Sams Teach Yourself
Windows Workflow Foundation
in 24 Hours

24 Proven One-Hour Lessons
Introduction

I have spent half of my career focused on business and the other half focused on software development. I am—through my business persona—driven by efficiency. Whereas software has led to tremendous efficiency gains, to say the least, it has also left tremendous room for improvement. Applications are hard to create, understand, and change. My quest for more efficient ways to create software led me to business process management a few years back. After watching the BPM and workflow industry for a couple of years, I was very excited when I learned of Windows Workflow Foundation (WF). After learning more about WF, I became more excited and decided to write this book.

I am convinced that we are entering a new phase of software development and that—on the Microsoft platform—WF lies at its core. Let’s look at the key benefits WF will drive:

- Simplified development and improved process comprehensibility are delivered because workflows are graphically created and therefore inherently self-evident.
- There is a built-in infrastructure to monitor running processes that supplies runtime transparency. The same graphical diagram that runs the workflow is used to illustrate its current step, previous executed steps, and potential completion paths.
- There is improved runtime flexibility because WF processes can be loaded at runtime from a database and executed without precompilation. Running processes can be changed. An individual running order, for instance, can have an additional approval step added to facilitate unexpected regulatory concerns.
- Simplified and powerful state management is provided. The workflow engine keeps track of the current process step, idles and persists as necessary, and restarts the workflow when appropriate. The workflow engine also allows for interrupting the prescribed process flow and skipping or redoing a step. For instance, it offers tools to transition an order to the earlier customer service step from the shipping step.
- Domain-specific languages can be created by adding a collection of custom activities (WF building blocks) and potentially a custom workflow designer as well.
Introduction

Cloud Service Infrastructure is provided. WF’s integration with WCF permits it to expose itself across the cloud and to access cloud services securely and reliably. When accessing multiple cloud services from a workflow, it becomes a cloud service composition platform.

The goal of this book is twofold: first, to explain what WF is, its value, and how and where its features fit into the product’s overall goal; second, it drills these concepts into you with pervasive hands-on labs. At the end of this book, you should be well-schooled in the “why” and the “how” of WF.

Book Target Audience

This book is targeted at all levels of .NET developers. It covers most aspects of WF. The labs walk you through the exercises step by step. It is appropriate for beginning .NET developers because of the step-by-step nature of the labs. It is appropriate for intermediate and advanced .NET developers because it covers a substantial amount of material that more advanced developers can take time to digest more thoroughly. All developers will also much better understand the “why” of WF. It is much more than a tool to use to create executable diagrams.

How This Book Is Organized

Each hour in this book, with the exception of Hour 1, is packed full of hands-on labs. When a new topic is introduced, it is first explained. The first time an item appears in a lab, you walk through each step and the item is explained in or near where it first appears. When the item appears again later in the hour or in a new hour, it is generally not re-explained. The lab, however, will almost always walk you through the steps to perform the task—although maybe in less detail. The reason I walk you through steps again (and sometimes again) is based on my own experience. I frequently am absorbed learning a new topic and do not want to divert the mental resources from learning the task at hand to remembering or looking up past topics. A contrary viewpoint is that not rewalking through the steps each time a topic reappears in a subsequent lab provides a better learning experience, because readers must learn how to perform the task on their own. This viewpoint is perfectly valid; when and if you want to take this approach, I recommend attempting to perform repetitive actions without reading step-by-step instructions.

By the Way

If you copy and paste code from the electronic version of this book, double-check how the quotes paste. Depending on your program, you might need to manually alter the quotes to make them compatible.
Hour Summary

In Hour 1, “Understanding Windows Workflow Foundation,” a conceptual overview of workflow is provided first. Then WF, the product, is covered in whole. Finally, WF’s main components are covered individually.

In Hour 2, “A Spin Around Windows Workflow Foundation,” you learn how to build a basic sequential workflow by dragging and dropping activities (WF’s building blocks) onto the workflow designer. You also learn to create workflows declaratively using XAML.

In Hour 3, “Learning Basic Hosting,” you dig into hosting a workflow. WF workflows run in the process spaces of another application, referred to as a host. You learn to register events to interact with the host and to register runtime services to change the host’s behavior.

