



# Building Human-Oriented Workflows Using Windows Workflow Foundation

**E**very day, information workers follow work processes to get their jobs done. Some of those work processes will be quite formal, others very informal. Some of those processes will involve one or more people within the same organization; still others will involve multiple people working for different companies. These work processes have one thing in common: they are all examples of human-oriented workflows. A subset of this wide range of workflows falls in the solution domain of SharePoint 2007. SharePoint 2007 allows information workers and developers to create and use document-centric workflows.

## Basics of Windows Workflow Foundation

SharePoint 2007 incorporates an infrastructure that supports the use of existing document-centric workflows as well as the creation of new document-centric workflows by either developers or information workers.

Windows Workflow Foundation (WF) forms the basis of the SharePoint 2007 workflow functionality. Windows Workflow Foundation is a part of the Microsoft .NET Framework 3.0.

---

**Note** The Microsoft .NET Framework 3.0 was formerly known as WinFX. It adds four new technologies to the .NET Framework 2.0: Windows Workflow Foundation, Windows Communication Foundation (WCF, formerly known as Indigo), Windows Presentation Foundation (WPF, formerly known as Avalon), and Windows CardSpace (WCF, formerly known as InfoCard).

---

Windows Workflow Foundation is not a stand-alone application; instead, it has a run-time workflow engine that can execute short-term as well as persistable workflows. The Windows Workflow Foundation run time provides additional services that are important for executing workflows such as tracking, notifications, and transactions.

Any application or service can leverage WF workflow services by hosting WF in its process. SharePoint 2007 acts as a host for WF and adds a couple of custom implementations for workflow services as well.

SharePoint 2007 offers two different tools for working with and creating workflows: SharePoint Designer 2007 and Visual Studio 2005 Designer. Both will be discussed in this chapter.

SharePoint Designer 2007 is a great tool for creating ad hoc workflows. It places the tools to create workflows well within the grasp of normal information workers. Visual Studio 2005 Designer, on the other hand, allows developers to create advanced, enterprise-level workflows. Table 4-1 provides a comparison of SharePoint Designer 2007 and Visual Studio 2005 Designer.

**Table 4-1.** *Comparison Between SharePoint Designer 2007 and Visual Studio 2005 Designer*

Feature	SharePoint Designer 2007	Visual Studio Designer 2005
Requires you to write code	No	Yes
Supports sequential workflows	Yes	Yes
Supports state machine workflows	No	Yes
Supports the use of custom forms within the workflow	Yes	Yes
Workflows can be started manually	Yes	Yes
Workflows can be started by an event	Yes	Yes
Workflows can associated with a list	Yes. Workflows created using SharePoint Designer 2007 can only be bound to a single list. You cannot alter this association after deployment.	Yes
Workflows can be associated with a content type	No	Yes
Workflows can be associated with a site	No	Yes
Supports the use of ASP.NET forms	Yes	Yes
Supports the use of other form technologies, such as InfoPath	No	Yes
Workflow deployment happens automatically	Yes	No, but you can use a Web Part Solution Package (.wsp) to deploy a workflow.
Step-by-step debugging is available	No	Yes

---

**Note** Actually, you can associate a workflow created in SharePoint Designer with more than one list. The auto-generated code contains hard-coded GUIDs representing a list. If you change those GUIDs, it is possible to associate a workflow with another list.

---

In the following sections, we will delve deeper into the concepts related to Windows Workflow Foundation.

## Activities

We will begin by looking at the basic structure of a workflow. A workflow represents a business process that consists of a set of transactions which form a model to describe a real-world problem. Business transactions form the elementary building blocks of a business process. A business transaction involves two parties:

- The initiator is responsible for starting a business transaction by issuing some sort of request.
- The executor is responsible for performing some kind of action based on a request.

Conceptually, the life cycle of a business transaction consists of three phases:

1. *Order phase*: This phase starts when the initiator makes a request for some kind of action to be performed. This phase ends with a promise by the executor to perform that action.
2. *Execution phase*: In this phase, the executor performs the requested action. The action succeeds or fails. Either way, a result is reached, thus ending this phase of the business transaction life cycle.
3. *Result phase*: In this phase, the executor communicates which result was reached, which is either accepted or rejected by the initiator.

In Windows Workflow Foundation, the abstract concept of business transactions (a concept that has meaning in the external world) is translated to activities (a concept that has meaning in the software world). Activities are the building blocks of a workflow; every single step within a workflow requires an activity. Windows Workflow Foundation utilizes three different kinds of activities:

- *Simple activity*: A simple activity represents a simple action in its most basic form. An example of such a simple activity is the `DelayActivity`.
- *Composite activity*: A composite activity is an activity that aggregates one or more child activities within a single activity. An example of such a composite activity is the `IfElse` activity.
- *Rule activity*: A rule activity (also called a *data-driven activity*) drives the flow of a workflow. An example of this type of activity is the `EventDriven` activity.

Out of the box, the Windows Workflow Foundation framework contains a set of activities that provide functionality for the creation of workflows that contain control flow, conditions, event handling, and state management, and are able to communicate with other applications and services. Table 4-2 shows the activities that are included out of the box in Windows Workflow Foundation and that can be used in a SharePoint workflow.

**Table 4-2.** *Out of the Box Activities That Can Be Used in a SharePoint Workflow*

Name	Description
CodeActivity	This activity lets you add custom logic to a workflow in the form of code written using VB.NET or C#.
ConditionedActivityGroup	This activity lets you aggregate one or more child activities that are executed conditionally. The child activities can be executed based on some kind of condition that applies to the <code>ConditionedActivityGroup</code> activity itself, or based on a condition that only applies to the child.
DelayActivity	This activity lets you incorporate delays into workflows that are based on time-out intervals.
EventHandlingScopeActivity	This activity aggregates child activities and decides whether event handling is supported when those child activities are executed.
HandleExternalEventActivity	This activity is able to communicate with a local service by handling an event that is raised the service and is often used in conjunction with the <code>CallExternalMethod</code> activity.
IfElseActivity	This activity supports the ability to add decisioning mechanisms to a workflow. Such activities test for a condition and perform activities that are a part of the first branch of the <code>IfElseActivity</code> that matches the condition.
InvokeWebServiceActivity	This activity allows you to add web service communication within a workflow.

*Continued*

**Table 4-2.** *Continued*

Name	Description
ParallelActivity	This activity can be used to add parallel processing power (multithreading) to your workflows. This activity allows you to add two or more child Sequence activities that are executed concurrently.
ReplicatorActivity	This activity lets you create single child activities.
SequenceActivity	This activity lets you link multiple activities together that are to be executed sequentially.
SetStateActivity	This activity lets you specify a transition to a new state if you are creating a workflow that contains multiple states.
StateActivity	This activity lets you represent each separate state in a state machine type of workflow.
StateFinalizationActivity	This activity aggregates child activities that are executed whenever a StateActivity activity is finished executing.
StateInitializationActivity	This activity aggregates child activities that are executed upon entering a StateActivity activity.
SuspendActivity	This activity lets you temporarily suspend the operation of a workflow. You might want to do this if there is some kind of event or error condition that requires special attention.
TerminateActivity	This activity allows you to end workflow execution immediately. You might want to do this if an irrecoverable error condition occurs.
ThrowActivity	This activity is part of a larger exception-management framework that allows you to implement an error-handling mechanism in your workflows. This activity allows you to throw exceptions to signal unexpected conditions in your workflow.
WhileActivity	This activity adds conditional logic that enable workflows to loop until a given condition is met.

---

**Note** Please note that the toolbox contains other activities that are available in normal WF workflows. They are not discussed in this section because they cannot be used in workflows in a SharePoint environment.

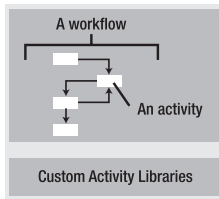
---

If you design a workflow, you can choose to make use of one of the activities belonging to this default set or you can create your own activities. Developers are free to create their own custom activity libraries and use them in other workflows. You can also use code activities to add custom code to a workflow. Code activities allow you to add basic behavior to a workflow; if you want complex behavior, such as the ability to create a composite activity that is able to contain other activities, you should create your own custom activity.

Activities can be sequential, in which case the order of workflow actions is specified at design time. Alternatively, activities can be event-driven. In such scenarios, the order of workflow actions is determined at run time in response to external events. Each activity contains the following logic/data:

- Metadata responsible for describing design time properties of the activity.
- Instance data describing the activity run-time state.
- Activity behavior in the form of programmed execution logic.
- Validation logic that can be used to validate activity metadata. This is optional.

Figure 4-1 shows a basic workflow that consists of several activities and uses custom activity libraries.

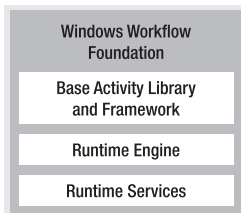


**Figure 4-1.** *Workflow with activities*

## Components

Windows Workflow Foundation provides services as well as a framework and consists of the following components (see Figure 4-2):

- *Base activity library:* This library contains all activities that are available out of the box. Activities are essential building blocks for creating custom workflows and are discussed in the previous section, “Activities.”
- *Runtime engine:* The workflow runtime engine is able to interpret and execute workflows created using Windows Workflow Foundation. The runtime engine is also responsible for keeping track of workflow state.
- *Runtime services:* The WF workflow runtime engine can be hosted by different host applications, including SharePoint. This is a very flexible system, which is made possible via the workflow runtime services. The workflow runtime services are also responsible for handling any communication between the host application and the workflow.



**Figure 4-2.** *Windows Workflow Foundation*

Windows Workflow Foundation ships with a couple important tools that facilitate workflow creation: a visual designer and a debugger. Both of them are integrated into Visual Studio 2005 and are discussed later in this chapter, in the sections “Visual Studio 2005 Designer” and “Debugging a Workflow.”

The workflow runtime engine is hosted in-process within a host application (never as a stand-alone process) and is responsible for creating and maintaining running workflow instances. For each application domain (in .NET, the fundamental process that executes code), there can be only one workflow runtime engine. A workflow runtime engine is able to run multiple workflow instances concurrently. The runtime engine needs to be hosted by a host process, such as a console application, Windows form-based application, ASP.NET web site, or a web service. Because a workflow is hosted in-process, it can easily and efficiently communicate with its host application.

The workflow runtime engine utilizes many WF services when a workflow instance runs. The following services are included in the runtime engine:

- Execution
- Tracking
- State management
- Scheduler
- Rules

Besides the runtime engine itself, WF contains multiple runtime services. Runtime services are pluggable, so applications can provide these services in unique ways within their execution environment. Table 4-3 shows an overview of the runtime services that are provided out of the box with Windows Workflow Foundation.

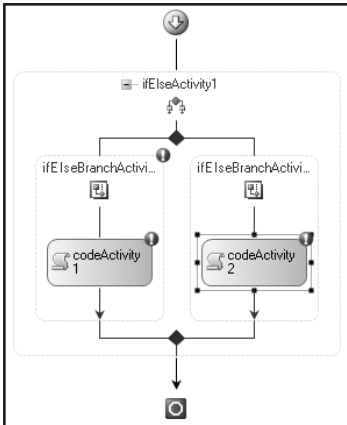
**Table 4-3.** *Runtime Services*

Name	Description
Persistence	The persistence service is responsible for saving and loading state data to and from a persistent data store, such as an SQL Server database. Some workflows, called <i>persistent workflows</i> , need to be able to survive system reboots or run for a long period of time. Building such workflows is easier with the Windows Workflow Foundation runtime services persistence service.
Communication	The communication service is responsible for handling the communication between a workflow and other applications, services, or workflows.
Timer	The timer service handles activities within workflows that are time-related. <code>DelayActivity</code> , discussed previously in the section “Activities,” is an excellent example of such an activity.
Tracking	The tracking service facilitates tracking and monitoring the execution flow of workflows. The tracking service also enables you to persist this information to a data store, such as a log file or an SQL Server database.
Transaction	The transaction service facilitates the incorporation of transactions within custom workflows.
Threading	The threading service facilitates the inclusion of multithreaded operations within custom workflows.

## Workflow Styles

Windows Workflow Foundation supports two styles of workflow: sequential and state machine. Before you start creating a workflow project, it is important to figure out what style of workflow you want to use.

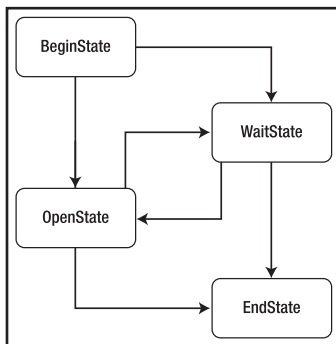
The *sequential* workflow style is used for repetitive operations such as designing a set of activities that must always be performed in the same order. The workflow follows the sequence until the last activity is completed, and the workflow always remains in control of what happens next. Such workflows are not necessarily entirely deterministic; you can use branching logic such as `IfElseActivity` activities or `ParallelActivity` activities and let the exact sequence of events vary. For example, imagine you are creating a workflow that describes the process of installing software. This is a typical scenario where you always follow the steps one by one and in the same order. Therefore, you can represent this process using a sequential workflow. Figure 4-3 shows an example sequential workflow.



**Figure 4-3.** *Sequential workflow*

The *state machine* workflow style is used when you want to model your workflow as a state machine. State machine workflows are made up of different states, one being the starting state, which is the state that is entered once the workflow process begins. Each state can receive a certain set of events. Based on these events, the workflow can transition from one state to another state. In state machine workflows, the end user is always in control. That is why this type of workflow is also known as a *human-oriented workflow*. It is not required to define a final state, but if you do happen to have a final state in a workflow and the transition is made to that state, the workflow will end.

An example of a state machine workflow is an online shop where one state of the workflow process describes the situation where the customer can submit an order and wait for approval of the credit card data. Another state in the workflow process describes the situation where the customer, for the time being, is satisfied with selecting a couple of products and wants to place the order at a later time. Therefore, each customer finds himself in one of the states in the workflow process and goes to another state depending on his choices. Figure 4-4 shows a state machine workflow.



