the taskId that you created.
8. Perform steps 7 and 8 again on the TaskProperties activity property and taskProperties workflow property.
9. Next is setting the taskId correctly. To do this code will be added to the MethodInvoking event. Double-click the createTask1 activity so that it creates the default event handler, which is MethodInvoking. Methods that end in 'ing' happen before the main work of the activity so you can use this method to set the taskId. You need to do this because you didn't give it a unique ID originally—just the default for a GUID (all zeros). Add the following line to the createTask1_MethodInvoking method that was created when you double clicked:

```
taskId = Guid.NewGuid();
```

10. In order to set the task title to something meaningful, add the following line immediately following the line added in step 10:

```
taskProperties.Title = "Task for " + workflowProperties.Item.Name;
```

11. Switch back to the design view. You're done with the createTask. The red exclamation point that was on it should be gone now. (Actually, it disappeared when you set the correlation token.)

Completing OnTaskChanged

For the onTaskChanged1 activity you have essentially the same properties to set, however, because they are already created the process is much easier. In this case the correlation token must be set as well as the TaskId property. In addition, taskProperties is bound to AfterProperties so that the taskItemId value will be set.

1. Select the onTaskChanged1 activity, right-click and select Properties from the context menu.
2. In the CorrelationToken field click the drop-down list box arrow and select taskToken.
3. Click on the TaskId property field and then the ellipsis.
4. Select taskId from the binding dialog and click the OK button.
5. Select the AfterProperties field, and then click the ellipsis.
6. Select taskProperties from the binding dialog and click the OK button.

The onTaskChanged activity is done. The red exclamation point should be gone for this activity too.

Completing the Code Activity

The final activity is the code activity where you're going to copy the fields from the task back to the document.

Double-click the codeActivity1 activity to create a handler for ExecuteCode. In this method you're adding these five lines:

```
SPListItem item = workflowProperties.Item;
SPListItem task =
workflowProperties.TaslKist.GetItemById(taskProperties.TaskItemId);
item["GradingScore"] = task["GradingScore"];
item["GradingNotes"] = task["GradingNotes"];
item.SystemUpdate(false);
```

These five lines get the item for which the workflow was started, the task that was created, and then copies two fields from the task into the item. Finally, it updates the fields without changing the modified information for the item.

The workflow is done. It should now build successfully. It's time to move on to deploying the workflow.

Completing the Delete Task Activity

Completing the Delete Task activity is the same as completing the OnTaskChanged activity. You must set the CorrelationToken and the TaskId properties of the deleteTask1 activity so that the workflow will compile and so that the task will be deleted. You can refer back to Completing OnTaskChanged for the steps to bind these two properties.

Modifying the Feature to Deploy the Workflow

When Visual Studio created the project it automatically added three files that will help in the deployment process. They are:

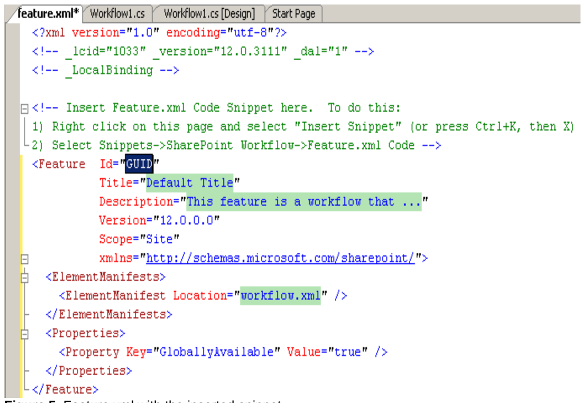
- Feature.xml—SharePoint needs this file to define the feature.
- Workflow.xml—This element manifest file defines the workflow itself.
- Install.bat—This batch file can be used to install the feature.

The following sections walk through each of these files and make the updates necessary to install the feature containing the Grades workflow.

Feature.xml

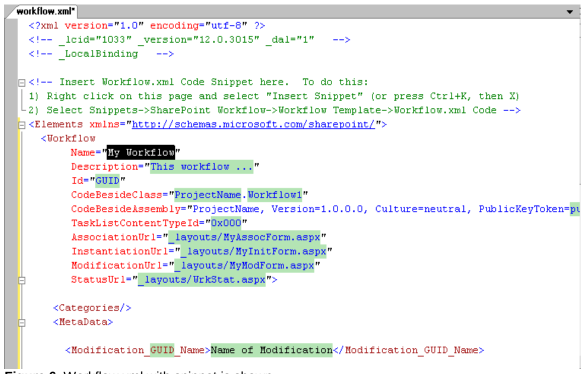
Upon opening feature.xml, you'll notice that there isn't much there besides a note telling you to use the snippets functionality with Visual Studio to add the nodes necessary in the feature.xml file. To create the feature.xml for the Grades Workflow follow these steps:

1. Add the feature from the Windows SharePoint Services SDK by right clicking, selecting Insert Snippet, Windows SharePoint Services Workflow, and finally Feature.xml Code. Figure 5 shows what you'll see when you add the snippet. Once the snippet is added the green highlighted sections must be changed. Specifically the Id, Title, and Description must be changed. The default value for the <ElementManifest> Location attribute is correct. First up is changing the Id.



```
<?xml version="1.0" encoding="utf-8"?>
<!-- _lcid="1033" _version="12.0.3111" _dal="1" -->
<!-- _LocalBinding -->

<!-- Insert Feature.xml Code Snippet here.  To do this:
1) Right click on this page and select "Insert Snippet" (or press Ctrl+K, then X)
2) Select Snippets->SharePoint Workflow->Feature.xml Code -->
<Feature  Id="GUID"
          Title="Default Title"
          Description="This feature is a workflow that ..."
          Version="12.0.0.0"
          Scope="Site"
          xmlns="http://schemas.microsoft.com/sharepoint/">
  <ElementManifests>
    <ElementManifest Location="workflow.xml" />
  </ElementManifests>
  <Properties>
    <Property Key="GloballyAvailable" Value="true" />
  </Properties>
</Feature>
```

Figure 5. Feature.xml with the inserted snippet.



```
<?xml version="1.0" encoding="utf-8" ?>
<!-- _lcid="1033" _version="12.0.3015" _dal="1"   -->
<!-- _LocalBinding   -->

<!-- Insert Workflow.xml Code Snippet here.  To do this:
1) Right click on this page and select "Insert Snippet" (or press Ctrl+K, then X)
2) Select Snippets->SharePoint Workflow->Workflow Template->Workflow.xml Code -->
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Workflow
       Name="My Workflow"
       Description="This workflow ..."
       Id="GUID"
       CodeBesideClass="ProjectName.Workflow1"
       CodeBesideAssembly="ProjectName, Version=1.0.0.0, Culture=neutral, PublicKeyToken=pu
       TaskListContentTypeId="0x000"
       AssociationUrl="_layouts/MyAssocForm.aspx"
       InstantiationUrl="_layouts/MyInitForm.aspx"
       ModificationUrl="_layouts/MyModForm.aspx"
       StatusUrl="_layouts/WrkStat.aspx">

    <Categories/>
    <MetaData>

      <Modification_GUID_Name>Name of Modification</Modification_GUID_Name>
```

Figure 6. Workflow.xml with snippet is shown.

2. Select Tools->Create GUID from the Visual Studio menu.
3. Verify that the Create GUID application has the Radio button for *4. Registry Format (ie. {xxxxxxxx-xxxx … xxxx })* selected.
4. Click the Copy button in the Create GUID application.
5. Switch back to Visual Studio and the Feature.xml file.
6. Select the text in the quotes behind the Id attribute and paste the GUID in with Ctrl-V.
7. Strip the leading and closing braces from the GUID that was pasted in.
8. Enter a descriptive name for the feature and for the description. For the Grades workflow the feature is going to be called "Grades" and the description is "Enable assignment grading."

Because workflow.xml is the correct name of the element manifest that you want to create, you're done with Feature.xml. Save it and you can move on to workflow.xml.

Workflow.xml

When workflow.xml is opened, you'll find the situation very similar to the feature.xml. Workflow.xml is a mostly empty file with a few instructions on how to add the content to the file. We'll follow a similar procedure as with the feature.xml. To customize workflow.xml for the Grades workflow follow these steps:

1. Right-click and select Insert Snippet, Windows SharePoint Services Workflow, and finally Workflow.xml Code. This will result in a workflow.xml that looks like the one shown in Figure 6.
2. Enter the name and description for the workflow. For the Grades workflow, use "Grades" and "Enable assignment grading" respectively for the name and description. It isn't required that you enter either the same or unique names here versus the values that are used in the feature.xml file.
3. Use the Create GUID tool to create a new GUID. (See Steps 2-4 of the feature.xml procedure immediately above if you don't remember how.)
4. Switch back to Visual Studio and the workflow.xml file.
5. Paste the GUID into the ID attribute's value and delete the extra braces.
6. The next step is the code-behind class; in this case the project name is 'Grades' so the namespace is 'Grades.' For the CodeBesideClass enter "Grades.Workflow1." Workflow1 is the name of the workflow that was created by the Visual Studio SharePoint Sequential Workflow project.
7. Next we need to get the correct assembly name; the first part of the CodeBesideAssembly is "ProjectName." This is the name of the DLL for the project. In this case the assembly name is 'Grades,' so replace ProjectName with 'Grades.'
8. The last part of the CodeBesideAssembly is the PublicKeyToken. For this the assembly must be strong named. This is a requirement for all workflow DLLs since they must be installed into the GAC. In Visual Studio, right-click on the project name (Grades) and select properties. On the Properties page that appears, select the Signing tab, which is the last tab on the left. The result will be similar to the page shown in Figure 7.