In Hour 4, “Learning Host-Workflow Data Exchange,” you learn how to send data from the host to the workflow and vice versa. Workflows send data to the host via synchronous methods, and hosts send data to the workflow via asynchronous events.

In Hour 5, “Creating an Escalation Workflow,” you learn how to create a workflow that is accessed by two different hosts, which is a very likely scenario. The first host invokes the workflow and allows approval or rejection to be specified. If the process requires further (managerial) approval, the workflow is accessed from the second host.

In Hour 6, “Creating Basic State Machine Workflows,” you learn how to create StateMachineWorkflows. StateMachineWorkflows hold a series of states that each contains a collection of valid events. They are the second most popular type of workflow style (behind sequential workflows).

In Hour 7, “Creating Advanced State Machine Workflows,” you learn how to interact with the state machine workflow using capabilities available only to state machine workflows. These include accessing the current state and overriding the current state to perform the skip and rework pattern.

In Hour 8, “Working with Parallel Activities and Correlation,” you learn to perform tasks concurrently. The workflow runtime handles the logistics of performing the tasks in parallel (or interleaved, as you learn in the hour). You then reconfigure the approval workflow to call for concurrent approval, which requires learning about correlation.

In Hour 9, “Working with the Replicator and While Activities,” you learn to use activities that perform a task n number of times, where n is specified at runtime. The Replicator—one of WF’s advanced control flow activities—can perform the tasks
sequentially or in parallel and also features an early termination clause. The `While` activity is similar to the C# `while` statement.

In Hour 10, “Working with `EventHandlingScope` and Strongly Typed Activities,” you learn to use the `EventHandlingScope` activity that allows one or more events to be received throughout the lifetime of the workflow, such as cancellation and approver maintenance.

In Hour 11, “Creating Data-Driven Workflows,” you learn to use the `ConditionedActivityGroup (CAG)` activity. The CAG is another advanced control flow activity. It allows concurrent processing, like the `Parallel` activity, with a few additions. Most noteworthy, each branch has a `When` condition, and the CAG has an overall `Until` condition.

In Hour 12, “Working with the WF `RuleSet`,” you learn to use WF `RuleSet` technology and add-on products to access `RuleSets` from a database. `RuleSets` are a collection of rules that can be prioritized and configured to reevaluate in case a dependent rule changes. Each rule in a `RuleSet` has a `Then` and an optional `Else` action. The first is executed when the rule evaluates to true and the second when it evaluates to false. You then work with third-party add-ons that allow `RuleSets` to be stored and analyzed in a database and loaded at runtime.

In Hour 13, “Learning to Track Workflows,” you learn to monitor running workflows. First, the tracking architecture is covered. Then you learn to create custom tracking profiles to filter the information that is tracked. Then you learn to augment the extracted information with business information, such as the order number and order amount. This information is useful to produce more meaningful reports and create alerts when, for example, the order amount falls below a threshold amount.

In Hour 14, “Working with Roles,” you learn to control access to the workflow. Each incoming event can be wired to an Active Directory or ASP.NET role provider. Then only users existing in the role provider are permitted access.

In Hour 15, “Working with Dynamic Update,” you learn to modify running workflows. Activities can be added and removed, and declarative rules can be changed. Changing running workflows are one of WF’s primary capabilities. Using this capability in conjunction with WF’s tracking is particularly intriguing.

In Hour 16, “Working with Exceptions, Compensation, and Transactions,” you learn to trap and handle errors and to create transactions. Exceptions and transactions work similarly to the way they work in standard .NET. Compensation is a WF-only capability that is used to correct already completed work in WF.
In Hour 17, “Learning Advanced Hosting,” you learn to use most workflow events to control the workflow from the host, add additional capabilities to alter the workflow runtime, and invoke another workflow from a workflow. You will experiment with the suspended, aborted, and other events. You will add runtime services that control transactions, threading, and other functions. Finally, you learn to call a workflow from another workflow.

In Hour 18, “Working with Web Services and ASP.NET Hosting,” you learn to expose a workflow as a web service, call a web service from a workflow, and to run workflows from ASP.NET.

In Hour 19, “Learning WF-WCF Integration,” you learn to integrate WF with Windows Communication Foundation (WCF), Microsoft’s new distributed technology. WCF can expose WF workflows as services accessible in the cloud and can be used to call out to services from WF workflows. These two products appear to be merging into one unified application server.