**Figure 4-4.** *State machine workflow*

Although you certainly can, you do not have to rush out and start creating your own workflows immediately, because, out of the box, SharePoint 2007 ships with a set of default workflows. Table 4-4 provides an alphabetical overview of the default workflows.

**Table 4-4.** *Overview of Default Workflows in SharePoint 2007*

<b>Workflow Name</b>	<b>Description</b>
Approval	This is probably the most basic and often-used workflows you can imagine. You can submit a document for approval; approvers can accept or reject a document or request changes.
Collect Feedback	Gathers feedback information from a group of reviewers.
Collect Signatures	Collects signatures from a group of approvers and requires the use of Microsoft Office clients.
Disposition Approval	Centered on document expiration and retention.
Group Approval	An advanced version of the Approval workflow.
Issue Tracking	Dedicated to an issue process involving tasks, task assignments, and task completion.
Translation Management	Facilitates a translation process involving documents, tasks, and translators.
Three-state	Tracks the status of an item through three states.

## Creating Basic Workflows with SharePoint Designer 2007

With SharePoint Designer, it is possible to design workflows with no-code application logic. By making use of SharePoint Designer's Workflow Designer, you can create rules consisting of conditions and actions that are available out of the box in SharePoint Designer.

### Designing a Workflow

Before you design the workflow, you need to make any necessary changes or customizations to your site, list, or library—for example:

- A SharePoint Designer workflow is always attached to exactly one SharePoint list or library. Your site must have at least one list or library before you create a workflow. If there are no lists in your site, you are prompted to create one when you create a workflow.
- If you want your workflow to use any custom columns or settings, you must make those changes before you create the workflow so that those columns and settings are available to you in the Workflow Designer.
- If you want your workflow to use any list or library features that are not turned on by default, such as Content Approval, you must turn on these features before you design the workflow.

---

**Note** The workflow feature is built on the Microsoft Windows Workflow Foundation, a component of Microsoft Windows. The same version of the Workflow Foundation must be installed on both your computer and the server. The first time you create a workflow, you may be prompted to install the Workflow Foundation.

---



## Rules: Conditions and Actions

With SharePoint Designer, you create rules to add conditional logic to the workflow. A rule in a workflow consists of conditions and actions. A rule sets up a condition, and when this condition is true an associating action will be performed. For instance, you can create a rule with a condition that checks whether a title of a document contains the word *test*, and if it does, an e-mail will be sent to the owner of the document.

Multiple conditions can be added together via And and Or clauses. You can also add more conditional branches to a workflow. With multiple branches, you will get scenarios like the following: if condition A is true then run action A; otherwise when condition A is false, run action B. When more than one conditional branch in your workflow is present, you will see a green diamond shape beside that conditional branch. Figure 4-5 shows a Workflow Designer wizard page with two conditional branches. The first branch has two conditions, an And clause, and two actions that will run in sequence.

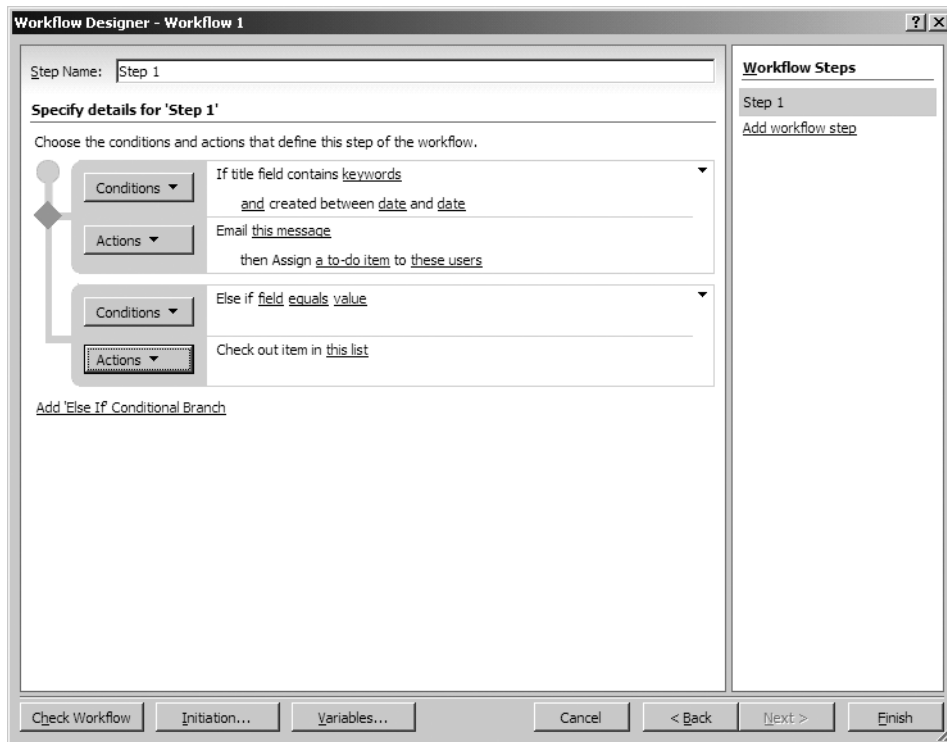


Figure 4-5. The Workflow Designer wizard with conditions and actions in SharePoint Designer

---

**Note** It is not necessary for a conditional branch to contain a condition. A conditional branch, however, must always consist of one or more actions.

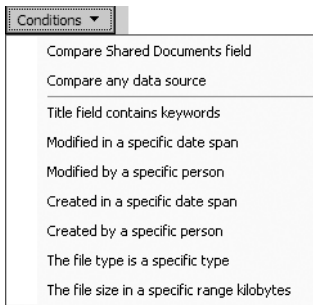
---

There are a lot of conditions available out of the box in SharePoint Designer. Some examples include

- The title field of a list or library contains specified keywords
- The item was created in a specific time span
- The item's file size is in a specific range of kilobytes

There are also conditions available that you can customize just the way you want. For example, you can create a custom condition that checks the value of a field with a custom value, or you can compare a field in the specified list with a value in another list. Conditions are created visually and you cannot add code fragments to these conditions.

The conditions that are available in the Conditions drop-down list depend on whether the workflow will be attached to a SharePoint list or library. You specify this list or library on the first page of the Workflow Designer wizard. Figure 4-6 shows the conditions available for a Shared Document document library.



**Figure 4-6.** *The Workflow Designer wizard with the conditions available for the Shared Documents document library*

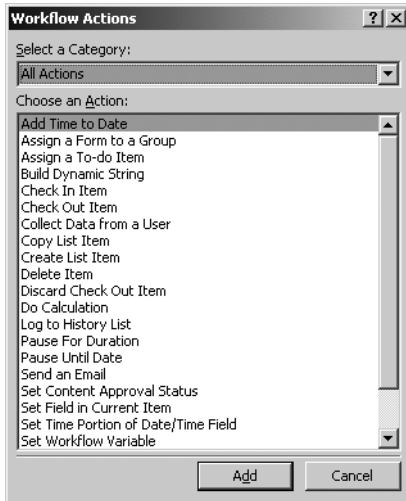
An action is the most basic unit of the workflow. SharePoint Designer contains several out of the box actions. If that is not enough, with Visual Studio 2005 Designer, you can create actions and deploy them to SharePoint Designer where they can be used.

There are 23 actions available in SharePoint Designer. You can use them via the Actions drop-down list. The actions are divided into the following categories:

- *All Actions:* This category contains an overview of all 23 actions available in SharePoint Designer.
- *Core Actions:* This category contains 13 core actions such as “Log to History List” and “Set Workflow Variable.”
- *List Actions:* This category contains 7 list-specific actions such as “Create List Item.”
- *Task Actions:* This category contains 3 task-specific actions such as “Assign a To-do item.”

Figure 4-7 shows an overview of the Workflow Actions window.

Multiple actions can be associated with a condition. It is possible to run the actions in sequence, which means that first action 1 runs and then action 2 will take place, or in parallel, which means action 1 and action 2 will be performed at the same time.



**Figure 4-7.** The Workflow Actions window showing the All Actions category

---

**Note** When you specify that actions run in parallel, you will not have the guarantee that they will be performed simultaneously. The exact order of running cannot be specified; it may vary each time the workflow runs.

---

## Creating an Example Workflow

In this section, we will create a simple example workflow using SharePoint Designer. We are going to create a workflow for an imaginary company's Human Resources department. They have placed an advertisement for a new C# developer and they expect a lot of letters and resumes. The CTO wants to read these letters and approve or decline them. If a letter is approved, the candidate will be invited for an interview. In this example, we are going to create the first part of the process: namely, a document (the letter from the candidate) will be uploaded to a document library called LettersCandidates. This document library will contain two columns: Assigned to and Status. The custom column Assigned to will contain three values: CTO, CEO, and CFO. When a document is uploaded to the document library and assigned to the CTO, the CTO will receive a new task in his Tasks list asking whether he approves or declines the document. The second custom column, Status, contains the following values: prospect, 1st interview, 2nd interview, declined, and hired.

Follow these steps to create a new document library called LettersCandidates and a new column:

1. Open a SharePoint site where you want to create a new document library.
2. Click the View All Site Content link on the left of the SharePoint site. This opens the All Site Content page.
3. Click Create. This opens the Create page.
4. Click Document Library in the Libraries section.
5. Name the document library **LettersCandidates** and click Create.
6. Select Settings in the LettersCandidates document library and click Create Column.

7. Name the column **Status**, check the Choice radio button, and enter the values: **prospect**, **1st interview**, **2nd interview**, **declined**, and **hired**. Click OK.
8. Click Create Column.
9. Name the column **Assigned to**, check the Choice radio button, and enter the values: **CTO**, **CEO**, and **CFO**. Click OK.

After setting up the LettersCandidates document library and adding a column, you are ready to open the SharePoint site in SharePoint Designer. The next thing to do is to start designing the example workflow. Follow these steps to create the workflow.

1. Open the SharePoint site containing the LettersCandidates document library in SharePoint Designer.
2. In the File menu, click New and select Workflow. This opens the Workflow Designer wizard.
3. In the Give a name to this workflow text box, specify the following name: **Candidates Notification WF**.
4. In the What SharePoint list should this workflow be attached to drop-down list, select the LettersCandidates document library.
5. In the Select workflow start options for items in LettersCandidates section, do the following. Uncheck the Allow this workflow to be manually started from an item check box. Check the Automatically start this workflow when an item is created check box and uncheck the Automatically start this workflow when an item is changed check box. This way the workflow will run each time an item is created. Figure 4-8 shows the first page of the Workflow Designer wizard.

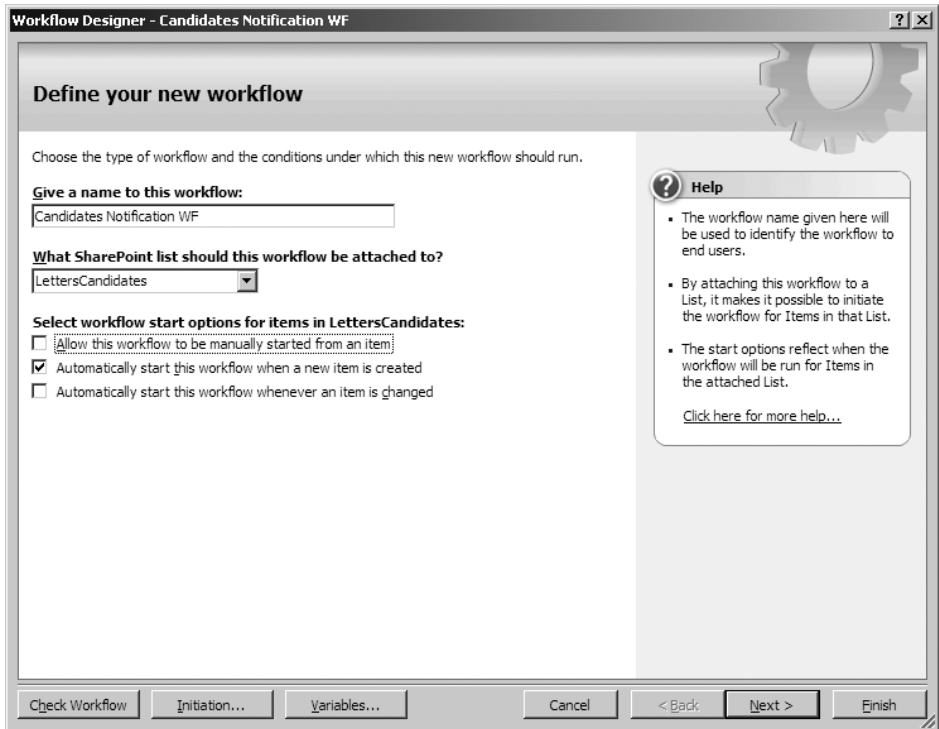
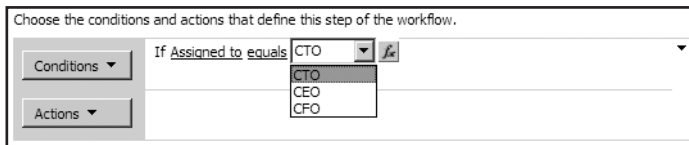


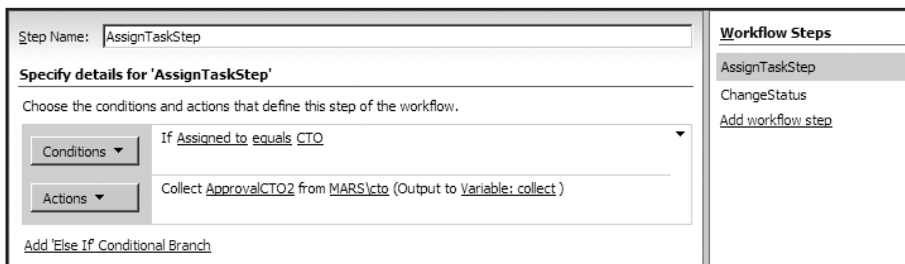
Figure 4-8. The Workflow Designer wizard page 1

6. Click Next.
7. In the Step Name text box, specify the following name: **AssignTaskStep**.
8. Click the Conditions drop-down list and select Compare LettersCandidates Field. The condition if field equals value is inserted, where field and value are the parameters that must be filled in.
9. Click the field parameter and select the Assigned to column. The field parameter shows a drop-down list filled with the specified document library's metadata.
10. Click the value parameter and select the CTO value. The value parameter shows a drop-down list filled with the values from the specified column. Figure 4-9 shows the value parameter's drop-down list.



