Workflow, Tomorrow's Application Logic

Date: Sep 16, 2005 By Scott Woodgate. Sample Chapter is provided courtesy of Sams.

At the PDC last week, Microsoft announced Windows Workflow Foundation as a key component of WinFX in conjunction with Windows Communication Foundation and Windows Presentation Foundation. This chapter describes the motivation behind Windows Workflow Foundation and provides an overview of its features.

Chapter 1: Workflow, Tomorrow's Application Logic

In this chapter

- The Wonder of Flowcharts
- Today's Workflow Scenarios
- Introducing Windows Workflow Foundation
- Windows Workflow Foundation Engine Architecture

At the professional developer conference in September 2005, Microsoft announced Windows Workflow Foundation as a key component of WinFX in conjunction with Windows Communication Foundation and Windows Presentation Foundation. This chapter describes the motivation behind Windows Workflow Foundation and provides an overview of its features.

NOTE

Windows Workflow Foundation is a beta technology and this book was written before the beta was released. It is very possible that changes to the technology will be made through the beta cycles, which will reflect on the accuracy of information presented in this book.

The Wonder of Flowcharts

Inside classrooms around the planet students often receive their first exposure to computer program design through the concept of a flowchart. The *flowchart* provides a graphical model that is used to formalize the design of program structure—with squares, diamonds, and triangles representing the various steps in the flow such as activities and decisions. Although the flowchart model is an excellent learning tool, the flowchart is not directly represented in running software. The expense of maintaining a flowchart model in software from a CPU-cycles perspective means that the flowchart is nothing more than documentation. After the concepts of flowchart program design are mastered by students, the flowchart model is forgotten and programs are written directly in code.

Although writing programs directly in code has been the main development paradigm for more than 25 years, and many millions of programs have been created, a pure coding approach is not without issues.

- Code written by one developer is often inherently hard to understand by another. Although well-documented or commented code helps, it is often unstructured in approach. Indeed the intellectual property that is the flow of the program often stays in the mind of the original developer and is at risk when that developer moves to a new job or a new role. The flowchart model, if instantiated in code, would certainly provide an additional level of visibility in program design.

- Once compiled and executed, code is inherently opaque. The resulting assembly language or intermediary language running on a CPU natively or through a virtual machine executes a predefined set of op-codes to perform a set of activities but does not provide visibility into how these activities are executed. This has some advantages as it protects the intellectual property of the code itself, but it also creates challenges. As programs become larger, visibility into these programs at runtime becomes more important to troubleshoot errors in context of program execution. The flowchart model, if available at runtime, could provide visibility into the program execution.

- Code is often grouped into procedures or objects. Encapsulation of code, and indeed data, in these formalisms makes program design more componentized and easier to maintain. On the other hand, it poses some interesting challenges. How does data in one procedure or object move to the next procedure or object? There are fundamentally two approaches. The first is to send a message from one object to another such as a method call from one object to another passing state, or an XML message serializing state. Although this design is functional in many cases, it isn't elegant as a state needs to be passed through a chain of many objects all through parameters or XML messages. The second approach centralized the shared state in an object. The state can be retrieved, updated, and persisted to the object when required. Examples of this method include web pages on a site that communicate through shared state held in databases that may be session based or otherwise. However, in each case these state management features were custom-crafted because of the lack of a consistent development framework to manage state. Some of the challenges such a framework would face include scalability challenges in managing state over long time periods. A simple flowchart model has the notion of shared state. A variable is often set in the first step of a flowchart and then referred to later in the thread of execution. If this model for accessing state were more accessible to the programmer in a scalable manner then state management could be more efficient.

Although not part of mainstream development, over the past five years instantiating a flowchart as part of a program runtime has proved so useful in certain scenarios that regardless of the CPU cost it has become the preferred method of interaction. Some of these scenarios are described in the following sections.

Today's Workflow Scenarios

Key scenarios that use the flowchart model, branded as workflow, in software include the following:

- *Document lifecycle management*—Historically, managing the content within documents across multiple users has been ad-hoc and other than versioning tracking within documents themselves the collaboration process required to produce a particular document has not been recorded. More recently, driven either by compliance to regulation, such as Sarbanes-Oxley, or the need to accelerate traditionally paper-based processes, more and more businesses are looking to document lifecycle management. Document lifecycle management includes key workflow aspects such as tracking the people involved, and often enforcing a particular interaction pattern among multiple participants across a piece of content. The most common document lifecycle management scenarios include expense report and absence report submission where a document is submitted to a workflow involving roles, such as the manager role, which approve the content before it flows to another system.