In Hour 20, “Creating Basic Custom Activities,” you begin to create your own custom activities. In WF, you are not limited to the activities provided out-of-the-box. Custom activities are a major part of WF. Therefore, five hours are devoted to them. In this hour you will create Customer and CreditCheck activities that encapsulate this “domain” functionality in activities that can be placed on workflows just as can be done with the activities that ship with WF.

In Hour 21, “Creating Queued Activities,” you learn to create activities that execute in multiple bursts and to work with WF’s queuing system, which underlies all WF communication.

In Hour 22, “Creating Typed Queued and EventDriven-Enabled Activities,” you learn to strongly type the data accessed in queues and to create a special type of queued activity called an EventDriven activity.

In Hour 23, “Creating Control Flow Activities Session 1,” you learn to create activities that serve as placeholders for child activities and that schedule their child activities for execution. You can create your own control flow patterns that match the need of your domain. You can also implement general workflow patterns like those found at www.workflowpatterns.com.

In Hour 24, “Creating Control Flow Activities Session 2,” you also learn to implement compensation at the activity level. Then you implement activity validation. Finally, you learn to use attached properties that allow a property, such as a condition, to be passed down from a parent activity to a child activity.
HOUR 1

Understanding Windows Workflow Foundation

What You’ll Learn in This Hour:
- What workflow is in general
- What Windows Workflow Foundation (WF) is
- The main components of WF one-by-one
- Installation instructions and requirements

This hour begins with a general description of workflow because many definitions exist. It then covers Windows Workflow Foundation (WF), first with an overview and then by diving into many of its main elements. Finally, it provides installation instructions and requirements to get you ready for the next 23 hours that are packed full of hands-on exercises.

Describing Workflow and Workflow Systems

This section provides a general overview of workflow and related topics. In addition to offering a general overview, it is intended to give you an understanding of Windows Workflow Foundation’s goals as you continue through this hour and the rest of the book.

A Conceptual Description of Workflow

Workflow is another overloaded technological term. One reason for this is that its meaning has been defined by companies with a vested interest that it matches the
features of their products. Another reason is that it is commonly bound to current technology capabilities. These reasons prevent it from being objectively defined. A workflow is logic—consisting of one or more steps that are predicated by one or more conditions—that a system or person must perform to complete a function. Because the logic or process automated by a workflow generally consists of more than one step that may occur over a period of time, it must track the state of the overall process. Here are some examples of workflows: an order process, an expense report, and rescheduling a missed meeting.

The order process is almost always automated using a traditional computer language. Its control flow is supported through if, while, foreach, and other statements in the C# language. The expense report may be automated using a traditional computer language or a workflow product with a graphical designer. The rescheduling is almost always performed manually. All of these are workflows, or logic. Why, then, are they automated differently? Expense reports are generally routed and escalated to people that must act on them. These touch points are called human intervention. Workflow systems are generally well suited at handling these scenarios that call for human intervention. Therefore, systems with prevalent human intervention are prime workflow system targets. The rescheduling is not automated, or rarely, because for the most part, systems are not yet ready to automate these types of tasks.

Both the order and expense report would generally have lifetimes that span hours, weeks, or months. Therefore the system used to automate them must be able to track their progression, or state. Rescheduling a meeting may only require logic to change the meeting, unless it must also await confirmations. In short, workflow consists of both logic and managing the process state.

A Sample Expense Report Workflow

A sample expense report workflow can make concrete some of the concepts about process discussed so far. Figure 1.1 illustrates a simplified workflow. The process is described directly following the figure.

The workflow receives an expense report and checks the amount; if it is less than or equal to $500, the expense report is approved. Otherwise, manager approval is required. The workflow then removes itself from memory to free resources, and the workflow product waits for the response. The manager may either approve or reject the expense report. In either case, the workflow product reactivates the workflow. If the manager approves, the expense report is marked approved and an approval email is sent to the submittee. If the manager rejects, the corollary rejection process occurs. If the manager does not approve in time, the timeout option is triggered and manager approval is rerequested, which is depicted in the line going back to the Request Manager Approval shape.
Let’s look at some of the benefits gained by using most workflow systems to automate this process:

- **Design-time transparency**—It is clear what the expense report process does just by looking at it.