**Figure 4-9.** The value parameter's drop-down list in page 2 of the Workflow Designer

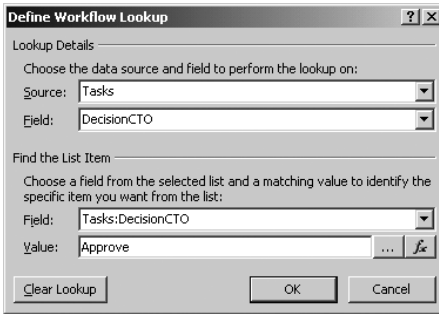
11. Click the Actions drop-down list and select Collect data from a user (output to variable:collect1). The action Collect data from this user is inserted, where data and this user are parameters that must be filled in. The parameter variable:collect1 will be where we save the data that is collected from the user.
12. Click data. This opens the Custom Task wizard. Click Next.
13. In the Name text box, specify the following name: **ApprovalCTO**. Click Next.
14. Click the Add button to define a custom task field that the user can fill out. This opens the Add Field dialog box.
15. In the Field name text box, specify the following name: **DecisionCTO**. In the Information type drop-down list, choose choice (menu to choose from). Click Next.
16. In the Choices list, enter the following two choices: **Approve**, **Decline**.
17. Uncheck the Allow fill-in choices check box. Click Finish twice.
18. Click this user. This opens the Select Users dialog box. Select a user from the list of users, click Add, and click OK. Figure 4-10 shows the AssignTaskStep workflow step.



**Figure 4-10.** The AssignTaskStep workflow step

19. Click Add workflow step. This adds a new workflow step to the workflow.

20. In the Step Name text box, specify the following name: **ChangeStatus**.
21. Click the Conditions drop-down list and select Compare any data source. The condition if value equals value is inserted, where value and value are the parameters that must be filled in.
22. Click the display data binding icon next to the first value parameter. This opens the Define Workflow Lookup window. The Source drop-down list shows the SharePoint site's available lists. The Field drop-down list shows all fields available in the specified list. In the Source drop-down list, select Tasks. In the Field drop-down list, select DecisionCTO. In the Value drop-down list, select Approve, as shown in Figure 4-11.



**Figure 4-11.** The Define Workflow Lookup window

23. Click OK.
24. Click the Actions drop-down list and select Set Field in Current Item. The action Set field to value is inserted, where field and value are parameters that must be filled in. Click the field parameter and select the Status value. The field parameter shows a drop-down list filled with the specified document library's metadata.

---

**Note** It is not possible to create custom activities with SharePoint Designer. However, it is possible to make use of custom activities created using Visual Studio 2005 in SharePoint Designer. More information about creating custom activities and using them in SharePoint Designer can be found in the section “Creating and Using a Custom Activity.”

---

25. Click the value parameter and select 1st interview. The value parameter shows a drop-down list filled with the values from the specified column. Figure 4-12 shows you the ChangeStatus workflow step.



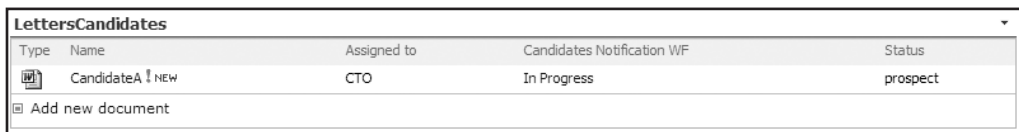
**Figure 4-12.** The ChangeStatus workflow step

26. Click the Check Workflow button to check whether your workflow contains any errors.

**Tip** If there is an error in your workflow, asterisks appear before and after the parameters that were not configured correctly. In addition, in the Workflow Steps section, an error symbol appears next to each step that contains an error.

27. Click Finish. The workflow is validated and associated with the document library.

The next set of figures walks you through the example workflow in a SharePoint site. Figure 4-13 shows you a letter from Candidate A in the LettersCandidates document library. The Candidates Notification Workflow is in progress.

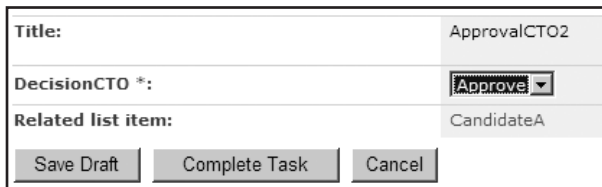


Type	Name	Assigned to	Candidates Notification WF	Status
	CandidateA NEW	CTO	In Progress	prospect

Add new document

**Figure 4-13.** *The LettersCandidates document library*

The workflow will assign a task to the CTO. This task will also be send as an e-mail to the CTO. The task can be edited and the CTO can approve or decline the document. This is shown in Figure 4-14.



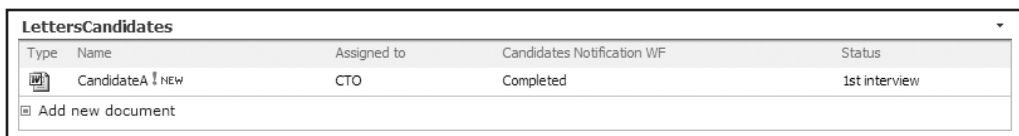
**Title:** ApprovalCTO2

**DecisionCTO \*:**

**Related list item:** CandidateA

**Figure 4-14.** *The LettersCandidates document library*

After the CTO has approved or declined the document, the status of the document will change. See Figure 4-15.



Type	Name	Assigned to	Candidates Notification WF	Status
	CandidateA NEW	CTO	Completed	1st interview

Add new document

**Figure 4-15.** *The LettersCandidates document library*

## Creating Advanced Workflow Solutions with Visual Studio

The team responsible for creating Windows Workflow Foundation has also created a Visual Studio add-in called Visual Studio 2005 Designer that makes it easy to create custom workflows. Visual Studio 2005 Designer can also be used to include custom code in a workflow, or to include custom forms that are used by the workflow to communicate with the workflow users. It is also possible to create custom activities that can be used in other workflows or in SharePoint Designer.

### Getting to Know Visual Studio 2005 Designer

Before even thinking about starting to create workflows, you first have to make sure your development environment is ready for the job. You have to install the following components:

- *Visual Studio 2005 extensions for .NET Framework 3.0 (Windows Workflow Foundation):* You can download the Visual Studio 2005 extensions for .NET Framework 3.0 here: <http://www.microsoft.com/downloads/details.aspx?familyid=5D61409E-1FA3-48CF-8023-E8F38E709BA6&displaylang=en>.
- *Windows SharePoint Services 3.0: Software Development Kit:* You can download the Windows SharePoint Services 3.0 SDK here: <http://www.microsoft.com/downloads/details.aspx?FamilyId=05E0DD12-8394-402B-8936-A07FE8AFAFFD&displaylang=en>.

---

**Note** Support for Workflow Services in Visual Studio 2008 (the next version of Visual Studio 2005) and .NET 3.5 has been improved. The most important new feature is the integration between Workflow Services and Windows Communication Foundation. In .NET 3.5, it is easy to invoke WCF services within WF workflows. Another interesting feature is the ability to expose a WF workflow as a WCF service.

---

Make sure you install the following components of the Visual Studio 2005 extensions for .NET Framework 3.0:

- Visual Studio 2005 Designer for Windows Workflow Foundation (also known as the Visual Studio 2005 Workflow Designer)
- Windows Workflow Foundation Debugger

---

**Note** Visual Studio 2005 extensions for .NET Framework 3.0 requires the final version either of Windows Workflow Foundation Runtime Components, Microsoft Windows Vista, or the .NET Framework 3.0 Runtime Components. The .NET Framework 3.0 Runtime Components can be downloaded here: <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=10CC340B-F857-4A14-83F5-25634C3BF043>.

---

The Windows SharePoint Services 3.0 SDK is very helpful if you are developing solutions based on Windows SharePoint Services. The WSS 3.0 SDK contains conceptual overviews, programming tasks, and references. The WSS 3.0 SDK also includes the Workflow Developer Starter Kit for Windows SharePoint Services 3.0.

The Workflow Developer Starter Kit for Windows SharePoint Services 3.0 contains the following items:



- Visual Studio Project Templates
  - Sequential Workflow Library
  - State Machine Workflow Library
- Sample Custom Workflow
  - Simple Collect Feedback using ASPX forms

Visual Studio 2005 Designer can be used to create two types of workflows: state machine workflows and sequential workflows. A state machine workflow is a workflow that contains a number of different states where one state always acts as the starting point. Each state in such a workflow receives events, which lead to the transition to the next state. A sequential workflow is a workflow that contains a fixed sequence of steps that always must be followed in the same order.

After installing the WSS 3.0 SDK and the Visual Studio 2005 extensions for .NET Framework 3.0, two new project types appear in the Project types list of the Visual Studio 2005 New Project dialog box. Figure 4-16 shows the Workflow project types.

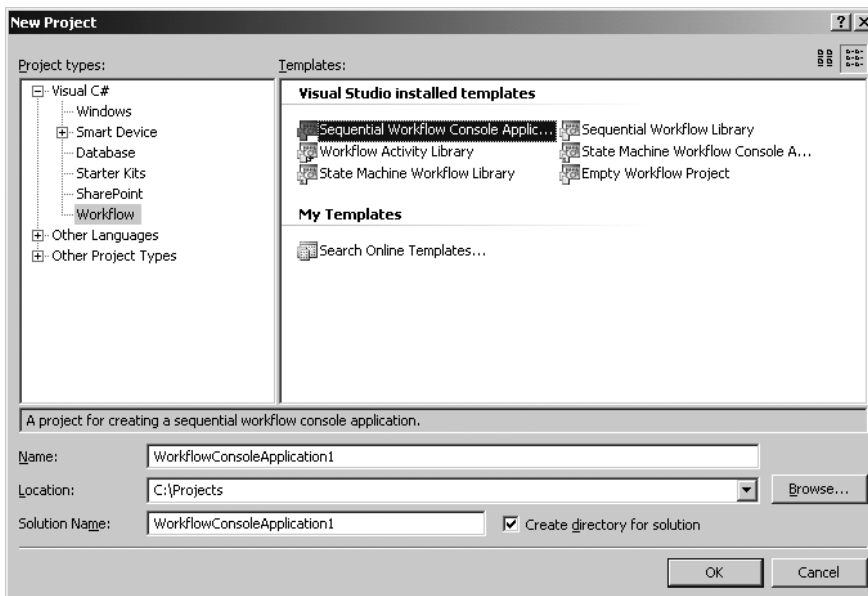


Figure 4-16. Workflow project types in Visual Studio 2005

The Workflow project types section contains the following Visual Studio 2005 project templates:

- *Sequential Workflow Console Application*: This project is a starting point for creating sequential workflows hosted by a console host application.
- *Workflow Activity Library*: This project is a starting point for creating your own activities that can be reused later as building blocks in a workflow.
- *State Machine Workflow Library*: This project is a starting point for creating your own state machine workflow in the form of a library (.dll).
- *Sequential Workflow Library*: This project is a starting point for creating your own sequential workflows in the form of a library (.dll).

- *State Machine Workflow Console Application*: This project is a starting point for creating state machine workflows hosted by a console host application.
- *Empty Workflow Project*: This project is an unconfigured workflow project that can include multiple workflows and and/or activities.

The Workflow project types let you create generic workflow projects, but whenever you want to create a specific SharePoint workflow, you are better off if you choose one of the SharePoint workflow project types. Figure 4-17 shows the SharePoint project types.

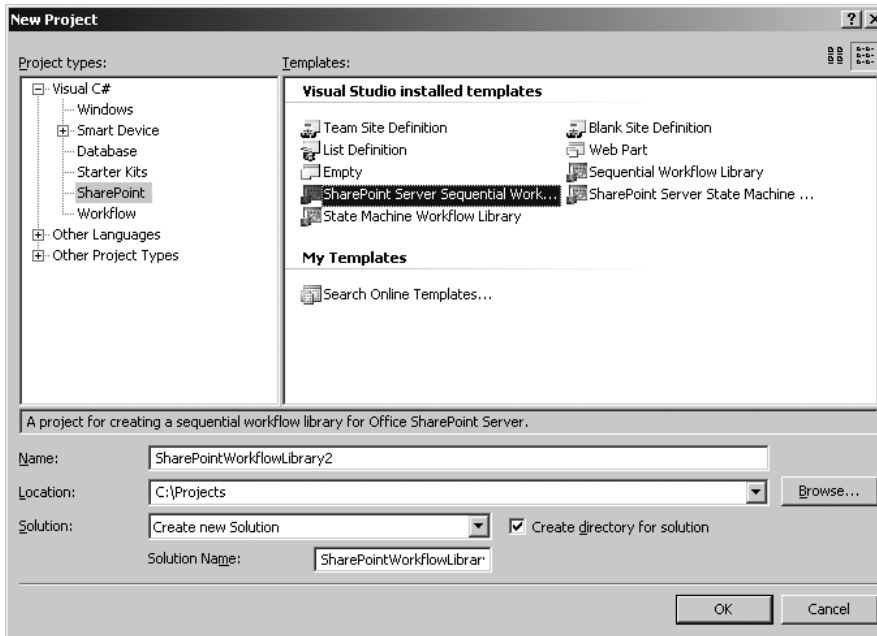


Figure 4-17. SharePoint project types in Visual Studio 2005

The SharePoint project types section contains the following Visual Studio 2005 workflow project templates:

- *SharePoint Server Sequential Workflow Library*: This project is a starting point for creating your own sequential workflows for Office SharePoint Server in the form of a library (.dll).
- *Sequential Workflow Library*: This project is a starting point for creating your own sequential workflows for Windows SharePoint Services in the form of a library (.dll).
- *SharePoint Server State Machine Workflow Library*: This project is a starting point for creating your own state machine workflows for Office SharePoint Server in the form of a library (.dll).
- *State Machine Workflow Library*: This project is a starting point for creating your own state machine workflows for Windows SharePoint Services in the form of a library (.dll).