- *Internal application workflow* —Several ERP, CRM, and other LOB vendors implement workflow within their applications to describe specific business processes that are executed within the applications. An example of this is the inventory process inside PeopleSoft. Vendors typically embed workflow to enable their end-users, sometimes developers and sometimes business users, to customize the LOB application. One of the vendor challenges associated with intra-application workflow is that each LOB vendor has to create its own workflow engine and technology rather than using a widely available common technology and as a result customers end up interacting with multiple workflow concepts, styles, and designers across multiple vendor products. Another challenge for customers is that at a business level

a workflow may transverse multiple systems, which leads to the requirement to create a workflow that is external to all LOB systems and interacts with them.

- *Business process management (BPM)* —Business process management is the category of products that provide externalized business process and workflow to a set of pre-existing applications. BPM's goal is to drive control back to the business and enable agility in terms of how software responds to changing business requirements and processes. Most often business process management is layered on top of traditional integration technologies that connect prepackaged applications such as SAP and Siebel as well as businesses to other businesses across the Internet to manage the asynchronous stateful aspects of integration.

- *Page flow* —Traversing a user interface often involves navigating across a set of linked pages or views with a subset of the variables shared between the views. Commonly this navigation functionality, or page flow, and the state it requires are intermingled into the user interface layer. By intermingling the navigation functionality with the user interface, there is limited reuse of the navigation functionality and multiple files need to be modified when a navigation change is made. Several vendors have recognized this problem and applied the model-view-controller approach to user interface design separating out the page flow element as a distinct controller element from the user interface. This has been most typically applied to websites with technologies such as the Java struts framework and BEA WebLogic PageFlow, but it is equally applicable in principle to rich user interfaces such as Windows Forms, as demonstrated by the Universal Interface Process block from Microsoft Prescriptive Architecture Group for both WinForms and websites.

All of these scenarios are similar in that by using workflow they achieve greater flexibility for the designer, greater visibility into the running program, and the ability to manage state across multiple steps. Although workflow has been and will continue to be important in these scenarios, workflow has not been generally accepted into program design. Some of the historical challenges with using workflow, such as the expense of an additional layer of abstract in terms of CPU cost, are no longer relevant. Other challenges such as enabling a running workflow to be modified in-flight to provide greater customization have not been well solved in workflow software until today. What if this workflow technology was broadly available for use in all programs?

Introducing Windows Workflow Foundation

Windows Workflow Foundation is a core component of the next generation of the .NET Framework, called WinFX. Windows Workflow Foundation is broadly available to all users of Windows running Windows XP, Server 2003, Windows Vista, and the next generation Windows Server release. Some of the key goals of Windows Workflow Foundation technology are to

- Provide a singular engine for workflow execution for all applications built on the Windows platform.

- Support a wide range of scenarios from a highly people-centric workflow to the highly structured application-centric workflow and a variety of blended rules based on conditional scenarios.

- Bring model-driven workflow development to the entire WinFX development community such that every developer who today is familiar with the .NET Framework application can be immediately productive, building workflow-enabled applications without learning a second parallel set of technologies.

- Enable reuseable workflow component development through strong extensibility points and ensure both developers and ISVs can deeply embed this technology in their applications.

This is a very bold set of goals and Microsoft is achieving these goals by focusing on the core workflow requirements across many scenarios. Windows Workflow Foundation is primarily an engine and a framework—it is not a fully featured product

built for a specific scenario. Indeed, although Windows Workflow Foundation is an incredibly important part of all the scenarios described earlier in this chapter, it is insufficient to complete each of these scenarios. For example, in BPM, workflow is supplemented by many features including publish and subscribe messaging, business activity monitoring, and specific development and management tools. Similarly, document lifecycle management requires information worker–centric user interfaces to manipulate the documents and collaboration centers to share them. Nevertheless, developers and ISV can use Windows Workflow Foundation and build all these scenarios around the technology.

Windows Workflow Foundation takes a unique approach to workflow, which is highly extensible. It does not have an inherent language and instead it executes a set of Microsoft and user-created steps. By taking this engine- and framework-based approach, Windows Workflow Foundation is able to address a broad range of scenarios rather than becoming restricted to a singular niche scenario and provide substantial opportunity for partners and ISVs to build a workflow ecosystem around the technology. The next section details the engine architecture for Windows Workflow Foundation.