- **State management**—The workflow system manages keeping track of the current step, removing the process from memory when waiting for manager approval, and going back to the approval step when required. A workflow system provides these state management features for you. In contrast, a process automated via a traditional computer language generally requires the developer to create tables and fields to manage the state themselves.

- **Runtime transparency**—The workflow system can also visually show the current step and the execution path of the process. This feature is generally referred to as tracking (you will learn more about it in the “Tracking” section of this hour).
Many think that there is no way to graphically model application logic because any complex process will have so many control flow statements the diagram will be unreadable, eliminating the promised transparency. A process automated with a graphical workflow application should be self-describing. It should serve the same benefit that a flowchart currently supplies (or at least most of it). Just as a flowchart does not include every single control flow statement, neither should a graphical workflow. Figure 1.1 provides a solid understanding of the process without detailing every control flow statement or diving into the individual steps. Some of the detail held is not relevant to understanding the process and does not need to be added to a workflow at all. Other steps, such as approve order, may be their own workflows with their own detail. They still show on Figure 1.1 as one step, which is appropriate because the details of the expense report approval step aren’t needed to achieve a general understanding of the expense report process.

**Workflow Segmentations**

Workflow is also frequently segmented along in-application, human, and integration lines. Traditional languages, like C#, are generally associated with in-application workflow. Human workflow systems arose from the desire to better support human-centric scenarios. Integration systems or integration-centric workflow systems arose from the need to better automate integration scenarios using dedicated integration systems. Let’s look a little closer at the human- and integration-centric workflow and the tools created to support them, because they are the roots of modern workflow systems.

Human workflow generally describes processes that require substantial human involvement and escalation. Human processes are also frequently nonlinear. A proposal, for example, may go to final approval only to be sent back to initial approval. Tasks and forms to request feedback are critical components of human workflow systems. The tasks and forms will generally be delivered to the requisite people via email, as Outlook tasks, or via a portal (such as SharePoint).

Integration-centric workflow systems are frequently used when connecting systems. Deciding whether an order should be added to SAP that is received from Microsoft CRM, for instance, may require a number of validations, such as whether all the required fields are filled out. These validations are logic, and logic is workflow.

Human workflow systems generally have better form and task support, and integration-centric ones generally process faster and have better transactional support. Although traditional computer languages are still used to automate many human and integration workflow scenarios, these other purpose-built systems have arisen as alternatives.
What Is a Business Process Management System?

The purpose of a business process management system (BPMS) is to create a system that manages processes more completely than human workflow and integration workflow systems and traditional computer languages. There are largely two, potentially overlapping, paths to this process completeness. One is to combine the strengths from human workflow and integration workflow systems. In the previous Microsoft CRM to SAP integration example, a BPMS could be used to integrate the systems and then call on its human workflow support to request human intervention in case of an exception.

The second path is application life cycle and human workflow driven. The application life cycle features generally include tools to deploy the process. Strong monitoring tools are frequently referred to as business activity monitoring (BAM). BAM builds multiple graphical views, portal integration, and business intelligence integration on top of tracking (mentioned in the “A Sample Expense Report Workflow” section) to provide enhanced analysis and monitoring of running processes. BPMSs sometimes allow running processes, such as an order, to be changed. An additional approval step, for instance, could be added to an order. Depending on their roots, human workflow or integration, a BPMS will generally be stronger at either human or system workflow. The human workflow element is gaining market momentum as the ability to interject people into processes is key to increased flexibility, a BPMS staple.

BPMSs also promise businesses better visibility into their processes. That the processes are graphically created and therefore visible combines with BAM to help achieve this visibility. They also frequently promise that business analysts can use their design tools to create processes without IT assistance.

Many BPMS proponents predict that a BPMS is a better way to do all workflow, including in-application. A mass move to using a BPMS for all workflow hasn’t occurred, although this idea has attracted much interest and this may change.

This is one perspective of workflow and BPMSs. If you are interested in learning more or gaining additional information, Wikipedia is a good place to start.