Visual Studio 2005 Designer for Windows Workflow Foundation contains a visual designer that you can use to create a workflow. You can drag and drop activities from the toolbox onto the design surface, where you can configure them.

---

**Note** You are now looking at a domain-specific language (DSL) at work. A DSL is a concept closely related to software factories. If you want to find out more about software factories and DSLs, refer to Chapter 5.

---

The Visual Studio 2005 Designer for Windows Workflow Foundation should be familiar to experienced Visual Studio users. In addition to the design surface, there are other windows available, such as the toolbox window that contains graphical representations of activities, the properties window that lets you configure activity properties, the Solution Explorer window, and the standard Visual Studio debug windows.

## Creating an Example Workflow

Once you are happy with the setup of your development environment, you are ready to move to the design phase. You could make a sketch on paper or design a model using Office Visio, or do whatever works for you, as long as you have a clear concept about what you want to accomplish in the workflow. As soon as you have reached that point, you are ready to move to the development phase.

---

**Tip** When you create a workflow for Microsoft Office SharePoint Server or Windows SharePoint Services, it is best to develop on a machine hosting a SharePoint server itself. This way it becomes much easier to deploy and debug the workflow.

---

The following procedure describes the steps you have to follow to develop SharePoint workflows or workflow activities with Visual Studio 2005 Designer.

- Create a workflow or workflow activity.
- Create and publish custom forms.
- Create a feature definition file and a workflow definition file. These files contain information about the workflow assembly and are used to bind the custom forms to the workflow.
- Make sure the workflow assembly is compiled as a strongly named assembly.

---

**Note** A strong-named assembly is an assembly that has a fully qualified name. Fully qualified names contain the following information: assembly name, version number, culture, and a public key token that can be used to uniquely identify the developer of the assembly.

---

- Deploy the workflow.
- Debug the workflow if necessary.

In this section we will discuss how to create a Microsoft Office SharePoint Server workflow with Visual Studio 2005 Designer. The other steps (deploying and debugging) will be discussed later in the sections “Deploying a Workflow” and “Debugging a Workflow.” To develop a custom workflow for Microsoft Office SharePoint Server, you start by creating a SharePoint workflow project in Visual Studio 2005 Designer.

The example workflow we are going to create will be a sequential workflow for Microsoft Office SharePoint Server that can be associated with a list or library. When a document is uploaded into the document library or changed in the document library, a task will be created for a specified person.

This task contains predefined instructions and comments. When the user opens the task, she will get a custom-designed InfoPath form with instructions to complete the task. We are going to create two custom InfoPath forms, one for associating the workflow to the list or library and specifying the name of the user who will get the task, and another for completing the custom task form (and the workflow).

## Creating an Initiation Form

The first thing we are going to do is to create an initiation form, which will double as an association form. More information about forms can be found in the section “Using Forms to Capture and Automate Your Processes.” The initiation form will be shown when a SharePoint item is associated with a workflow. We are going to create the initiation form using InfoPath 2007.

---

**Tip** It is also possible to create custom ASP.NET forms to interact with the user in your workflow. This allows developers to choose the development model they are most comfortable with. InfoPath forms are easier to create and more powerful. When creating a workflow for Windows SharePoint services 3.0, you do not have this option: you can only use ASP.NET forms. Custom forms will be displayed to the user during some stage in the workflow. You can find more information about forms in the section “Using Forms to Capture and Automate Your Processes.”

---

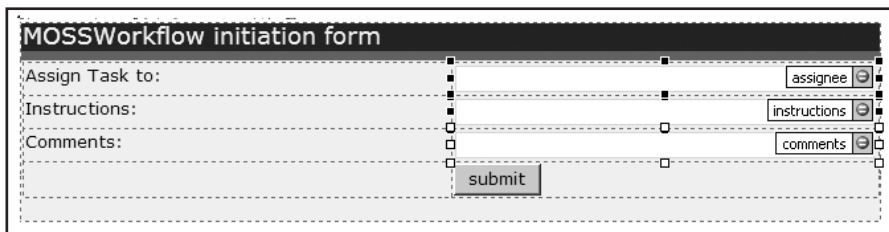
The following procedure shows how to create an initiation form.

1. Open InfoPath by clicking All Programs ► Microsoft Office ► Microsoft InfoPath 2007. This opens the Getting Started window.
2. Click Design a Form Template in the Design a Form section. This opens the Design a Form dialog box.
3. Select Form Template in the Design a new section.
4. Select Blank in the Based on area.
5. Make sure the Enable browser-compatible features only check box is checked.
6. Click OK.

Now that we have created an empty InfoPath form, we are going to design this form and add three text boxes to it. The first text box contains the name of the person to whom the task is assigned, the second text box allows the user to enter instructions, and the last text box allows the user to specify additional comments. The form will also contain a Submit button. The following procedure shows how to design such a form in InfoPath.

1. In the Design Tasks pane, click Layout.
2. Drag a Table with Title onto the form. Enter the following title: **MOSSworkflow initiation form**.
3. Drag a Custom Table with two columns and four rows onto the form.
4. In the Design Tasks tool pane, click Controls.
5. In the left column of the first row, type the following: **Assign Task to:**
6. Drag a Textbox control onto the form in the right column of the first row. Right-click the Textbox and select Text Box Properties. In the Field Name text box, enter the following value: **assignee**. Click OK.
7. In the left column of the second row, type the following: **Instructions:**

8. Drag a Textbox control onto the form in the right column of the second row. Right-click the Textbox and select Text Box Properties. In the Field Name text box, enter the following value: **instructions**. Click OK.
9. In the left column of the third row, type the following: **Comments:**.
10. Drag a Textbox control onto the form in the right column of the third row. Right-click the Textbox and select Text Box Properties. In the Field Name text box, enter the following value: **comments**. Click OK.
11. Drag a Button control onto the form in the right column of the fourth row. Right-click the Button and select Button Properties. In the Label text box, enter the following value: **submit**. Click OK. Figure 4-18 shows the way the InfoPath form should look.



**Figure 4-18.** Custom workflow initiation form

Next, we are going to give the form fields collection (all the text boxes on the form) a unique name. We do this because we want to extract the XML schema (.xsd) file for this form and use this schema as the basis for the class we are going to use in the `OnWorkflowActivation` activity. The root element of the schema file will have the same name as the form fields collection. The class that will be generated from the schema file will have the same name as the schema file root element. Therefore, when you specify a unique name for the form fields collection, you will get a class with a unique name. This is especially important when you are programming a workflow that deserializes multiple forms. Follow these steps to configure the InfoPath form.

1. Go to the Design Tasks pane and click Data Source.
2. Double-click the `myFields` fields collection. This opens the Field or Group Properties dialog window.
3. Enter the following value in the Name text box: **InitForm**.
4. Click OK.

We are going to add two rules to the InfoPath form's Submit button. The first rule submits the form information to Microsoft Office SharePoint Server 2007; the other rule closes the form when the user clicks the Submit button. Follow these steps to add these two rules to the Submit button.

1. Right-click the Button control and select Button Properties. This opens the Button Properties dialog window.
2. On the General tab, click the Rules button. This opens the Rules dialog window.
3. Click the Add button. This opens the Rule dialog window.
4. Click the Add Action button. This opens the Action dialog window.
5. Select Submit using a data connection from the Action drop-down list. Click the Add button. This opens the Data Connection Wizard dialog window.

6. Select Create a new connection and select the Submit data radio button. Click Next.
7. Select To the hosting environment and click Next.
8. Click Finish.
9. Click OK twice.
10. In the Rules dialog window, click Add to add another rule. This opens the Rule dialog window.
11. Click the Add Action button. This opens the Action dialog window.
12. Select Close the form from the Action drop-down list. You will get a message stating the following: Prompting the user to save on close is not supported.

---

**Note** In this scenario, we do not want to support save on close, since this form will be rendered to HTML by InfoPath Forms Server 2007. As a result, the form is shown in the browser. Closing the form equates to closing the entire browser, which is undesirable. We want the user to save the form, not close the browser.

---

The next thing to do is to set the form's security level to a mode that is supported by InfoPath Forms Server. This allows InfoPath Forms Server to host the form. The security model for InfoPath is related to the security settings in Internet Explorer. InfoPath has three security levels for forms: Restricted, Domain, and Full Trust. We are going to give this form a Domain security level, so the form can access content that is stored in the form itself.

---

**Tip** More information about InfoPath and MOSS 2007 can be found in Chapter 8.

---

The next procedure shows you how to change the security level of a form.

1. Select Tools from the menu and click Forms options. This opens the Forms Options dialog window.
2. Select the Security and Trust category from the Category section.
3. Uncheck the Automatically determine security level check box and select Domain. Click OK.

The next procedure shows how to publish the initiation form. By publishing the initiation form to the workflow project's Feature files directory, you allow the initiation form to be deployed as part of the workflow feature.

1. In the File menu, click Publish. This opens the Save As dialog window.
2. Enter the following value in the FileName text box: **InitiationForm**. Then, click Save.
3. Select To a network location and click Next.
4. Click the Browse button to browse to where you keep your Visual Studio project and go to the Deployment Files\Feature Files directory to publish the form. Then, click OK.
5. Enter the following value in the Form template name text box: **InitiationForm**. Click Next.
6. Make sure the alternate access path text box is empty and click Next. This way, the initiation form can only be accessed within the workflow process (as opposed to accessing the form directly from within a browser).
7. Click Publish and then click Close.

To get the data from the submitted form into the workflow, we have to generate a class using `xsd.exe`. The Microsoft .NET Framework 2.0 command-line tool `xsd.exe` lets you generate a new class file based on the `myschema.xsd` form schema. The `xsd.exe` tool is located by default in the following location: `[drive letter]:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin`. Follow these steps to generate a new class file that is based on the form schema file.

1. Select **Save as Source Files** in the **File** menu. Browse to the location where you want to save the InfoPath source files and click **OK**.

---

**Note** An InfoPath template consists of a collection of form source files, including the schema file. By default, the form schema file is named `myschema.xsd`. You can find more information about InfoPath in Chapter 8.

---

2. Go to **Start** ► **All Programs** ► **Microsoft Visual Studio 2005** ► **Visual Studio Tools** ► **Visual Studio 2005 Command Prompt**. This opens a Visual Studio 2005 Command Prompt window.
3. From the command prompt, navigate to the location of the `myschema.xsd` file and enter the following line of code.

```
xsd myschema.xsd /c
```

4. This generates a new class file based on the form schema called `myschema.cs`. Rename the `schema.cs` file to `InitForm.cs`.
5. Add the `InitForm.cs` class file to the workflow project.

Listing 4-1 shows the content of the `InitForm.cs` file. As you will notice, the name of the class is identical to the name of the form fields collection in the InfoPath form.

**Listing 4-1.** *Generated Code Class from `myschema.xsd`*

```
using System.Xml.Serialization;

[System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
[System.Xml.Serialization.XmlRootAttribute(
    Namespace="http://schemas.microsoft.com/office/
    infopath/2003/myXSD/2007-05-25T12:13:02", IsNullable=false)]
public partial class InitForm
{
    private string assigneeField;
    private string instructionsField;
    private string commentsField;
    private System.Xml.XmlAttribute[] anyAttrField;

    public string assignee
    {
        get { return this.assigneeField; }
        set { this.assigneeField = value; }
    }
}
```

```

public string instructions
{
    get { return this.instructionsField; }
    set { this.instructionsField = value; }
}

public string comments
{
    get { return this.commentsField; }
    set { this.commentsField = value; }
}

[System.Xml.Serialization.XmlAnyAttributeAttribute()]
public System.Xml.XmlAttribute[] AnyAttr
{
    get { return this.anyAttrField; }
    set { this.anyAttrField = value; }
}
}

```

## Creating a Custom Task Form

The second form we are going to create is the task form. This form enables users of the workflow to interact with the task that is assigned to them. More information about forms can be found in the section “Using Forms to Capture and Automate Your Processes.” Again, as we did in the “Creating an Initiation Form” section, we are going to create the new form using InfoPath 2007. The next procedure shows how to create a workflow task form in InfoPath 2007.

1. Open InfoPath by clicking All Programs ► Microsoft Office ► Microsoft InfoPath 2007. This opens the Getting Started window.
2. Click Design a Form Template in the Design a Form section. This opens the Design a Form dialog box.
3. Select Form Template in the Design a new section.
4. Select Blank in the Based on area.
5. Make sure the Enable browser-compatible features only check box is checked.
6. Click OK.

We are going to place one text box, one check box, and one button on the task form. The text box displays instructions to a user. This text box will be prepopulated with data from the initiation form. The check box is used to indicate that the workflow task is completed; the OK button closes the task form. The next procedure shows how to design the task form in InfoPath.

1. In the Design Tasks pane, click Layout.
2. Drag a Table with Title onto the form. Enter the following title: **MOSSworkflow edit task form**.
3. Drag a Custom Table with two columns and three rows onto the form.
4. In the Design Tasks tool pane, click Controls.
5. In the left column of the first row, type the following: **Instructions**.
6. Drag a Textbox control onto the form in the right column of the first row. Right-click the Textbox and select Text Box Properties. In the Field Name text box, enter the following value: **instructions**. Click OK.



7. In the left column of the second row, type the following: **Task completed**.
8. Drag a Check Box control onto the form in the right column of the second row. Right-click the check box and select Check Box Properties. In the Field Name text box, enter the value **isFinished**, and select True/False from the Data type drop-down list. Click OK.
9. Drag a Button control onto the form in the right column of the third row. Right-click the Button and select Button Properties. In the Label text box, enter the following value: **OK**. Click OK. Figure 4-19 shows what the InfoPath form should look like.