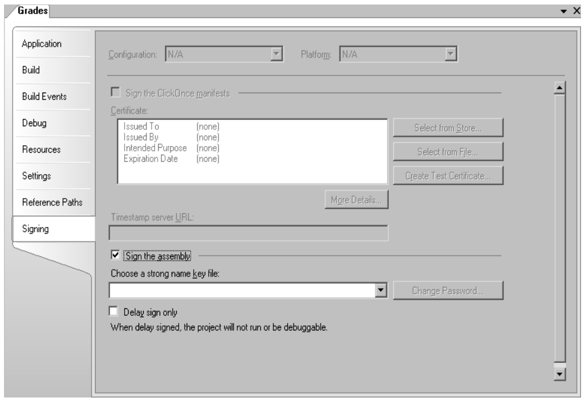
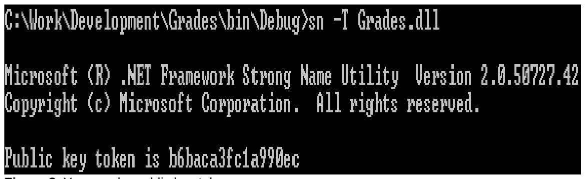


Figure 7. The signing tab of the project properties page is shown.

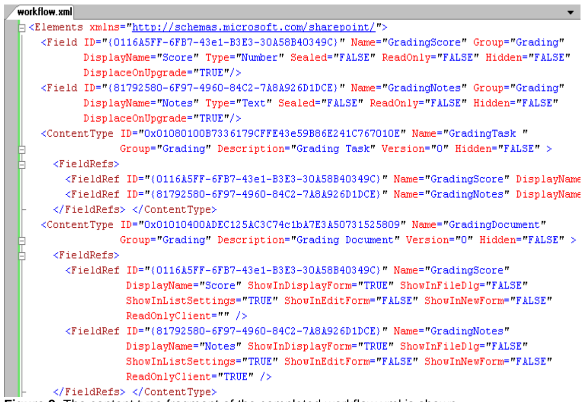


```
C:\Work\Development\Grades\bin\Debug>sn -T Grades.dll

Microsoft (R) .NET Framework Strong Name Utility  Version 2.0.50727.42
Copyright (c) Microsoft Corporation.  All rights reserved.

Public key token is b6baca3fc1a990ec
```

Figure 8. You need a public key token.

9. From the drop-down box select 'Choose a strong name key file' and select New. A Create Strong Name Key dialog will appear.
10. Enter a key file name of Grades.
11. Uncheck the Protect my key file with a password checkbox and click the OK button.