.NET Framework 3.0 and 3.5

Windows Workflow Foundation (WF) is part of the .NET Framework 3.0 and 3.5. The .NET Framework 3.5 is a newer version of the .NET Framework 3.0. Both the .NET Framework 3.0 and 3.5 are add-ons to the 2.0 Framework (see Figure 1.2). Neither
replaces the 2.0 Framework. They simply add new namespaces that make up WF, Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), Windows Cardspace, and other new features.

WCF unifies Microsoft’s current messaging technologies: web services, .NET Remoting, and Enterprise Services/Com+. WCF is significant to WF because it provides a resilient way for WF to communicate with other applications, inside and outside of the firewall. Hour 19 covers WF and WCF integration.

WPF is Microsoft’s new forms technology that looks to unify and improve on its current web and Windows forms capabilities while simultaneously adding multimedia and other features. WF and WPF can both be created using the same markup language, XAML (described in the “XAML Workflows and Serialization” section of this hour), which may prove interesting when creating applications that utilize both.

Windows Cardspace is Microsoft’s new consumer-oriented authentication technology and has no underlying connecting to WF.

WF is very similar across the 3.0 and 3.5 Framework versions. The main difference across framework versions is there are two WCF modeling activities in WF 3.5 that are not in 3.0 (covered in Hour 19 “Learning WF–WCF Integration”).

**Overview of WF**

All applications have workflow, no matter how they automate it. Many applications also look to offer tools to allow others to build custom workflows on top of their product. WF’s goal is to support both of these scenarios and do so at mainstream scale. If you are building an application on Windows that tracks orders, it is WF’s goal to power this workflow. It doesn’t matter if it is an ASP.NET, Windows Forms, Windows Service, or other application. If you are building a platform, like SharePoint, where people build their own custom workflows on top of it, WF’s objective is to provide the tools for this as well. WF also intends to serve all three types of workflow: in-application, human, and integration-centric. WF includes all the foundational elements to create all three types of workflow.
WF must play well with others if it expects other applications to use it to power their workflow and potentially to permit custom workflows to be built with it on their platform. WF has to offer its services in such a way that the applications can use its services and cannot make assumptions about what elements will be available to the application. For instance, if WF is hosted (called from) in a Windows Forms application on a client, there may not be access to SQL Server. It therefore cannot build a hard dependency on storing idled workflows to SQL Server.

These higher-level functions, such as where to store idled workflows, how and if tasks should be stored, and functions such as BAM, are up to the host application to implement, as needed. An application using WF for in-application workflow support, for instance, may not need BAM. The host application can generally start from one of WF’s building blocks to add higher level functionality. The host, for instance, can choose to use WF’s persistence service that stores idled workflows to SQL Server out-of-the-box (OOB), or with minimal effort to another medium, such as Oracle. Likewise, the host can build on WF’s tracking service to supply BAM. It is critical to WF that these building blocks are extremely powerful so that it will be embraced by other products looking for workflow support. Therefore, many WF features may not include the final user interface, but they will be based on an extremely strong infrastructure, as both the persistence and tracking services are.

SharePoint provides a fully functional WF host, as described in the “SharePoint Workflow” section of this hour. Many use SharePoint’s WF implementation just for its hosting capabilities, even those not interested in SharePoint. Microsoft CRM also provides a host. Microsoft is in the process of building a “generic” host based on the combination of WF and WCF. WCF hosting of WF was added in the .NET Framework 3.5. Going forward, this tandem is likely to form a standard host that can be used by any Windows application that chooses to employ workflow functionality. Third parties, such as K2 BlackPearl, have also built hosts on top of WF.

By the end of this hour you will have a better understanding of the rich and powerful capabilities WF offers to create workflow applications and products. These concepts will be expanded on throughout this book.

Next, let’s look at the fundamental pieces of WF.
Standard Modeling Activities

Activities are the unit of design and execution in WF. WF comes with a set of modeling constructs that it calls activities and a workflow designer (see Figure 1.3). The activities are dragged and dropped from the toolbox onto the workflow designer. The properties of the activities are then set. A workflow is the composition of the activities placed on the workflow designer.

WF ships with approximately 30 activities. It calls these activities the Base Activity Library (BAL). The activities are largely segmented as follows: control flow activities, activities that facilitate data exchange between the workflow and the application running the workflow (a Windows Forms application, for example), one that permits arbitrary code to be written, and another group that supplies exception handling.