**Figure 4-19.** Custom workflow edit task form

We are going to add two rules to the Submit button of the task InfoPath form: one rule to submit the form information to Microsoft Office SharePoint Server 2007, and another rule to close the form when the user clicks the Submit button. Follow these steps to add two rules to the Submit button:

1. Right-click the Button control and select Button Properties. This opens the Button Properties dialog window.
2. On the General tab, click the Rules button. This opens the Rules dialog window.
3. Click the Add button. This opens the Rule dialog window.
4. Click the Add Action button. This opens the Action dialog window.
5. Select Submit using a data connection from the Action drop-down list. Click the Add button. This opens the Data Connection Wizard dialog window.
6. Select Create a new connection and select the Submit data radio button. Click Next.
7. Select To the hosting environment and click Next.
8. Click Finish.
9. Click OK twice.
10. In the Rules dialog window, click Add to add another rule. This opens the Rule dialog window.
11. Click the Add Action button. This opens the Action dialog window.
12. Select Close the form from the Action drop-down list. You will get a message stating the following: Prompting the user to save on close is not supported. This is precisely the way we want it, because we are going to use InfoPath Forms Services. Click OK four times.

The workflow task form is used to display data from the initiation form. In this example, we show the instructions for the task assignee that were added in the initiation form. You have to add the workflow task schema to the workflow edit task form as a secondary data source to allow Microsoft Office SharePoint Server 2007 to prepopulate the task form's Instructions field. The Item-Metadata.xml file represents the task schema. In the file, each task property that you want to use as data in your form must be defined. You can define a task property by adding an attribute with prefix `ows_` and the name of the task property. In this example, that will be `ows_instructions`. The value of

the attribute must be set to an empty string, like this: `ows_instructions=""`. The next procedure shows you how to create the `ItemMetadata.xml` file.

1. Open the text editor of your choice, for example, Notepad.
2. Create a file with the following name: **ItemMetadata.xml**.
3. Add the following line of XML to the file.

```
<z:row xmlns:z="#RowsetSchema"
  ows_instructions=""
/>
```

4. Save the `ItemMetadata.xml` file.

---

**Note** The name of this file must always be `ItemMetadata.xml`, and the filename is case sensitive. Microsoft Office SharePoint Server 2007 always sends task data XML to the task edit form. Therefore, you must always add the `ItemMetadata.xml` file as a secondary data source. Otherwise, you will receive an error.

---

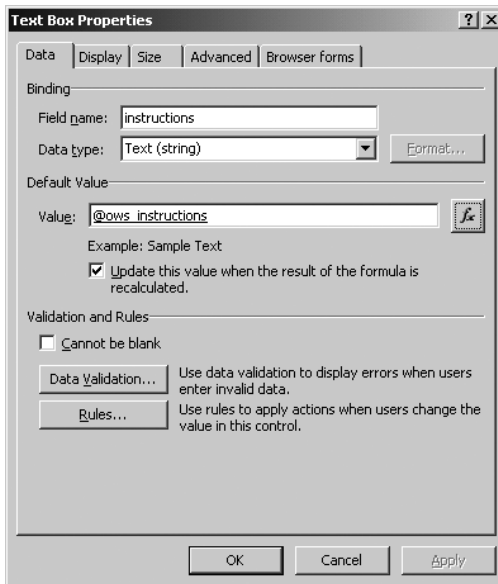
The following procedure shows you how to add the `ItemMetadata.xml` file to the task edit form as a secondary data source.

1. In the Design Tasks pane, click Data Source.
2. Click Manage Data Connection in the section Actions. This opens the Data Connections dialog box.
3. Click Add. This opens the Data Connection Wizard.
4. Select Create a new connection to and select Receive data. Click Next.
5. Select XML document and click Next.
6. Browse to the location where you have saved the `ItemMetadata.xml` file, select the file, and click Next.
7. Select the Include the data as a resource file in the form template or template part radio button and click Next.
8. In the Enter a name for this data connection text box, enter the following name: **ItemMetadata**.
9. Make sure the Automatically retrieve data when form is opened check box is checked. Click Finish and then click Close.

If the `ItemMetadata.xml` file is included as a resource file in the form, like we just did, we no longer need this file anymore. So you do not have to include the `ItemMetadata.xml` file in the workflow solution. The next thing to do is to bind the instructions Textbox control to the `instructions` attribute in the workflow schema that we have added as a data source. The following procedure shows you how to do this.

1. Right-click the instructions text box and select Text Box Properties. This opens the Text Box Properties dialog box.
2. Select the Data tab. In the Default value section, next to the Value text box, click the formula button. This opens the Insert Formula dialog box.
3. Click the Insert a Field or Group button. This opens the Select a Field or Group dialog box.
4. Select the `ItemMetadata` (secondary) data source from the Data Source drop-down list.

5. Select the `ows_instructions` attribute and click OK. Figure 4-20 shows the Text Box Properties dialog box.



**Figure 4-20.** Databinding the instructions text box to the instructions attribute

6. Click OK two more times.

At this time, the instructions text box is bound to the instructions workflow task property. When the task edit form is loaded by Microsoft Office SharePoint Server 2007, this form will display the task instructions in the text box.

The next thing we need to do is set the security level of the form so it can be hosted by InfoPath Forms Server and publish the task edit form. The next procedure shows you how to do this.

1. Select Tools from the menu and click Forms options. This opens the Forms Options dialog window.
2. Select the Security and Trust category from the Category section.
3. Uncheck the Automatically determine security level check box and select Domain. Click OK.

---

**Note** When the Automatically determine security level option is checked, InfoPath will automatically select the appropriate security level for the functionality of the form. This way, the minimum level of trust that is needed for the form is enabled.

---

4. On the File menu, click Publish. This opens the Save As dialog window.
5. Enter the value `EditTaskForm` in the FileName text box and click Save.
6. Select To a network location and click Next.
7. Click the Browse button to browse to the place where you keep your Visual Studio project, go to the `Deployment Files\Feature Files` directory to publish the form, and click OK.

8. Enter `EditTaskForm` in the Form template name text box and click Next.
9. Make sure the alternate access path text box is empty and click Next. This way, the task edit form can only be accessed from within the workflow process.
10. Click Publish and click Close.

## Developing the Workflow

Developing a workflow consists of adding and configuring activities and adding code behind these activities. The first step is creating a new SharePoint workflow project. The next procedure shows you how to create a new SharePoint workflow project.

1. Open Visual Studio 2005.
2. Click the File menu, click New, and select Project. This opens the New Project dialog box.
3. Select the SharePoint project type.
4. In the New Project dialog box, you have the choice between four different workflow project templates, as can be seen in Figure 4-25. Choose the SharePoint Server Sequential Workflow Library project type.
5. Type the following name in the Name text box: `MOSSWorkflow`.
6. Click OK.

Visual Studio 2005 Designer opens a new SharePoint workflow project. This workflow project contains a reference to the `Microsoft.SharePoint.dll`, so you have access to the Microsoft Windows SharePoint Services 3.0 object model in your code. It also contains a reference to the `Microsoft.SharePoint.WorkflowActions.dll` to enable access to predefined Microsoft Office SharePoint Server activities.

Table 4-5 shows an overview of the Microsoft Office SharePoint Server–specific activities that are added to the Toolbox pane. These activities are specially designed for use within Visual Studio 2005 Designer.

---

**Note** Some of the activities mentioned in Table 4-5 (all activities starting with `On*`) implement the `IEventActivity` interface, which provides methods and properties for event-driven activities that allow consumers to subscribe to events. Such activities are important when building SharePoint workflows. Human-oriented workflows such as the ones you will be building in SharePoint typically run for an extended period of time, during which end users cannot proceed with the execution of the workflow until another end user has completed a task. To be able to build such workflows, you need to rely on activities that can support asynchronous tasks.

---

**Table 4-5.** *Microsoft Office SharePoint Server Activities*

Activity Name	Description
CompleteTask	Some activities add tasks to a SharePoint Tasks list once the workflow is started. The Approval workflow that is available out of the box is an example of such a workflow. The CompleteTask activity marks a task as completed.
CreateTask	The CreateTask activity adds a task to the SharePoint Tasks list, along with specific task properties.
CreateTaskWithContentType	The CreateTaskWithContentType activity adds a task of a specific content type to the SharePoint Tasks list.
DelayFor	The DelayFor activity delays the workflow for a specified duration.
DelayUntil	The DelayUntil delays the workflow until the specified date.
DeleteTask	The DeleteTask activity deletes a task from the SharePoint Tasks list.
EnableWorkflowModification	The EnableWorkflowModification activity enables a workflow modification form so that it can be used within the workflow.
InitializeWorkflow	The InitializeWorkflow activity initializes a new instance of the workflow.
LogToHistoryListActivity	The LogToHistoryListActivity activity logs audit information in the workflow's History List.
OnTaskChanged	The OnTaskChanged activity responds to the <code>ITaskService.OnTaskChanged</code> event that is raised by Microsoft Office SharePoint Server 2007 when a task associated with the workflow is edited. This activity implements the interface <code>IEventActivity</code> and allows you to perform asynchronous tasks.
OnTaskCreated	The OnTaskCreated activity responds to the <code>ITaskService.OnTaskCreated</code> event that is raised by Microsoft Office SharePoint Server 2007 when a task that is associated to the workflow is created. This activity implements the interface <code>IEventActivity</code> and allows you to perform asynchronous tasks.
OnTaskDeleted	The OnTaskDeleted activity responds to the <code>ITaskService.OnTaskDeleted</code> event that is raised by Microsoft Office SharePoint Server 2007 when a task that is associated with the workflow is deleted. This activity implements the interface <code>IEventActivity</code> and allows you to perform asynchronous tasks.
OnWorkflowActivated	This activity responds to the <code>ISharePointService.OnWorkflowActivated</code> event that is raised by Microsoft Office SharePoint Server 2007 when a workflow instance is initiated for a SharePoint item. The OnWorkflowActivated activity must always be the first activity in a Microsoft Office SharePoint Server 2007 workflow and will be ignored in the workflow if it is not the first activity. This activity takes care of the initialization of the correlation between the workflow id and the correlation token. This activity implements the interface <code>IEventActivity</code> and allows you to perform asynchronous tasks.

*Continued*

Table 4-5. *Continued*

Activity Name	Description
OnWorkflowItemChanged	The OnWorkflowItemChanged activity responds to the event that is raised by Microsoft Office SharePoint Server 2007 when an item in the list is changed. This activity implements the interface IEventActivity and allows you to perform asynchronous tasks.
OnWorkflowItemDeleted	The OnWorkflowItemDeleted activity responds to the event that is raised by Microsoft Office SharePoint Server 2007 when an item in the list is deleted. This activity implements the interface IEventActivity and allows you to perform asynchronous tasks.
OnWorkflowModified	The OnWorkflowModified activity responds to the event that is raised by Microsoft Office SharePoint Server 2007 when a user submits a modification form. This activity implements the interface IEventActivity and allows you to perform asynchronous tasks.
RollbackTask	The RollbackTask activity rolls back a task that was previously modified.
SendEmail	The SendEmail activity creates and sends an e-mail message to specified users.
SetState	The SetState activity sets the status of the workflow, as shown in the status column of the list the workflow operates on.
SharePointSequentialWorkflowActivity	The SharePointSequentialWorkflowActivity enables execution of multiple activities in a given event.
UpdateAllTasks	The UpdateAllTasks activity updates all tasks in the SharePoint Tasks list.
UpdateTask	The UpdateTask activity updates a task in the SharePoint Tasks list.

Figure 4-21 shows the MOSSWorkflow project in Visual Studio 2005 Designer. You will notice that there are a number of files that are created by default:

- *Feature.xml*: The feature definition file feature.xml contains information necessary to deploy the workflow. You can learn more about the feature.xml file in the section “Deploying a Workflow.”
- *Workflow.xml*: The workflow definition file workflow.xml contains information about the workflow that SharePoint requires to instantiate and run the workflow. More about the workflow.xml file can be found in the section “Deploying a Workflow.”
- *Manifest.xml*: The manifest file is used by Visual Studio 2005 and defines how to populate the cabinet file that is used to deploy the workflow project. Please refer to Chapter 1.
- *Wsp\_structure.ddf*: This file specifies the structure of the .wsp solution cab file.
- *Workflow1.cs*: This class file contains the actual workflow code and components.
- *PostBuildActions.bat*: This batch file is used to deploy the workflow. If you want to learn more about deploying a workflow, please refer to the section “Deploying a Workflow.”



**Figure 4-21.** A SharePoint Server Sequential Workflow Library project in Visual Studio 2005 Designer

The workflow you are creating for Microsoft Office SharePoint Server 2007 will have access to the complete context of SharePoint. In other words, your workflow can write to lists and libraries, change properties, and create, edit, and delete items. Your workflow can do everything that the user account running the workflow is authorized to do.

We are going to add the following activities to the design view of the workflow1.cs file: CreateTask, While, OnTaskChanged, and CompleteTask. The next procedure shows how to add activities to a workflow.

1. Double-click the workflow.cs file to open it in design view. By doing this, you are opening the so-called Workflow Designer.
2. As discussed in Table 4-5, every workflow starts with an OnWorkflowActivated activity. By default, each workflow contains the OnWorkflowActivated activity.
3. Double-click the workflow1.cs file in the Solution Explorer. This opens the design window.
4. Drag and drop a CreateTask activity right below the OnWorkflowActivated activity.
5. Drag and drop a While activity below the CreateTask activity.
6. Drag and drop an OnTaskChanged activity in the While activity.
7. Drag and drop a CompleteTask activity below the While activity. The workflow design view should now look like Figure 4-22.

You will notice that each activity has a red exclamation mark in its right corner, which means that the configuration of that activity is incomplete. These exclamation marks indicate that workflow configuration is not complete yet, and the workflow is not ready for use.

To finish off the workflow configuration, we start with the OnWorkflowActivated activity. Right-click OnWorkflowActivated and select Generate Handlers to create an event handler for the Invoked event. We are going to use this event to set up initial variable values. Figure 4-23 shows the Properties task pane of the OnWorkflowActivated activity.

Correlation tokens allow you to identify workflows and workflow tasks in a unique way and map them to the WF host executing the workflows. The workflow engine distills the correlation token from an incoming request to determine which workflow instance the request is meant for and then delivers the request to that instance.