Windows Workflow Foundation Engine Architecture

Given the breadth of scenarios requiring workflow and the key goal of providing a singular technology layer to support all these scenarios, the workflow technology is a well-factored framework with numerous pluggable interfaces and extensibility as a core design principle. The architecture for Windows Workflow Foundation is depicted in Figure 1.1.

Figure 1.1 Windows Workflow Foundation engine architecture.

At the bottom of Figure 1.1 is the *host process*. Windows Workflow Foundation has no inherent execution process. Instead, Windows Workflow Foundation is an in-process engine that runs inside a host process. The host process is responsible for providing a set of services to Windows Workflow Foundation. A wide variety of host processes are available on the Windows platform including console applications, WinForms applications, web applications, web services applications, SharePoint Server, and NT Service applications. Effectively, any executable process can host the Windows Workflow Foundation runtime engine.

This also presents some interesting challenges because the capabilities of each host are often different to another host. SharePoint is a dramatically different environment than a console application. For this reason the Windows Workflow Foundation architecture *hosting layer* provides a set of pluggable interfaces from Windows Workflow Foundation to the host.

Sitting on top of the hosting layer in the Windows Workflow Foundation architecture is the *runtime layer*. The runtime layer is the core of the engine providing both workflow execution and workflow lifecycle management capabilities.

Finally, the workflow model layer is where most developers will interact with Windows Workflow Foundation. The workflow model layer includes the various workflow models, APIs, and the activities. The following sections provide more details on the hosting, runtime, and workflow model layers.

Hosting Layer

The hosting layer provides interfaces between Windows Workflow Foundation and a particular host for the following key services: Communication, Persistence, Tracking, Timer, Threading, and Transaction. The implementations of the former three

services that ship with Windows Workflow Foundation are durable while the latter two services are stateless. However, none of the services are necessarily durable if you write your own. By abstracting each of these services Windows Workflow Foundation can take advantage of specific capabilities available in specific hosts. The following sections describe the functions performed by each of these services:

- *Persistence*: Although some workflows may execute for a short period of time, workflow is inherently asynchronous and a particular workflow, such as the Ph.D. thesis approval process, may take many days or months. A workflow engine that retained its state in memory for that period would not scale as each instance of the workflow would consume memory for the duration and eventually the system memory would be exhausted. Instead, a persistent architecture is used where workflow is executed in memory, and should it be required, the workflow state will persist to a store while it waits for a response that might take some time such as the "Ph.D. approved" step in the workflow. Each host application has a specific set of persistence requirements. For example, to persist state, ASP.NET uses a set of Session State objects that has state client providers for in-memory persistence and SQL Server persistence. In contrast, SharePoint persists state in a SharePoint-specific set of tables in SQL Server and your console application may choose to persist state to the file system as XML. With such a large variety of host-specific persistence capabilities, it would not be sensible for a broadly applicable technology such as Windows Workflow Foundation to specify a single persistence provider. The Windows Workflow Foundation hosting layer persistence interface enables Windows Workflow Foundation to work across the full gamut of host persistence architectures.

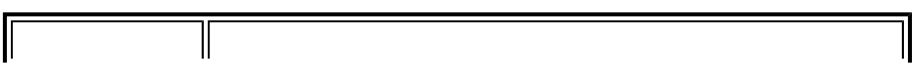
- *Timer*: Workflows often need to wait for an event to continue. The timer is the supplied clock that is used to manage these delays. For example, an approval workflow may delay and unload from memory until a particular approver completes the necessary approval. The timer implementation in this case might be a durable timer that survives a potential system restart while waiting for approval.

- *Tracking*: A key reason to implement workflow is because the workflow model provides a greater degree of system transparency at runtime than amorphous code. Indeed, all workflows are automatically instrumented without any programming. The tracking instrumentation is consistent across both the event content and the tracking interface. Depending on the host, the target tracking infrastructure is often different. For example, a LOB application often persists workflow tracking information within the LOB database whereas a console application may persist tracking information to an XML file. The tracking interface receives tracking events from the Windows Workflow Foundation runtime and passes them on to the host application.

- *Communications*: Workflows send and receive events or messages from their host. These events trigger workflows, and move the workflow to the next step in the overall flow. There are a wide variety of communications infrastructures available on the Windows platform including web services, .NET calls, loosely coupled messaging, and so on. For this reason, Windows Workflow Foundation does not provide its own proprietary communications layer. Instead, Windows Workflow Foundation provides pluggable interfaces that can be implemented for any communications layer. Of course, there are easy-to-use, prebuilt communication interfaces to and from common targets such as web services and for passing data objects in and out of a workflow—perhaps from a form.