The control flow activities include a Sequence activity that is a shell for other activities. It is equivalent to {} in C#. It is a block where activities may be added. A Sequence activity can hold a tree of activities, as you will see in Figure 1.10 when its sequential workflow cousin is discussed. Figure 1.4 shows a Sequence activity that contains two other activities.
The While (Figure 1.5) activity loops while the condition associated with it remains true. There is a condition not seen in the figure. Conditions are discussed in the “Rule Capabilities” section of this hour.

The IfElse (Figure 1.6) activity holds one or more branches that are each governed by a condition. The first branch to return true is executed; no other branches are executed.

Another control flow activity is the Listen (Figure 1.7) activity that allows two or more branches to be added that each wait for an external event. The left branch, for instance, may wait for approval and the right branch for rejection. A timer may also be placed in a Listen activity branch. The branch that receives the first event executes unless the timer goes off first, in which case the timer branch executes. The Listen activity supports a prototypical workflow pattern to wait one or more responses and then timeout if response is not received within a specified duration.
A **Parallel** activity executes two or more branches concurrently. (Branches are actually executed in an interleaved fashion as described in Hour 8, “Working with **Parallel** Activities and Correlation.”) The **Parallel** (Figure 1.8) activity will wait for all branches to complete before completing.

**FIGURE 1.8**
The **Parallel** activity.

Advanced control flow activities include the **Replicator** activity, which can process a number of elements specified at runtime and can do so in serial or parallel. It is similar to the C# foreach statement with the additional capability to process in parallel as well as sequentially. This activity is critical to document approval and other scenarios that have different numbers of approvers on different instances and therefore require the number of approvers to be specified at runtime. The **EventHandlingScope** activity is similar to a **Listen** activity but it allows the events to be received multiple times. The combination of the **Replicator** and **EventHandlingScope** activities power much of the OOB SharePoint workflows that require the number of participants (approvers) to be specifiable at runtime and must be changeable throughout the workflow life cycle.

The data exchange activities include **CallExternalMethod** and **HandleExternalEvent**. The first is used to send data from the workflow to the application running the workflow (the host). The latter allows data to be sent from the host to the workflow. There is also a set of activities to expose a workflow as a web service (**WebServiceOutput** and **WebServiceInput**) and call a web service from a workflow (**InvokeWebService**). Finally, **Send** and **Receive** WCF activities exist that can be used only with .NET 3.5. The **Send** activity is used to connect to a WCF endpoint (or any compatible endpoint) from a workflow. The **Receive** activity is used to expose a workflow as a WCF Service.

The **Code** (Figure 1.9) activity points to a handler with standard .NET code. It can be used to add custom functionality to a workflow, although in many cases it is better to use a custom activity, as discussed in the upcoming Custom Activities section.
Multiple Workflow Styles

WF ships with OOB support for three styles of workflows: sequential, state, and data. The third, data, is powered by an advanced control flow activity called the ConditionedActivityGroup that can be placed on both sequential and state machine workflows. It is not a standalone workflow style like the other two. It is included here as a separate workflow style because it is well suited for certain types of processes, as you will soon see. It is therefore sometimes referred to as an activity and at other times as a workflow.

Let’s first start with sequential workflows (Figure 1.10), which is summarized following the figure.

A sequential workflow, like a Sequence activity, serves as a container for a tree of activities. A sequential workflow lays out a process in a linear form from top to bottom. It is similar to a flowchart. The general convention is that sequential workflows are best suited for processes that have a linear path and that are not highly dynamic. The notion is challenged because advanced control flow activities (like the Replicator, EventHandlingScope, and ConditionedActivityGroup that you will learn about shortly) allow sequential workflows to handle some highly dynamic processes. All three styles of workflow are covered, so you can make your own decision.
The process begins with receipt of an order. It is then automatically approved, sent down the more information required path, or automatically rejected. If more information is required, it waits for manual approval or rejection with an option to time-out. As you can see, the process starts at the top, then branches, and continues to move down until completion. Sequential workflow coverage begins in Hour 2, “A Spin Around Windows Workflow Foundation.” You will also want to look at many other hours that cover the different activities because it is these activities, placed on the sequential workflow, that dictate the workflow’s behavior.