The next thing we are going to do is to set the properties of the other activities one by one. Let us start with the CreateTask activity. After the workflow is started, we want it to create a task for a user; this will take place in the CreateTask activity. The next procedure shows you how to create and bind variables to the properties of the CreateTask activity.

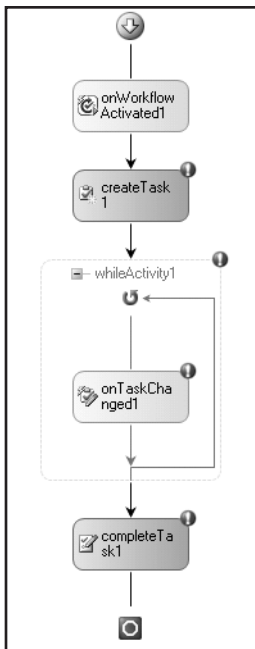


Figure 4-22. The workflow activities

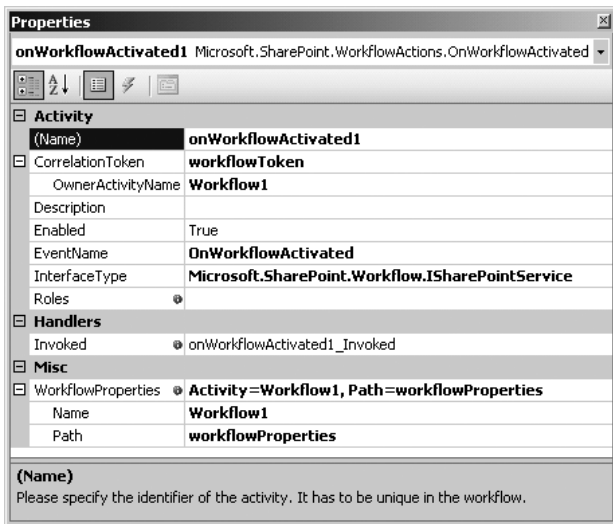


Figure 4-23. The OnWorkflowActivated activity properties

1. Right-click the CreateTask activity and select Properties. This opens the Properties task pane.
2. Enter the value **taskToken** for the CorrelationToken property and hit the Enter key.



3. Select the following value for `OwnerActivityName`: `Workflow1`. `Workflow1` is the name of the workflow class that we are using. By adding the name of the workflow class to the `OwnerActivityName` property, we will ensure that the lifetime of the workflow is correctly managed.
4. Click the default value of the `TaskId` property and click the button with the ellipsis. This opens the Bind ‘`TaskId`’ to an activity’s property dialog box.
5. Select the Bind to a new member tab.
6. Enter the following value in the New member name text box: `taskID`.
7. Select the Create Field radio button and click OK. The `taskID` variable contains a unique id for the task. The following simple member variable is added automatically to the code:

```
public Guid taskID = default(System.Guid);
```

8. Click the default value of the `TaskProperties` property and click the button with the ellipsis. This opens the Bind ‘`TaskProperties`’ to an activity’s property dialog box.
9. Select the Bind to a new member tab.
10. Enter the following value in the New member name text box: `taskProperties`.
11. Select the Create Field radio button and click OK. The `taskProperties` variable contains information about the task. The following simple member variable is added automatically to the code:

```
public SPWorkflowTaskProperties TaskProperties = ➔
new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();
```

The last thing you need to do with the `CreateTask` activity is create an event that fires when the `CreateTask` activity executes. You can create this event by right-clicking the `CreateTask` activity in the design view and selecting `Generate Handlers`. Figure 4-24 shows the `CreateTask` activity’s `Properties` task pane.

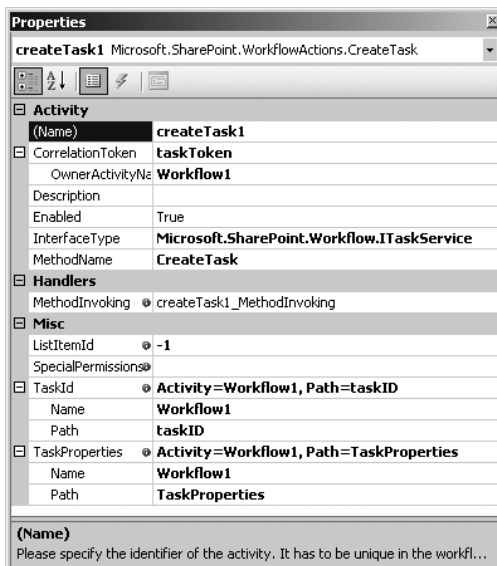


Figure 4-24. The `CreateTask` activity properties

After the workflow is assigned to a user, the workflow engine waits until the user has completed the workflow. The While activity is a loop that contains an OnTaskChanged activity. The responsibility of the While activity is to check whether the task has been changed to complete before moving on to the CompleteTask activity. The OnTaskChanged activity checks whether the task has been changed. Follow these steps to create and bind variables to the properties of the While activity.

1. Right-click the While activity and select Properties. This opens the Properties task pane.
2. Select the following value for the Condition property: **CodeCondition**.
3. Expand the Condition field, and enter the value **notFinished** for the extra field, and press Enter. This creates an event handler where you can check whether the CodeCondition of the While activity is met. Figure 4-25 shows the Properties task pane of the While activity.

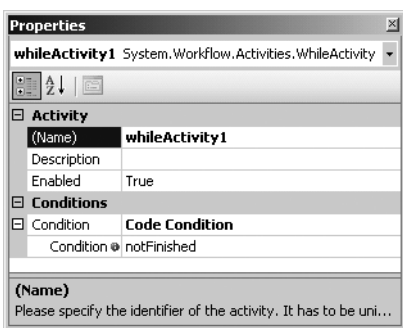


Figure 4-25. The While activity properties

The next activity that we will configure is OnTaskChanged. This activity is located within the While activity. That means that the While loop runs and checks the code condition until the condition is met. In this example, that will be until the notFinished method returns false. Follow these steps to set the properties of the OnTaskChanged activity.

1. Right-click the OnTaskChanged activity and select Properties.
2. Select the following value for CorrelationToken: **taskToken**.
3. Click the default value of the AfterProperties property and click the button with the ellipsis. This opens the Bind 'AfterProperties' to an activity's property dialog box.
4. Select the Bind to a new member tab.
5. Enter the following value in the New member name text box: **afterProperties**.
6. Select the Create Field radio button and click OK. The following variable is added automatically to the code:

```
public SPWorkflowTaskProperties afterProperties = ➡
new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();
```

7. Click the default value of the BeforeProperties property and click the button with the ellipsis. This opens the Bind 'BeforeProperties' to an activity's property dialog box.
8. Select the Bind to a new member tab.
9. Enter the following value in the New member name text box: **beforeProperties**.

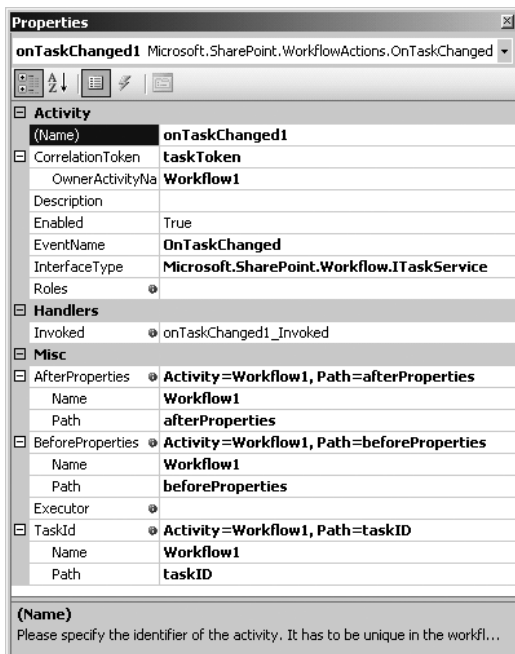
10. Select the Create Field radio button and click OK. The following variable is added automatically to the code:

```
public SPWorkflowTaskProperties beforeProperties =
new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();
```

**Note** You can use the Create Field or Create Property radio buttons. The functionality provided by both controls is comparable. Seen from an object-oriented programming perspective, it is often preferable to create a property that provides better encapsulation.

11. Click the default value of the TaskId property and click the button with the ellipsis. This opens the Bind 'TaskId' to an activity's property dialog box.
12. Select the Bind to an existing member tab.
13. Select taskID and click OK.
14. Right-click the OnTaskChanged activity in the design view and select Generate Handlers to create an event handler for the Invoked event.

Figure 4-26 shows the Properties task pane of the OnTaskChanged activity.



**Figure 4-26.** *The OnTaskChanged activity properties*

The last activity that needs to be configured is CompleteTask. The CompleteTask activity lets you initiate further steps in the process or take actions in other systems. The following procedure shows you which properties of the CompleteTask activity must be set.

1. Right-click the CompleteTask activity and select Properties.
2. Select the following value for the CorrelationToken: taskToken.
3. Click the default value of the TaskId property and click the button with the ellipsis. This opens the Bind 'TaskId' to an activity's property dialog box.
4. Select the Bind to an existing member tab.
5. Select taskID and click OK. Figure 4-27 shows the Properties task pane for the CompleteTask activity.

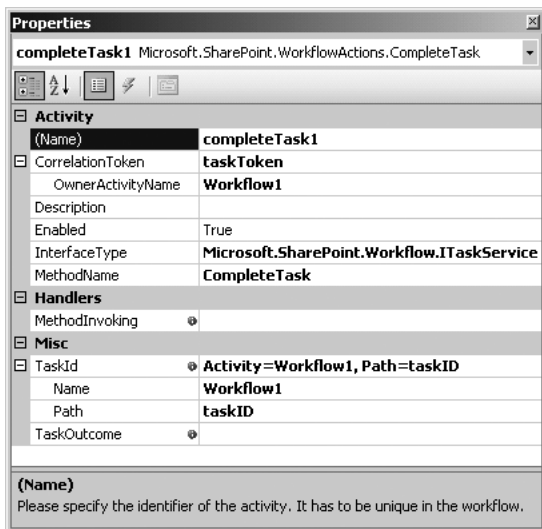


Figure 4-27. The CompleteTask activity properties

After adding and configuring the activities, the next step is adding code to the code view of the workflow1.cs class. We are going to add the following string declarations to the workflow1.cs class:

```
private String assignee = default(String);
private String instructions = default(String);
private String comments = default(String);
```

We will retrieve the values of these three variables from our initiation form. They are passed to the workflow as an XML string represented by the SPWorkflowActivationProperties object's InitiationData property. To access these properties, we need to parse this XML string. We can do this by using the generated class from the schema of the initiation form. We have added a class called InitForm.cs to our workflow project. The following code fragment shows the WorkflowActivated1\_Invoked method that retrieves values from our initiation form:

```
private void onWorkflowActivated1_Invoked(object sender, ExternalDataEventArgs e)
{
    workflowId = workflowProperties.WorkflowId;

    XmlSerializer serializer = new XmlSerializer(typeof(InitForm));
    XmlTextReader reader = new XmlTextReader(
        new System.IO.StringReader(workflowProperties.InitiationData));
    InitForm initform = (InitForm)serializer.Deserialize(reader);

    assignee = initform.assignee;
    instructions = initform.instructions;
    comments = initform.comments;
}
```

The `createTask1_MethodInvoking()` method runs whenever a new task is created. We will add a unique task id (called `taskID`) to the task along with some other properties such as `Title`, `Description`, `comments`, and `instructions`. The `Comments` and `instructions` are custom properties that are passed to the workflow as key-value pairs via the `SPWorkflowTaskProperties` object's `ExtendedProperties` property. The `createTask1_MethodInvoking()` method looks like this:

```
private void createTask1_MethodInvoking(object sender, EventArgs e)
{
    taskID = Guid.NewGuid();
    TaskProperties.Title = "MOSS Workflow Task";
    TaskProperties.AssignedTo = assignee;
    TaskProperties.Description = instructions;
    TaskProperties.ExtendedProperties["comments"] = comments;
    TaskProperties.ExtendedProperties["instructions"] = instructions;
}
```

The last thing we have to do is to make sure is that the `While` loop stops when the task is completed. Therefore, we add a private boolean variable called `IsFinished` to the code.

```
private bool isFinished;
```

In the `CodeCondition` property of the `While` loop, we have specified that the `notFinished()` method is called. One of the arguments of the `notFinished()` method is `ConditionalEventArgs`. `ConditionalEventArgs` has a property called `Result`. If the `Result` property is set to `false`, the `While` loop ends. If the `Result` property is set to `true`, the `While` loop continues. Once the task has been finished, the `isFinished` variable contains the value `true`. The `notFinished()` method looks like this:

```
private void notFinished(object sender, ConditionalEventArgs e)
{
    e.Result = !isFinished;
}
```

The last bit of coding we have to do is set the `isFinished` variable to `true` when the `isFinished` value of the `afterProperties.ExtendedProperties` object is `true`. The `onTaskChanged1_Invoked()` method looks like this:

```
private void onTaskChanged1_Invoked(object sender, ExternalDataEventArgs e)
{
    isFinished = bool.Parse(afterProperties.ExtendedProperties["isFinished"]. ➡
        ToString());
}
```

Listing 4-2 shows the complete code for the `workflow1.cs` class.