The physical job of development of these interfaces for specific hosts is relatively challenging compared to other aspects of workflow development described shortly. For this reason, ISVs will typically build host layer providers into their host applications so that end-user developers can simply reuse these services. In addition, Windows Workflow Foundation ships prebuilt support for ASP.NET 2.0 and the interfaces shown in Table 1.1.

Table 1.1 Prebuilt Host Layer Service Implementations



Host Layer	Service Implementation
Persistence	SQL Server state persistence
Timer	Both an in-memory and SQL Server–based timer
Threading	.NET thread pool ASP.NET thread pool
Tracking SQL	Server Tracking Persistence and Event Log recording for termination
Communications	.NET components and web services

Sitting on top of the hosting layer is the runtime layer.

Runtime Layer

The runtime layer is core to Windows Workflow Foundation. In direct comparison to the other layers in the architecture, the runtime layer is not pluggable as it contains the mission-critical services required for workflow. These services include the following:

- *Execution*: The execution service schedules activities and supports common behaviors such as event handling, exceptions, tracking, and transactions.
- *Tracking*: The tracking service creates the tracking events that are serialized through the tracking interface.
- *State management*: The state management service manages states that may be persisted through the persistence interface.
- *Scheduler*: The scheduler service schedules execution of activities.
- *Rules*: The rules service provides policy execution functionality and CodeDOM condition evaluation.

Workflow Model Layer

The workflow model layer is where most application developers will spend the majority of their time writing code for Windows Workflow Foundation. This layer includes support for multiple workflow model types, activities, and the main programming APIs use by most developers.

Windows Workflow Foundation supports two models out of the box:

- *Sequential workflow model*: This model is primarily a structured workflow where a step in the workflow leads to another step in a manner that may be determined often at design-time and can be represented in a flow diagram such as that shown in Figure 1.2.

  Sequential workflows are most often used to represent structured workflows such as system-to-system workflow. These transformational workflows are self-driven once they are initiated, have a highly predictable path through the events, and are often literally sequential in nature.

- *State machine workflow model*: This model uses the paradigm of states and transitions between states to represent the workflow. There is no deterministic path between the steps from a design perspective because the workflow does not execute in a sequential nature. Rather, the workflow is a set of events that are handled in a highly variable order with one event completing and triggering a state that another event may itself trigger from. The state machine model can literally jump from any stage in the workflow to any other stages, and often will do so multiple times before reaching a completed state. The order workflow is a state machine where various messages are received and trigger the workflow to progress to a particular state. Each iteration of this workflow may result in a different path through

the model as depicted in Figure 1.3.

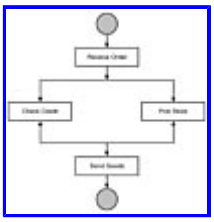


Figure 1.2 Sequential workflow.

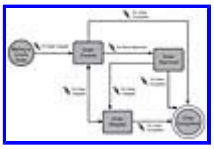


Figure 1.3 State machine workflow model.

State Machine Workflows are an effective way of representing highly people-centric workflows where the workflow thread of execution is not easily represented in a flow. This workflow model is also very useful for scenarios where a high priority event must be processed even though work is already in process or when a large number of events may occur at a particular stage in the workflow. A perfect example is a stage in the order workflow where an "order cancelled," "order updated," and "order completed" event that may be received at any time and should immediately cancel the entire process.

Although Windows Workflow includes these two models, customers can inherit from them to create their own specific models or create new models.

Regardless of model and sequencing behavior the basic element of execution and reuse is called an *activity*. An example of an activity is "send goods" in the previous sequential workflow example.

There are two types of activities—the simple activity and the composite activities. What makes Windows Workflow Foundation very different from traditional workflow engines is that the engine has no fixed underlying language or grammar. Instead the engine chains a set of activities that are supplied by Microsoft and created by you the developer. Microsoft-supplied constructs include "If," "Code" blocks, activities for web services, and many more. In addition, you can create your own control flow activities such as "do until" but more likely you will be creating higher level activities such as the "Receive Order" activity described previously. By using an activity execution methodology rather than a language, Windows Workflow Foundation is able to support a broad range of scenarios and you can reuse your activities across multiple workflows. Indeed the flow and the state workflow models share a majority of activities with some specific inclusions and exclusions to each model. Activities become the unit of encapsulation in much the same way that ActiveX controls were for Visual Basic 6 applications. It is expected that customers and partners will share activities in the community and generate business from activity creation.