State machines are a very common modeling technique—for creating both executing programs and static diagrams—built on process milestones and events. State machines are largely predicated on the notion that processes are dynamic and take many divergent paths to completion. This is why state machines are frequently associated with human-centric processes. In fact, one reason WF proclaims to have the capability to support human workflow is that it includes state machine workflow modeling. Figure 1.11 illustrates a state machine workflow. Directly following the figure is a description.

**FIGURE 1.11**
Sample state machine workflow.

The business milestones are the states (Initial, Approval, and Completed). The events are the text within the states (Approved, Rejected). This means that in the
Approval state both the Approved and Rejected event may be received. The lines from the event to the state represent the transition that occurs when that event is received. For example, when the Created event is received and the workflow is in the Initial state, the workflow transitions to the Approval state. When either the Approved or Rejected event is received in the Approval state, the workflow transitions to the Completed state. It would be just as easy to send the workflow back to the Initiation state from the Approval state if necessary. Processes with many states and many possible gyrations are solid state machine workflow candidates. No prescribed order exists. Each state is autonomous and equally accessible. Hours 6 and 7 describe creating state machine workflows.

Behind each event is a Sequence activity that holds the activities that process the work for the selected event. Figure 1.12 shows the sequential logic executed when the Approved event is selected. In essence, a state machine workflow is an inverse sequential workflow. The events are on top and the sequential logic embedded when using a state machine workflow. In a sequential workflow it is the opposite. Look at the sequential workflow and you will see the Listen activity and event embedded in the sequential logic.
The ConditionedActivityGroup (CAG) activity is an advanced control flow activity that can be placed on either a sequential or state machine workflow. From a process standpoint it can be viewed as an alternative style of workflow to the better-known sequential and state machine styles. Generally, sequential workflows are recommended for deterministic processes that have a well-defined beginning and end. State machine workflows are best suited for dynamic processes that iterate through states without a guaranteed order. Data-driven workflows, on the other hand, are best suited when the data determines the process execution order. Many pricing and promotional algorithms exist that may not execute a branch based on the data at the beginning of a process, but will do so later when other parts of the process change the data the branch depended on. For instance, the order may not be subject to a discount until after tallying the line items.

Depending on your needs, the CAG activity may be the only activity on a sequential workflow. In this case, all your child activities will be embedded in the CAG, and you will have created a data-driven workflow. In other cases, you may embed a CAG into a larger sequential or state machine workflow. In this case, only a subset of your workflow will be data driven.

The CAG (Figure 1.13) has three lanes. Each lane will execute one or more times during the lifetime of the CAG. The lanes execute when their When condition is true during the lifetime of the CAG.
ing the CAG lifetime (lanes with no \texttt{when} condition execute exactly once). The CAG is covered in Hour 11, “Creating Data-Driven Workflows.”

Hour 2 begins covering sequential workflows. State machine workflows are covered in Hours 6 and 7. The CAG is covered in Hour 11. However, workflows are a composition of activities, so you will need to be familiar with the OOB activities, custom activities (covered in the forthcoming “Custom Activities” section of this hour), and other WF functions to create workflows of any type. These topics are interspersed throughout the book.

**Hosting**

WF is not a standalone application. WF is an engine that executes workflows on behalf of a host application. A Windows Forms, ASP.NET, Windows Service, or other Windows application starts the workflow engine running. The host application is then responsible for managing the life cycle of the workflow engine. The host application must remain active while workflows are running and communicate with the workflow engine. If the host application terminates while workflows are running, they will stop running. The host application needs to know the status of workflows the workflow engine is running on its behalf. It achieves this by subscribing to a number of events the workflow engine makes available to it. Three of the most common are \texttt{completed}, \texttt{terminated}, and \texttt{idled}.

The \texttt{completed} event is fired when a workflow completes processing successfully. The \texttt{terminated} event is fired when the workflow completes unsuccessfully. The \texttt{idled} event means the workflow is not complete but inactive. The workflow in Figure 1.10 would enter the \texttt{idled} state when waiting for a response. In a state machine workflow, the only time the workflow is active is when processing the logic behind an event (Figure 1.12). At other times, it is idle waiting for the next event.