**Listing 4-2.** Complete Code for the *workflow1.cs* Class

```

using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Collections;
using System.Drawing;
using System.Workflow.ComponentModel.Compiler;
using System.Workflow.ComponentModel.Serialization;
using System.Workflow.ComponentModel;
using System.Workflow.ComponentModel.Design;
using System.Workflow.Runtime;
using System.Workflow.Activities;
using System.Workflow.Activities.Rules;
using System.Xml.Serialization;
using System.Xml;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Workflow;
using Microsoft.SharePoint.WorkflowActions;
using Microsoft.Office.Workflow.Utility;

namespace MOSSWorkflow
{
    public sealed partial class Workflow1 : SharePointSequentialWorkflowActivity
    {
        public Workflow1()
        {
            InitializeComponent();
        }

        public Guid workflowId = default(System.Guid);
        public Microsoft.SharePoint.Workflow.SPWorkflowActivationProperties workflowProperties =
            new Microsoft.SharePoint.Workflow.SPWorkflowActivationProperties();
        public SPWorkflowTaskProperties afterProperties =
            new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();
        public SPWorkflowTaskProperties beforeProperties =
            new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();

        private String assignee = default(String);
        private String instructions = default(String);
        private String comments = default(String);

        private void onWorkflowActivated1_Invoked(object sender, ExternalDataEventArgs e)
        {
            workflowId = workflowProperties.WorkflowId;
            XmlSerializer serializer = new XmlSerializer(typeof(InitForm));
            XmlTextReader reader = new XmlTextReader(new
                System.IO.StringReader(workflowProperties.InitiationData));
            InitForm initform = (InitForm)serializer.Deserialize(reader);
            assignee = initform.assignee;
            instructions = initform.instructions;
            comments = initform.comments;
        }
    }
}

```

```
public Guid taskID = default(System.Guid);
public SPWorkflowTaskProperties TaskProperties =
new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();

private void onTaskChanged1_Invoked(object sender, ExternalDataEventArgs e)
{
    isFinished = bool.Parse(afterProperties.ExtendedProperties["isFinished"].
    ToString());
}

private bool isFinished;
private void notFinished(object sender, ConditionalEventArgs e)
{
    e.Result = !isFinished;
}

private void createTask1_MethodInvoking(object sender, EventArgs e)
{
    taskID = Guid.NewGuid();
    TaskProperties.Title = "MOSS Workflow Task";
    TaskProperties.AssignedTo = assignee;
    TaskProperties.Description = instructions;
    TaskProperties.ExtendedProperties["comments"] = comments;
    TaskProperties.ExtendedProperties["instructions"] = instructions;
}
}
```

---

**Tip** If you want more information about building workflows for SharePoint environments, you should check out the book *Workflow in the 2007 Microsoft Office System* by David Mann (Apress, 2007).

---

## Deploying a Workflow

In this section, we will discuss the process of deploying your workflow so that it can be used within Microsoft Office SharePoint Server 2007. A custom workflow can be deployed to SharePoint as a feature. In order to do so, you must create the following files: `feature.xml` and `workflow.xml`. However, the first thing you need to do is place the workflow assembly inside the Global Assembly Cache (GAC), which means that the workflow assembly needs to be strong named.

---

**Note** Strong naming is discussed in more detail in Chapter 1.

---

We are going to deploy the example workflow we made in the earlier section “Creating a Workflow.” Follow these steps to sign your assembly:

1. Right-click the `MOSSWorkflow` project and select `Properties`.
2. Select the `Signing` tab and check the `Sign the assembly` check box.
3. Select `<New>` from the `Choose a strong name key file` drop-down list. This opens the `Create Strong Name Key` dialog box.

4. Specify a name in the Key file name text box and uncheck the Protect my key file with a password check box.
5. Click OK.
6. Rebuild the workflow project.

To install the assembly in the Global Assembly Cache, you can use the `gacutil` tool. This tool allows you to view and manipulate the contents of the GAC. The following command installs an assembly in the Global Assembly Cache.

```
gacutil /i [assembly.dll]
```

---

**Tip** You can find more information about the `gacutil` tool at the following URL: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfGlobalAssemblyCacheUtilityGacutil.exe.asp>. Alternatively, you can install an assembly in the GAC by dragging and dropping it to the [drive letter]:\Windows\Assembly folder.

---

## Creating a feature.xml File

In the next step, you have to create a feature definition file called `feature.xml`. This file contains all the information necessary to deploy the workflow. Follow this procedure to create a feature definition file.

1. Open the `Feature.xml` file in Visual Studio 2005.
2. Right-click in the `Feature.xml` file and select **Insert Snippet ► SharePoint Server Workflow ► Feature.xml Code**.

---

**Note** If the code snippets are not loaded in Visual Studio 2005, you can add them yourself. Go to **Tools ► Code Snippets Manager**. Select **XML** in The Language drop-down list. Click **Add**, browse to [drive letter]:\Program Files\Microsoft Visual Studio 8\Xml\1033\Snippets\SharePoint Server Workflow, and click **Open**. Click **OK**. Now you will have the SharePoint Server Workflow snippet available in Visual Studio 2005.

---

The `Id` attribute of the `<Feature>` element contains a global unique identifier (GUID). You have to create your own guid. You can do this by using the `guidgen.exe` tool (which should be available after installing Visual Studio 2005 and C++), or you can create one yourself by calling the `Guid.NewGuid()` method in a C# application and copying the string result to the feature file. The `Scope` attribute contains the scope of the workflow assembly; in this example the scope is set to `Site`, which refers to a site collection-level scope. The other scopes that are available are `Farm`, `WebApplication`, and `Web` (which refers to a site level scope). The `<ElementManifest>` element specifies the relative path to the workflow definition file. Each workflow feature must have a workflow definition file. Listing 4-3 shows what the `feature.xml` file will look like.



**Listing 4-3.** *Content of the feature.xml File*

```
<Feature Id="C3287B4A-2688-4cc6-C34F-92EF5B787BCE"
  Title="MOSSWorkflow"
  Description="Description of the workflow"
  Version="12.0.0.0"
  Scope="Site"
  ReceiverAssembly="Microsoft.Office.Workflow.Feature, Version=12.0.0.0,
  Culture=neutral, PublicKeyToken=71e9bce111e9429c"
  ReceiverClass="Microsoft.Office.Workflow.Feature.WorkflowFeatureReceiver"
  xmlns="http://schemas.microsoft.com/sharepoint/">
  <ElementManifests>
    <ElementManifest Location="workflow.xml" />
  </ElementManifests>
  <Properties>
    <Property Key="GloballyAvailable" Value="true" />
    <Property Key="RegisterForms" Value="*.xsn" />
  </Properties>
</Feature>
```

## Creating a workflow.xml File

The workflow definition file `workflow.xml` contains information that SharePoint requires to instantiate and run the workflow. Follow these steps to create a workflow definition file.

1. Open the `Workflow.xml` file in Visual Studio 2005.
2. Right-click in the `Workflow.xml` file and select **Insert Snippet** ► **SharePoint Server Workflow** ► **Workflow.xml Code**.

The `Name` attribute of the `<Workflow>` element contains the name of the workflow. The `Id` attribute must be a different GUID than the one specified in the `feature.xml` file. The `PublicKeyToken` attribute contains the public key token of the workflow assembly. The `<TaskListContentTypeId>` element specifies the content type ID of the content type assigned to the workflow task list (this is optional).

---

**Note** By default, the content type of the workflow task is the Task base content type. The content type ID of the Task base content type is 0x0108.

---

The `<StatusPageUrl>` element contains the URL of a custom workflow status page. If this URL is not specified, Microsoft Office SharePoint Server 2007 uses the default status page that is located at `_layouts/WrkStat.aspx`.

The `<Instantiation_FormURN>` and `<Association_FormURN>` elements contain the IDs of the initiation form. You can get the ID of the initiation form by opening the form in InfoPath in design mode. Select **File** ► **Properties** and copy the value from the ID text box. The `<Task0_FormURN>` element contains the ID of the task edit form. Listing 4-4 shows what the `workflow.xml` file will look like.

**Listing 4-4.** *Content of the workflow.xml File*

```

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Workflow
    Name="MOSSWorkflow"
    Description="Description of the workflow"
    Id="D250636F-0A26-4019-8425-A5232D592C10"
    CodeBesideClass="MOSSWorkflow.Workflow1"
    CodeBesideAssembly="MOSSWorkflow, Version=1.0.0.0, ↵
    Culture=neutral, PublicKeyToken=bebe1ea49fdec076"
    TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"
    AssociationUrl="_layouts/CstWrkflIP.aspx"
    InstantiationUrl="_layouts/IniWrkflIP.aspx"
    ModificationUrl="_layouts/ModWrkflIP.aspx"
    StatusUrl="_layouts/WrkStat.aspx">
    <Categories/>
    <MetaData>
      <Association_FormURN>
        urn:schemas-microsoft-com:office:infopath:InitiationForm: ↵
        -myXSD-2007-05-25T12-13-02
      </Association_FormURN>
      <Instantiation_FormURN>
        urn:schemas-microsoft-com:office:infopath:InitiationForm: ↵
        -myXSD-2007-05-25T12-13-02
      </Instantiation_FormURN>
      <TaskO_FormURN>
        urn:schemas-microsoft-com:office:infopath:EditTaskForm: ↵
        -myXSD-2007-05-25T17-06-24
      </TaskO_FormURN>
    </MetaData>
  </Workflow>
</Elements>

```

## Deploying the Workflow

After creating these two XML files, you have to create a directory in the following location: [drive letter]:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\FEATURES. Specify a unique and recognizable name for this directory. Copy the feature.xml and workflow.xml files and the InfoPath forms in this directory.

The last thing to do to deploy your workflow is install and activate the workflow via the SharePoint Administrators tool stsadmin.exe. The stsadmin tool can be found in the following location: [drive letter]:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN\. The following command will install the workflow.

```
STSADM.EXE -o installfeature -name MOSSWorkflow -force
```

To activate the workflow feature use the following command:

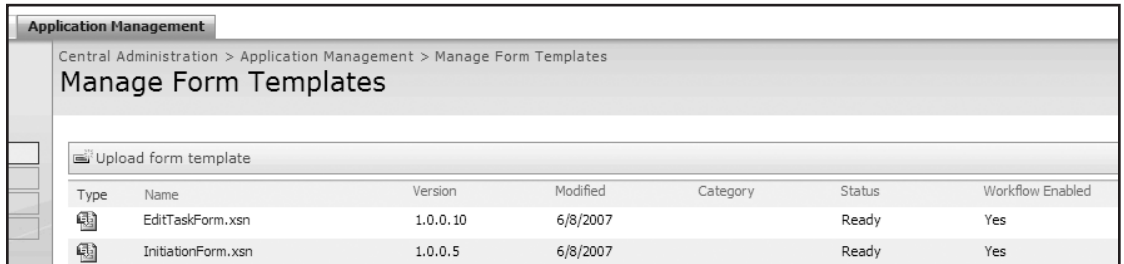
```
STSADM.EXE -o activatefeature -filename MOSSWorkflow\feature.xml -url ↵
http://myserver
```

---

**Note** If this does not work, open a command prompt, type **iisreset**, and try again.

---

Via SharePoint Central Administration, you can check whether the InfoPath forms are correctly uploaded. Open SharePoint Central Administration ► Application Management ► Manage form templates. In the Manage Form Template page, you will find the two InfoPath forms with Status set to Ready and WorkflowEnabled set to Yes, as shown in Figure 4-28.

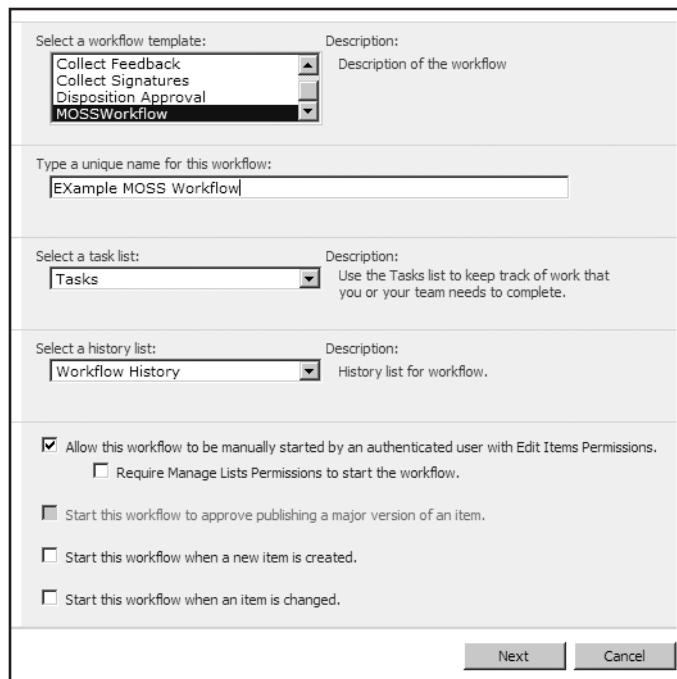


The screenshot shows the 'Application Management' section of SharePoint Central Administration, specifically the 'Manage Form Templates' page. The breadcrumb trail is 'Central Administration > Application Management > Manage Form Templates'. Below the title, there is an 'Upload form template' button. A table lists the uploaded forms:

Type	Name	Version	Modified	Category	Status	Workflow Enabled
	EditTaskForm.xsn	1.0.0.10	6/8/2007		Ready	Yes
	InitiationForm.xsn	1.0.0.5	6/8/2007		Ready	Yes

**Figure 4-28.** InfoPath forms with WorkflowEnabled set to Yes

At this point, you are ready to use the workflow in the SharePoint site. For instance, you can link the workflow to a document library, which is shown in Figure 4-29.



The screenshot shows the 'Add a custom workflow to a document library' dialog box in SharePoint. It contains the following fields and options:

- Select a workflow template:** A dropdown menu with options: Collect Feedback, Collect Signatures, Disposition Approval, and MOSSWorkflow (selected). Description: Description of the workflow.
- Type a unique name for this workflow:** A text input field containing 'EXample MOSS Workflow'.
- Select a task list:** A dropdown menu with 'Tasks' selected. Description: Use the Tasks list to keep track of work that you or your team needs to complete.
- Select a history list:** A dropdown menu with 'Workflow History' selected. Description: History list for workflow.
- Permissions:** A checked checkbox 'Allow this workflow to be manually started by an authenticated user with Edit Items Permissions.' and an unchecked checkbox 'Require Manage Lists Permissions to start the workflow.'
- Start conditions:** Three unchecked checkboxes: 'Start this workflow to approve publishing a major version of an item.', 'Start this workflow when a new item is created.', and 'Start this workflow when an item is changed.'
- Buttons:** 'Next' and 'Cancel' buttons at the bottom right.