The set of flow control activities that ship with Windows Workflow Foundation include the following:

- Control flow activities: `Sequence`, `Parallel`, `While`, `IfElse`, `Listen`, `EventDriven`, `ConditionedActivityGroup`, `Replicator`, `Delay`
- Transaction and exception activities: `ExceptionHandler`, `Throw`, `Compensate`, `Suspend`, and `Terminate`
- Data/form-centric activities: `UpdateData`, `SelectData`, `WaitForData`, `WaitForQuery`
- Communication activities: `InvokeWebService`, `WebServiceReceive`,

`WebServiceResponse`, `InvokeMethod`, and `EventSink`

- The code activity: `Code`

There are three additional activities that are specific for the state machine workflow model: `StateInitalization`, `State`, and `SetState`.

Each activity is derived from the `Activity` base class and includes the code that will execute when the activity and a set of design-time properties for use in the designer is called. Later chapters in this book will go into detail on each of these activities; however, a few activities are worth mentioning here. The data- and form-centric activities enable you to bind data from forms and easily surface that information into a workflow; the web services activities give you the capability to consume and expose web services; more advanced activities such as the conditioned activity group enable policy- or rules-based condition evaluation.

Over time, many activities will become available through the broader activity ecosystem through blogs and shared source as well as through Microsoft partners.

Now that we have addressed activities in detail it is important to point out that the model itself is nothing more than a "root" level activity in the Windows Workflow Foundation infrastructure.

One of the significant features of Windows Workflow Foundation is that it offers you the ability to dynamically create workflow at runtime and dynamically update an existing workflow. This is discussed in more detail later in this book.

Now that you have completed your tour of the Windows Workflow Foundation architecture, it’s time to examine the coding and design-time support for you to interact with the engine.

Design-time

Typically, workflow technologies have provided graphical designers that give users the capability to drag and drop shapes to build a workflow. Some tools are targeted at business users while other tools are targeted at developers. Windows Workflow Foundation provides a set of graphical tools inside Visual Studio .NET targeted at developers. The goal of the Visual Studio .NET design experience is to be as familiar as possible to existing .NET developers writing C# and VB.NET applications and building Longhorn applications using Windows Presentation Foundation and Windows Communication Foundation.

After Windows Workflow Foundation is installed, an additional category appears on the Visual Studio .NET File New dialog box called "Workflow," as shown in Figure 1.4.

Most interesting in this introductory chapter are the Sequential Workflow Library and State Machine Workflow Library templates. The Workflow Activity Library is a template for creating custom activities and creates a project with an activity-specific designer, making it easier for you to create activities. The workflow console application and state machine console application are simple host applications for Workflow, and finally, the empty workflow project is an unconfigured workflow project.

When the Sequential Workflow template is chosen, a new project is created and `System.Workflow`, `System.WorkflowActivities`, and `System.WorkflowComponentModel` are all added to the project references. In addition workflow1.cs and workflow1.designer.cs are added to the project. By default, selecting workflow1.designer.cs with the mouse will graphically render the workflow directly inside the Visual Studio designer as shown in Figure 1.5.

Figure 1.4 File New Visual Basic workflow project.

Figure 1.5 Visual Studio .NET designer for sequential workflow.

The workflow is described by dragging and dropping shapes from the toolbox. Just like traditional WinForms or ASP.NET development, when a shape has code associated with it, clicking on the shape will open the code-behind where custom code may be entered directly in VB or C#. Any changes to the code-behind file that are relevant to display in the workflow are automatically reflected back in the designer.

In addition you can add a new item to the project that is a workflow with xoml. The xoml file is actually an XML file containing an XML definition of the workflow. That XML is fully accessible to the developer by choosing File, Open With and selecting the XML editor inside Visual Studio .NET. This XML and Visual representation is consistent with the Windows Presentation Framework approach in WinFX as shown in Figure 1.5.

The state machine workflow model is depicted in Figure 1.6.

Figure 1.6 Visual Studio .NET designer for state machine workflow.

This simple state machine workflow looks similar to a sequential workflow. However, in this case each box represents a state and in most examples the lines in the diagram will not just go in a forward direction as depicted but also backwards—for example an event from the `orderprocessedstate` may send the state machine back to the `orderopenstate`. Also notice the state machine–specific activities, such as `StateInitialization`, in the activity toolbox.

Visual Studio .NET provides "smart tags" to alert the user that properties on specific activities are not properly configured. When the workflow is complete the developer builds the project and the two partial classes that represent the code-beside and the workflow model are compiled and stored in a managed assembly.