12. Build the solution to get the public key token now. (Ctrl-Shift-B will build the solution.)
13. Next open a Visual Studio 2005 command prompt. This can be done by selecting it from the Start menu. The entry is located under Visual Studio 2005->Visual Studio Tools and is named Visual Studio 2005 Command Prompt.
14. Navigate to the directory where the project DLL is created. In our example I created the solution in *C:\Work\Development\grades*. The debug DLL is created under bin\debug by default so the command to change to the correct directory is 'CD \work\Development\grades\bin\debug'.
15. Type the command 'SN –T Grades.DLL.' SN responds with the public key token for the file as shown in Figure 8.
16. Right-click in the window and select Mark.
17. Click and drag across the public key token. When you're done let up the mouse and press Enter.
18. Switch back to Visual Studio and to the workflow.xml tab.
19. Select the green highlighted publicKeyToken text and then press Ctrl-V to paste the public key token.
20. One final edit for the CodeBesideAssembly attribute is to change the version number from 1.0.0.0 to 3.0.0.0.
21. Next set the TaskListContentTypeId. This is the Id shown above in the content type section. In this case the value was the 0x01080100B7336179CFFE43e59B86E241C767010E value from the content type tag, which is defined the content type for the task.
22. Delete the AssociationUrl, InstantiationUrl, ModificationUrl, and StatusUrl attributes because the Grades project doesn't implement any of these features.
23. Delete the whole MetaData node and the modification element under it because the Grades workflow doesn't implement a workflow modification either.
24. If you were just doing the workflow in this file you would be done. However, you're going to simplify things a bit and add your fields and content types to the top of this file. To do this, copy the field and content type nodes above the <Workflow> node you've been modifying. Figure 9 shows what the completed file should look like. Note that the formatting was changed and extraneous comments removed to get everything to fit on the screen.



```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field ID="{0116A5FF-6FB7-43e1-B3E3-30A58B40349C}" Name="GradingScore" Group="Grading"
         DisplayName="Score" Type="Number" Sealed="FALSE" ReadOnly="FALSE" Hidden="FALSE"
         DisplaceOnUpgrade="TRUE"/>
  <Field ID="{81792580-6F97-4960-84C2-7A8A926D1DCE}" Name="GradingNotes" Group="Grading"
         DisplayName="Notes" Type="Text" Sealed="FALSE" ReadOnly="FALSE" Hidden="FALSE"
         DisplaceOnUpgrade="TRUE"/>
  <ContentType ID="0x01080100B7336179CFFE43e59B86E241C767010E" Name="GradingTask "
               Group="Grading" Description="Grading Task" Version="0" Hidden="FALSE" >
    <FieldRefs>
      <FieldRef ID="{0116A5FF-6FB7-43e1-B3E3-30A58B40349C}" Name="GradingScore" DisplayName
      <FieldRef ID="{81792580-6F97-4960-84C2-7A8A926D1DCE}" Name="GradingNotes" DisplayName
    </FieldRefs> </ContentType>
  <ContentType ID="0x01010400ADEC125AC3C74c1bA7E3A50731525809" Name="GradingDocument"
               Group="Grading" Description="Grading Document" Version="0" Hidden="FALSE" >
    <FieldRefs>
      <FieldRef ID="{0116A5FF-6FB7-43e1-B3E3-30A58B40349C}" Name="GradingScore"
                DisplayName="Score" ShowInDisplayForm="TRUE" ShowInFileDlg="FALSE"
                ShowInListSettings="TRUE" ShowInEditForm="FALSE" ShowInNewForm="FALSE"
                ReadOnlyClient="" />
      <FieldRef ID="{81792580-6F97-4960-84C2-7A8A926D1DCE}" Name="GradingNotes"
                DisplayName="Notes" ShowInDisplayForm="TRUE" ShowInFileDlg="FALSE"
                ShowInListSettings="TRUE" ShowInEditForm="FALSE" ShowInNewForm="FALSE"
                ReadOnlyClient="TRUE" />
    </FieldRefs> </ContentType>
```

Figure 9. The content type fragment of the completed workflow.xml is shown.

With workflow.xml done, there's only one more file to finish, the install.bat file.

Install.bat

The Install.bat file opens when you double-click it in the Solution Explorer. It's a nice change from the feature.xml and workflow.xml files. The changes you need to make to the install.bat file are basically two global search and replaces. To customize the install.bat file follow these steps:

1. The first search and replace is for MyFeature and replace it with what you want to name your feature—in this case 'Grades.' To perform the search and replace start by pressing Ctrl-H.
2. Enter MyFeature in the "Find what" box and Grades in the "Replace with" box. Make sure that the Look in drop-down list is set to "Current Document." Press the Replace All button.
3. Click OK in the message box indicating 14 replacements were made.
4. The second search and replace is for http://localhost with the name of your server. In my case the server name is w2k3server so I'm replacing http://localhost with http://w2k3server.
5. Click the OK button in the message box indicating three replacements were made.

Once you've made all of the replacements select Save All in Visual Studio (File->Save All) and open a command prompt. (The Visual Studio 2005 command prompt used earlier will work fine.) Navigate to the project directory (*C:\Work\Development\Grades*) and run install by typing 'install' and pressing enter at a command prompt.

Associating the Workflow to a Document Library

Now that the workflow and associated content types are created it's time to associate those content types and the workflow to a set of lists in SharePoint. For these instructions you'll need a blank team site. I've created one called Grades under the root of my server. You may use the root site of your installation or create a site underneath the root site to associate the content types and workflow.

To associate the content type for the grading document, which has the score and notes field, to the document library and to remove the default content type, follow these steps:

1. Open up the document library and then from the Settings menu select the Document Library Settings entry.
2. The first step is to activate the content type. Do that by clicking Advanced Settings in the General Settings section. The top item on the Document Library Advanced Settings page is Allow management of content types. Select the Yes radio button and scroll down to the bottom of the page and click the OK button.
3. Back on the Customize Shared Documents page in the Content Types section, click the 'Add from existing site content types.'
4. On the Add Content Types page (which is in the drop-down list for 'Select site content types from'), select Grading. Click the Add button to move the GradingDocument to the box of added content types on the right. Click the OK button to finish.
5. The next step is to remove the Document type so that all documents that are created in the document library will be the GradingDocument type. Click on the Document link under the Content Types heading.
6. From the List Content Type page click the 'Delete this content type' link.
7. Click the OK button on the popup dialog that asks if you're sure.