**Figure 4-29.** Adding a custom workflow to a document library in SharePoint

---

**Note** If you build the workflow solution in release mode in Visual Studio 2005 Designer, the `postbuildactions.bat` batch file creates and deploys a `.wsp` solution file that can be used for deployment on a production server. The `manifest.xml` and `wsp_structure.dfd` files are filled with the following information from the workflow project: the feature directory name, the name of the feature.xml file, the name of the workflow.xml file, and the name and relative path of the compiled workflow assembly. For more information about the deployment of a `.wsp` solution file, check out the following link: <http://msdn2.microsoft.com/en-us/library/ms460303.aspx>.

---

## Debugging a Workflow

Debugging workflows is not very different from debugging other types of .NET projects. This is not surprising, as the Visual Studio 2005 Workflow Designer is hosted within Visual Studio 2005. In addition to standard debugging capabilities such as the ability to set break points and the option to view call stack windows, it also supports a range of visual indicators that provide information about the debugging process. It is possible to step in, step out, and step over steps in a workflow.

Two types of debugging are not supported in Visual Studio 2005 Workflow Designer:

- Just-in-time debugging of run-time exceptions in the hosting process
- Just-in-time debugging by selecting a process in the Task Manager

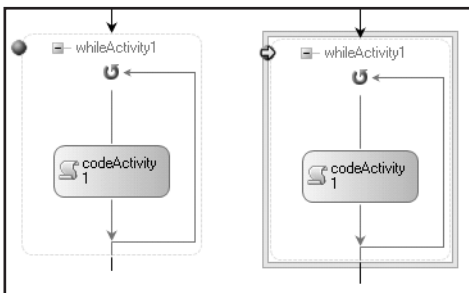
---

**Note** It is only possible to debug workflow applications if the Windows Workflow Foundation Debugger component is installed. This component is part of Visual Studio 2005 extensions for .NET Framework 3.0. You can find more information about Visual Studio 2005 extensions for .NET Framework 3.0 in the section “Getting to Know Visual Studio 2005 Designer.”

---

Follow these steps to attach to a Microsoft Office SharePoint Server 2007 process in Visual Studio 2005 Designer.

1. Open the workflow project in Visual Studio.
2. Add one or more breakpoints to the workflow. This can be done either in code view or in design view in the Workflow Designer. Figure 4-30 shows a breakpoint that is set with the Workflow Designer design view.



**Figure 4-30.** Setting and hitting a breakpoint in the design view of the Workflow Designer

3. Right-click the project and select Properties.
4. Select the Debug tab.
5. In the Start Action section, enter the SharePoint site URL in the Start browser with URL text box.
6. Hit F5.
7. Start your workflow in Microsoft Office SharePoint Server 2007 by selecting the action menu of an item that is stored in the Shared Documents document library. Then, choose the Workflows option and click Example Microsoft Office SharePoint Server 2007 Workflow.

## Using Forms to Capture and Automate Your Processes

It is possible to make a workflow more dynamic and flexible by adding forms to it. This way, you can gather information from users at specific points in the workflow process. One important thing to know about Windows SharePoint Services workflows is that they are form-agnostic. This means that you can create forms with any technology you want as long as the forms are capable of doing the following:

- Invoking the Windows SharePoint Services 3.0 object model
- Generating the data necessary to send to the Windows SharePoint Services object model
- Receiving and parsing the required data from the Windows SharePoint Services object model

Information passed to the form is formatted as a string; this is also true for data that the form must pass back to the Windows SharePoint Services object model when the user submits the form. This string characteristically contains XML, although any data format can be used, as long as the data is a string. A common example of a form type used within SharePoint workflows is custom forms created using ASPX that use XML as the data format of choice.

There are three types of forms available in a workflow:

- Association and initiation forms
- Modification forms
- Custom task forms

Association and initiation forms are forms that are displayed before the workflow starts. This way the user can set parameters and other information for the workflow in advance.

Association forms are used to take care of how the workflow applies to a specific list, library, or content type. These forms are displayed to the person who decides to associate a workflow to a list, library, or content type. When a workflow is associated with a document library, the Add new workflow page will be displayed first. On the Add new workflow page, you can set specific settings that are common to every workflow, such as the type of workflow and which Tasks list you want to use. The next page is the association page. This page is specifically designed for this specific workflow. It will collect customized association information to create the association with the list, library, or content type.

Initiation forms are optional forms that deal with a workflow as it is associated with a specific SharePoint item. The initiation form uses the data from the association form to prepopulate initial values. Sometimes the association and initiation forms are the same form. This way you can enable users to overwrite the initial values set by the administrator.

Modification forms enable the user to make modifications while the workflow is running. With a modification form, a user can make modifications at any moment, for example, to assign a task to another user.

Custom task forms are custom forms that are specified for the tasks in a workflow. A custom task form is based on the workflow task type. Each workflow task type in SharePoint has a content type assigned with it. This content type determines whether the custom task form is specified for a certain workflow task type.

## Creating and Using a Custom Activity

To get familiar with every aspect of Windows Workflow Foundation, we are also going to create a simple custom activity. We will create an activity that shows a string in a message box. Follow these steps to create a custom activity:

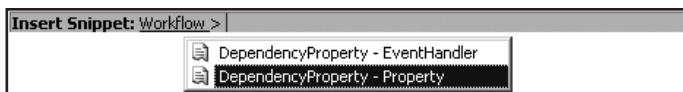
1. Click File ► New ► Project. The New Project dialog window will open.
2. Choose to create a Workflow Activity Library and give it the following name: **TestActivity**. Click OK.

An activity is a class that ultimately inherits from the `System.Workflow.ComponentModel.Activity` base class. You can create an activity based on other existing activities by inheriting from a built-in activity, a custom activity, or an activity created by a third-party vendor. If you want to build an activity that is completely new, you need to inherit from the `Activity` base class directly.

In this section, we are going to create a new activity that is a direct child of the base `Activity` class. By default, the activity class that is created when you create a Workflow Activity Library project inherits from the `SequenceActivity` class. To change this, go to the code view of `Activity1.cs` and change the base class of `Activity1` to `Activity`.

After the Workflow Activity Library project is compiled, the new activity becomes available automatically on the toolbox, so users can drag and drop your activity onto the design view of a workflow. It is useful for an activity to have *dependency properties*, which bind their values to relevant data, including other properties in other activities that use this custom activity. Follow these steps to insert a dependency property.

1. Go to the code view of the `Activity1.cs` class.
2. Right-click in the class and choose Insert Snippet.
3. Choose Workflow and press Tab.
4. Choose `DependencyProperty - Property` and press Tab (see Figure 4-31).



**Figure 4-31.** Inserting the `DependencyProperty - Property` code snippet

The following code fragment shows the result of inserting this code snippet:

```

public static DependencyProperty MyPropertyProperty =
    System.Workflow.ComponentModel.DependencyProperty.Register("MyProperty",
        typeof(string), typeof(Activity1));

[Description("This is the description which appears in the Property Browser")]
[Category("This is the category which will be displayed in the Property Browser")]
[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public string MyProperty
{
    get
    {
        return ((string)(base.GetValue(Activity1.MyPropertyProperty)));
    }
    set
    {
        base.SetValue(Activity1.MyPropertyProperty, value);
    }
}

```

This code creates a static instance of the `DependencyProperty` type and defines a property (this time a member type, not a static type) that uses the `DependencyProperty` static instance as a key for retrieving the run-time value of the `MyProperty` property.

You need to give the static member the following name: `MessageProperty`. The name of the actual property (the member type) will be `Message`. You also need to change the `[Description]` and `[Category]` attributes of the property. The values of these attributes are displayed in the Property Browser. Use the Tab key to tab through the code snippet and customize the code so it looks like the following:

```

public static DependencyProperty MessageProperty =
    System.Workflow.ComponentModel.DependencyProperty.Register("Message",
        typeof(string), typeof(MyTestActivity));

[Description("This is the description which appears in the Property Browser")]
[Category("This is the category which will be displayed in the Property Browser")]
[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public string Message
{
    get
    {
        return ((string)(base.GetValue(MyTestActivity.MessageProperty)));
    }
    set
    {
        base.SetValue(MyTestActivity.MessageProperty, value);
    }
}

```

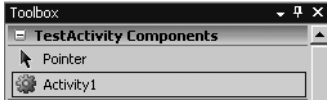
Finally, you need to add an `Execute()` method. This method is a mediator that manages the core tasks of the workflow. The code in this method makes use of the `System.Windows.Forms` namespace.

```

protected override ActivityExecutionStatus Execute
    (ActivityExecutionContext executionContext)
{
    MessageBox.Show(Message);
    return ActivityExecutionStatus.Closed;
}

```

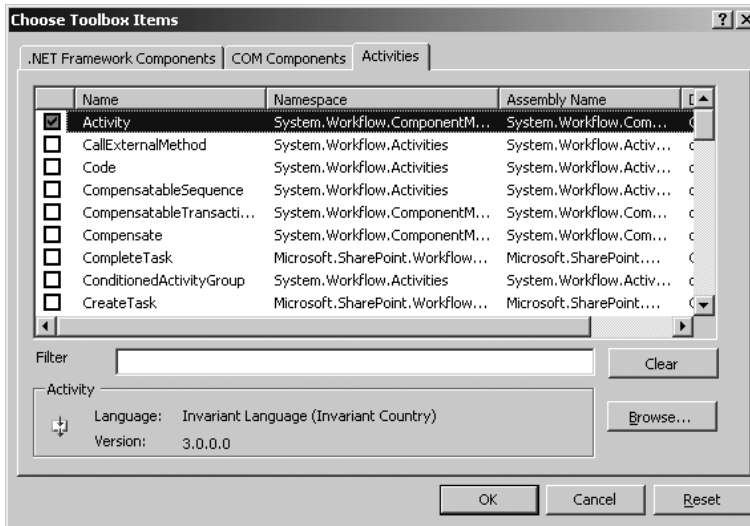
You can find the new custom activity on the Toolbox of the workflow solution used to develop the new activity (see Figure 4-32).



**Figure 4-32.** Custom Activity1 in Toolbox

If you want to reuse this activity in another workflow project in Visual Studio 2005 Designer, follow these steps:

1. Right-click in the Toolbox area and choose Choose Items.
2. In the Choose Toolbox Items dialog window, click the Activities tab and click Browse.
3. Browse to your activity assembly and click Open.
4. Your activity is mentioned in the list of activities and you can add a check to its check box to include it on the Toolbox (see Figure 4-33).



**Figure 4-33.** Choose Toolbox Items dialog window

It is not possible to create custom activities with SharePoint Designer. However, you can use custom activities created using Visual Studio 2005 in SharePoint Designer. To use a custom activity, you must make sure that the activity is added to the safe controls list in Microsoft Office SharePoint Server 2007. This way, all activities must be preapproved by a server administrator before using them. It is not possible to deploy a custom activity without the approval of a server administrator.

If you want to use a custom activity assembly in SharePoint Designer, you have to place the assembly inside the GAC, which means that the workflow assembly must be strong named. Follow the next procedure to sign your assembly:



1. Right-click the activity project in the Solution Explorer and select Properties.
2. Select the Signing tab on the left.
3. Check the Sign the assembly check box.
4. Select <New> from the Choose a strong name key file drop-down list. This opens the Create Strong Name Key dialog box.
5. Specify a name in the Key file name text box and uncheck the Protect my key file with a password check box.
6. Click OK.
7. Rebuild the activity project.

The next step is to add the activity to the action safe list in the SharePoint web application's web.config file. The web.config file can be found via the following procedure:

1. Open Internet Information Services (IIS) Manager.
2. Expand the [server name] (local computer) node.
3. Expand the Web Sites node.
4. Right-click the SharePoint web application and choose Properties. This opens the [web application] Properties window.
5. Click the Home Directory tab and copy the value of the Local path text field.
6. Open an instance of Windows Explorer and navigate to the path found in the previous step.
7. Open the web.config file in any text editor.

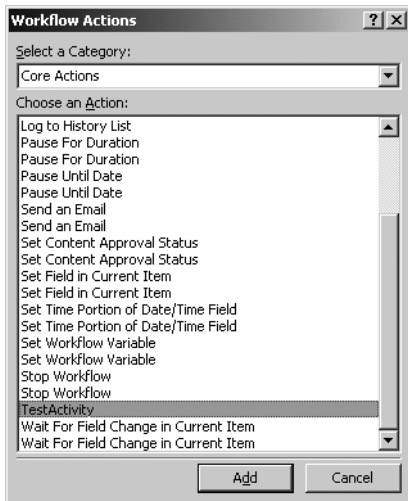
Locate the <SafeControls> section and add a <SafeControl> element. This element should look like this:

```
<SafeControl Assembly="[assembly name],  
Version=1.0.0.0, Culture=neutral,  
PublicKeyToken=[public key token]"  
Namespace="[namespace name]" TypeName="*" Safe="True" />
```

The last thing to do is add an <Action> element, rules, and parameters to the WSS.actions file. You can find the WSS.actions file in the following location: [drive letter]:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\1033\Workflow. The Action element gives SharePoint Designer all the information it needs to let the action appear and function correctly. The following code fragment shows the <Action> element that must be added to the WSS.actions file:

```
<Action Name="TestActivity"  
  ClassName="[class name]"  
  Assembly="[assembly name], Version=1.0.0.0, Culture=neutral,  
  PublicKeyToken=[public key token]"  
  Category="Core Actions"  
  AppliesTo="all">  
</Action>
```

After adding a <SafeControl> element to the SharePoint web application's web.config file and an <Action> element to the WSS.actions file, the custom activity is added to the Actions list in the Workflow Designer. Figure 4-34 shows that we've added the custom activity TestActivity to the Workflow Designer in SharePoint Designer.



**Figure 4-34.** *The custom activity TestActivity in the Workflow Actions dialog window*

## Summary

In this chapter, we discussed Windows Workflow Foundation. We have taken a look at the basics of Windows Workflow Foundation. Next, we discussed the two development tools available to create workflows: SharePoint Designer and Visual Studio 2005 Designer. We showed you how to create a workflow with SharePoint Designer and how to create, deploy, and debug a workflow with Visual Studio 2005 Designer. Finally, we discussed the different forms that you can use to capture and automate the workflow process.