Windows Workflow Foundation provides complete support for the familiar F5 debugging experience. Breakpoints can be set on the models at design-time using the mouse or F9 key and when a break-point is triggered, a complete call-stack is available and the debugger allows stepping into and out of the code for the workflow in debug mode.

There is no business user design surface primarily because these surfaces tend to be scenario-specific and Windows Workflow Foundation needs to support a variety of scenarios. The good news is that you can build your own business user design surface on top of Windows Workflow Foundation. The workflow designer can be completely rehosted outside of Visual Studio .NET in your custom application. Further, every aspect of the designer can be reskinned so that the look and feel of

the designer control, including the size, color, and geometry of the shapes, matches your application style. By rehosting the designer and providing your customers with specialized activity packages, you can give your end-users the ability to create and modify workflow. If rehosting the designer does not provide enough flexibility then ISVs and developers can create the XML representation of the workflow from a custom tool directly, or better still, build an activity tree in code.

A sample XML representation of a workflow is depicted here:

```
<?Mapping XmlNamespace="Activities" ClrNamespace="System.Workflow.Activities"
 Assembly="System.Workflow.Activities" ?>
<SequentialWorkflow x:Class="MyWorkflow" xmlns="Activities" xmlns:x="Definition">
 ...
</SequentialWorkflow>
```

There is another option for generating workflow that is a significant innovation in Windows Workflow Foundation. Unlike many workflow technologies, Windows Workflow Foundation supports a complete coding experience enabling you to create and modify workflows in code. You can choose to use a tool as simple as Notepad and the command-line compiler `wfc.exe` to author workflow. Workflow is simply a class. A sample sequential workflow that executes a single activity called `SendEmail` is shown here:

```
Public Class Workflow1
  Inherits SequentialWorkflow

  Public Sub New()
    MyBase.New()
    InitializeComponent()
  End Sub

  Private Sub InitializeComponent()
    Me.s = New SendEmail
    Me.s.MailFrom = "Paul"
    Me.s.MailTo = "James"
    AddHandler Me.s.OnBeforeSend, AddressOf Me.OnBeforeSend

    Me.Activities.Add(Me.s)
    Me.DynamicUpdateCondition = Nothing
    Me.ID = "Workflow1"

  End Sub

  Private WithEvents s As SendEmail

  Private Sub OnBeforeSend(ByVal sender As System.Object, ByVal e As _
     System.EventArgs)
    Me.s.MailSubject = "Hello at " & System.DateTime.Now.ToString()
  End Sub

End Class
```

There is a designer for activities inside Visual Studio .NET. However, just like workflows, activities are also simply classes derived from the `Activity` base class and can be completely created in code. The `SendEmail` activity skeleton fragment is shown here:

```
<ToolboxItem (GetType(ActivityToolboxItem))> _
Partial Public Class SendEmail
  Inherits System.Workflow.ComponentModel.Activity
  Public Sub New()
    MyBase.New()
    InitializeComponent()
  End Sub

  Public Shared MailToProperty As DependencyProperty = _
   DependencyProperty.Register("MailTo", GetType(System.String), _
   GetType(SendEmail))
```

```
  Public Shared MailFromProperty As DependencyProperty = _
   DependencyProperty.Register("MailFrom", GetType(System.String), _
   GetType(SendEmail))
  Public Shared MailSubjectProperty As DependencyProperty = _
   DependencyProperty.Register("MailSubject", GetType(System.String), _
   GetType(SendEmail))

  <DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)> _
    <ValidationVisibility(ValidationVisibility.Optional)> _
      <Browsable(True)> _
        Public Property MailTo() As System.String
    Get
      Return CType(MyBase.GetValue(SendEmail.MailToProperty), String)
    End Get
    Set(ByVal value As System.String)
      MyBase.SetValue(SendEmail.MailToProperty, value)
    End Set
  End Property

  <DesignerSerializationVisibility (DesignerSerializationVisibility.Visible)> _
    <ValidationVisibility (ValidationVisibility.Optional)> _
      <Browsable (True)> _
        Public Property MailFrom() As System.String
    Get
      Return CType(MyBase.GetValue(SendEmail.MailFromProperty), String)

    End Get
    Set(ByVal value As System.String)
      MyBase.SetValue(SendEmail.MailFromProperty, value)

    End Set
  End Property

  <DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)> _
    <ValidationVisibility(ValidationVisibility.Optional)> _
      <Browsable(True)> _
        Public Property MailSubject() As System.String
    Get
      Return CType(MyBase.GetValue(SendEmail.MailSubjectProperty), String)

    End Get
    Set(ByVal value As System.String)
      MyBase.SetValue(SendEmail.MailSubjectProperty, value)

    End Set
  End Property

  Public Event OnBeforeSend As eventhandler
End Class
```

In addition to design-time API workflow generation, dynamic update makes it possible to update running workflows on the fly. The dynamic update functionality opens the door to many interesting scenarios that were not previously possible with traditional workflow engines such as examining the state of the model at runtime and making behavioral changes as a result of variables within the flow.