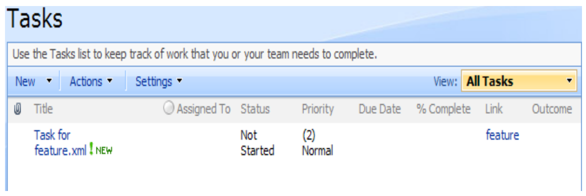
With the content type in place it's time to associate the workflow. Follow these steps:

1. From the Customize Shared Documents page in the Permissions and Management column click the Workflow Settings link.
2. On the Add a Workflow page in the 'Select a workflow template:' listbox, scroll down and select the Grades workflow template.
3. Enter a name for this workflow association: "Grades" works fine.
4. Scroll down to the Start Options section and check the box for 'Start this workflow when a new item is created.' Click the OK button. The Grades workflow is added to the document library.

All of the plumbing is done. All that is left is to test the workflow and ensure that it works.

Testing the Workflow

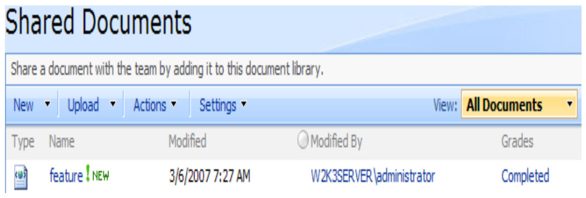
Now that the workflow is associated with a document library and set to activate on new items all that needs to happen is for a document to be uploaded. When this happens the Grades workflow kicks off and adds a new task to the workflow. Figure 10 shows the task created when the feature.xml file from the Grades project was uploaded to the Shared Documents library.


Tasks

Use the Tasks list to keep track of work that you or your team needs to complete.

Title	Assigned To	Status	Priority	Due Date	% Complete	Link	Outcome
Task for feature.xml ! NEW		Not Started	(2) Normal			feature	

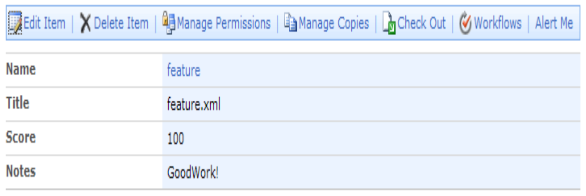
Figure 10. A new task is created by the Grades Workflow when a document is uploaded.


Shared Documents

Share a document with the team by adding it to this document library.

Type	Name	Modified	Modified By	Grades
	feature ! NEW	3/6/2007 7:27 AM	W2K3SERVER\administrator	Completed

Figure 11. The Grades workflow is completed.



Name	feature
Title	feature.xml
Score	100
Notes	GoodWork!

Figure 12. The results of the Grades workflow is shown

If you update the task to include a score and notes you'll notice that the task doesn't always go away immediately. Sometimes when you first refresh the task list the task will appear and sometimes it won't. Because the workflow and the refresh of the task list page occur at the same time it's possible you'll see or not see the task depending upon which thread executed quicker. Refreshing the task list page will definitely make the task disappear.

Switching back to the Shared Documents list you'll see that there is a column called Grades—which matches the association name you provided when you associated the workflow with the list—and that it has a value of Completed. Figure 11 shows the completed Grades workflow.

The properties of the document were in fact updated based on what was entered in the task. Figure 12 show that the student got a score of 100 and a note of "Good Work!"

SharePoint Workflows don't have to be scary. Yes, they are powerful. Yes, they are flexible. Yes, they require a lot of knowledge. However, using SharePoint Workflows can quickly create solutions to difficult problems.

About the Author

Robert Bogue, MCSE (NT4/W2K), MCSA:Security, A+, Network+, Server+, I-Net+, IT Project+, E-Biz+, CDIA+ is the president of Thor Projects LLC. He has contributed to more than 100 book projects and numerous other publishing projects. He was recently honored to become a Microsoft MVP for Microsoft Office SharePoint Server. Before that Robert was a Microsoft Commerce Server MVP and before that Microsoft Windows Servers-Networking MVP. Robert blogs at http://www.thorprojects.com/blog. You can reach Robert at Rob.Bogue@thorprojects.com.