This section has just touched the surface of activities; you will learn much more about activities later in this book.

In summary, Windows Workflow Foundation provides a complete authoring experience inside Visual Studio .NET with custom designers and debugging support but also provides the flexibility for you to create workflow using code directly or through XML. Now take a quick look at how a sample application uses Windows Workflow Foundation.

Office 12 Workflow

Office is the most popular desktop productivity program on the planet. One of the

key feature requests from Office customers is the capability to collaborate on documents using workflow. Office 12 uses the Windows Workflow Foundation engine embedded inside the SharePoint host for workflow. The office client applications, such as Word, can kick off workflow through web services integration with Office. That workflow is executed on SharePoint and as a result a document may be sent through email to a user in Outlook to perform an action. Once the action is performed the workflow continues.

Office also provides a simple design experience in FrontPage and for more complex design, Office has its own specific activity package containing more than 30 custom activities and customers can use this directly in Visual Studio .NET. In addition, the InfoPath designer in Office 12 supports data binding form elements to a workflow. Several prebuilt workflows ship with Office 12 including Review, Approval, and Document Expiration and Office 12 supports custom workflows. Figure 1.7 shows the Review workflow in Microsoft Word.

Figure 1.7 Microsoft Word showing a Review workflow.

Developers looking to create people-based workflows should first look into customizing Windows Workflow Foundation in the context of the SharePoint host as this strategy will get you access to the Office applications immediately. There is no doubt workflow is a key feature of Office 12 and the Windows Workflow Foundation technology will be used immediately by many information workers around the world.

You have read about the Windows Workflow architecture, design tools, and a brief description of the Office 12 functionality; however, you might be wondering when and why you should use Workflow. Now that you have briefly learned about the capabilities that Windows Workflow Foundation provides, the final section of this chapter addresses the core tenets of workflow. You can think of these as the call to action because these four tenets address key aspects of application design that are fulfilled by workflow.

Call to Action: Core Workflow Tenets

Before we get to the specific tenets, it is appropriate to revise the tenets for service-oriented architecture because often workflow will be coupled in application architecture with web services. As part of the next generation web services effort, several key tenets have been identified. Although these are described in detail elsewhere on the web they have been summarized as follows:

- *Boundaries are explicit*: When compared to objects or remote objects, services have clear boundaries. Calling services is relatively expensive compared to calling code within a service. Choosing where these boundaries exist is an interesting challenge that leads to the second tenet.

- *Services are autonomous*: The key point here is that services, unlike objects, can execute autonomously from other services. The key word here is *can* because of course in many cases one service will call another.

- *Share schema and contract and not class*: In short, the notion here is that loose coupling is good and sharing a schema between services enables a greater degree of loose coupling than sharing explicit class types.

- *Service compatibility is based on policy*: This tenet emphasizes the separation of semantic and structural concerns. Structural aspects of the services are stored in policy files and service compatibility is driven from these, in some cases without modifying the specific business logic of the service.

These tenets focus on boundaries between components, and the point-to-point connections between these components needed to build composite applications. These four tenets in themselves are insufficient for a service-oriented architecture where distributed services are composed into composite applications. What is missing? The key missing pieces are guidance on how to compose these services together, the characteristics of such a system, and the need to involve people as a part of composite application design. To that end, the four core tenets that follow assume the initial set of tenets and expand on them to provide guidance for composite application development:

- *Workflows are long-running and stateful*: Fundamentally, systems are created by the composition of multiple services and, following good design practices, components that perform specific roles should be created as autonomous services. The behavior between these services can be as simple as passing and mapping behaviors or more complex such as sharing transactional semantics and temporal constraints. The temporal aspects of service composition enforce asynchronous requirements on the system. A service that submits a purchase order to other services as part of a composite application waits for two hours for an acknowledgement. This asynchronous controller, or model, should be independently factored as a specialized service component. Hand in hand with the requirement to model asynchrony is the need to include state as part of the model. The service that sent out a purchase order and received an invoice requires information on the purchase order to later update another service. State may be relevant to a single call, relevant across multiple calls, or relevant across an entire asynchronous workflow.

- *Workflows are transparent and dynamic through their lifecycle*: Services and the applications to which they compose should not be, as they are today, like concrete bunkers with no windows. Although policy provides rigor in the definition of how to talk to the service, what the service does—its behavior—is opaque beyond its method call syntax. System behavior should be transparent, enabling the developer to rapidly ascertain the behavior at design-time and make a change in that behavior. Even more importantly, system behavior should be transparent at the runtime level. If the behavior of each service in a composite application were transparent the advantages would be significant. No longer is the service a concrete bunker; rather, it is more analogous to a greenhouse. Troubleshooting, tracking, and understanding the overall composite application behavior becomes dramatically simpler because of the additional visibility into the thread of execution. Entirely new scenarios can be envisioned where the behavior of the system thus far executed can be queried at runtime and used to influence of the future behavior. With access to the system behavior metadata, the thread of execution can analyze the currently completed behavior and make changes to its behavior on the fly on the basis of new program conditions. Having asserted that system behavior should be transparent by default, there are times when the developer will want to decrease the level of transparency. For example, some service behavior may need to be obstructed for intellectual property reasons, or specific service behavior is visible within a bounded set of services but not beyond that. The dial that sets the service transparency should be set through policy and access control rather than the traditional development approach where a transparent system is not a default and developers add small windows to prebuilt concrete bunkers.

- *Workflows coordinate work performed by people and by software*: The original tenets for services make no reference to people and yet people are a vital part of any composite application. Today services typically send messages to people through email and pagers, or receive inputs from people through user interfaces that pass their parameters to and from services. Sometimes a single person provides the information required for the composite application to continue but more often than not an entirely out-of-band interaction occurs between multiple people in a manner that has been historically challenging to model in software before a result is returned to the composite application to continue. People are important; optimizing their behavior as part of the overall system behavior may provide as high a return as optimizing the system behavior itself with some systems having 80% of their cost in exceptions managed by people. The downside of including people from a software engineering perspective is that modeling their behavior is significantly less straightforward than modeling service behavior.

People tend to work in a more ad hoc manner—it is typically more challenging to be imperative with people, and the flow of information between people may not be easily drawn in a flow-based sense because they make choices that may change the workflow at runtime, but is more ad hoc and is better represented as a set of interacting exception conditions.

- *Workflows are based on extensible models:* Every workflow comprises some number of steps, each of which performs a particular part of a complete business process. The set of actions that are used to construct a workflow can be thought of as comprising a model for that particular problem domain. Different problem domains have different actions, and so a single model isn't appropriate for all workflows. Instead, different groups of actions—different models—can be created for specific domains. Those actions can then be used to construct workflows supporting various business processes in that domain.

With these tenets you can evaluate when to use workflow in your application. There are many, many applications that could benefit from the use of workflow, yet the lack of availability of a workflow model is highly consistent with the typical programming paradigm. With Windows Workflow Foundation for the first time you get to take advantage of this opportunity; moving forward, it will change the way you think about application development.

Summary

Historically, workflow has been limited in its usage within applications due to the lack of availability of generalized technology. For the first time, Windows Workflow Foundation provides application developers and independent system vendors with a generalized workflow engine for building workflow on the Windows platform. Windows Workflow Foundation can be used by developers to support a broad range of scenarios including, but not limited to, composite application development, document lifecycle management, BPM, IT provisioning, and workflow within LOB applications. Windows Workflow Foundation supports multiple workflow models including sequential and state flow machine. Windows Workflow Foundation's designer, available inside Visual Studio .NET and rehostable in your application, provides a productive environment for developers to build workflow in a manner consistent with familiar ASP.NET, Windows Communication Foundation, Window Presentation Foundation, and WinForm development paradigms. The base unit of a workflow in Windows Workflow Foundation is an activity. Activities are fully extensible and it is expected that developers will build a vast range of activities from generic horizontal activities such as "Send Email" to specialized vertical activities such as "Legal approval" and deliver those to customers and end-users through activity packages. You can use the four tenets of workflow to guide you on using workflow as part of your application development process.

The remaining chapters in this book delve into the details of the various aspects of Workflow Foundation with code examples so that you, the developer, can rapidly build skills with this technology and use it routinely when building next-generation applications on top of WinFX.

In addition to this book, a vibrant workflow community is growing so check out blogs and other community sites—or even post your own experiences and samples.