How do the workflows receive events if they are idle? This is the job of the workflow engine. It passes the incoming event on to the correct workflow instance. If the workflow is idle, the workflow engine will reactivate it first. This is important because many workflows run for hours, days, or longer. They are generally active only for very short bursts during their lifetime.

In addition to being embeddable in different hosts, the workflow engine must also be configurable to meet the needs of different host applications. For example, when a workflow idles, it should be serialized and stored in a storage medium. If not, a server would hold countless workflow instances over their entire lifetimes. This would devastate scalability and be very risky. If the server went down, the workflows would be lost.
because they are in memory. This is actually the default behavior of WF because it is not aware of the hosts’ needs or capabilities. There may be some scenarios where all workflows have very short life cycles, and no need exists for storage during inactivity. In some environments, such as client scenarios, no SQL Server is available to store the current state of the workflow. For these reasons, WF ships with a number of pluggable services that can be added to the runtime as needed. WF ships with a SQL Server persistence service that can be added. When the persistence service is added, idle workflows are saved to SQL Server. This meets the first criteria—that the service should be available when necessary. It does not meet the second criteria—that the storage medium the workflow is saved to should be flexible. The second option requires extending the base persistence service. It is not available OOB but is an anticipated extensibility point and can be done in a straightforward manner.

Hosting in WF requires starting the engine, choosing which runtime services to add to the runtime, and subscribing to the events you are interested in.

The `WorkflowRuntime` type in the `System.Workflow.Runtime` namespace is the workflow engine. The following code demonstrates instantiating the workflow engine (`WorkflowRuntime` type), adding the persistence service to it, subscribing to the closed event, and starting the workflow engine. This code would be the same regardless of whether the workflow engine was hosted from a Windows Forms, ASP.NET, Windows Service, or other application.

```csharp
WorkflowRuntime workflowRuntime = new WorkflowRuntime();
workflowRuntime.AddService(sqlPersistenceService);
workflowRuntime.WorkflowCompleted +=
    new EventHandler<WorkflowCompletedEventArgs>
    (workflowRuntime_WorkflowCompleted);
workflowRuntime.StartRuntime();
```

Hosting is covered in Hour 3, “Learning Basic Hosting,” and Hour 17, “Learning Advanced Hosting.”

## Tracking

Tracking allows workflow information to be extracted from running workflows. The information is usually saved to a storage medium for monitoring or analysis. Tracking is not added to the WF runtime in its default configuration; it is an optional, pluggable runtime service (just as the persistence service is). WF ships with a SQL Server tracking service that stores running workflow information to SQL Server. If you
need to store the information in a different storage medium, you can customize the tracking service to do so. WF also ships with the WorkflowMonitor SDK sample that reads the information in the SQL Server tracking database and graphically displays it. Figure 1.14 shows a running sequential workflow graphically displayed (the same one displayed at design time in Figure 1.10). The checkmarks illustrate which activities executed. The left pane allows you to select the workflow model and specific instance you are interested in. There is also a filtering mechanism at the top that can be used to select which workflows to show.

WF's very powerful tracking capability is used to extract the tracking information from running workflows. The information can be stored to any medium. WF's SQL Server tracking service is used to store the information to SQL Server. Then the WorkflowMonitor SDK application is used to graphically display and interact with the information that is extracted from the workflow and stored in SQL Server.

If the objective were to compare orders to target and display pie charts and trigger alerts for problem orders, another front-end application would be developed to do this. A BPMS would generally have more tools as part of its BAM offering that simplify developing user interfaces that show workflow information in analytical formats. WF's focus is to make sure the information can be extracted and not on the tools that control the output, although the WorkflowMonitor application is useful in and of itself.

Tracking is covered in Hour 13, “Learning to Track Workflows.”
Rule Capabilities

WF has two types of rules, and it stores rules in two different formats. The first type of rule, a conditional rule, is bound to a control flow activity and is used to determine how the control flow activity processes. For example, which branch of an IfElse activity should execute? The second type of rule in WF, a RuleSet, executes a collection of rules and adds prioritization, reexecution, and other capabilities to the collection of rules.

The next sections look at conditional rules first in both formats and then at RuleSets.

Conditional Rules

The While and IfElse activities covered in the “Standard Modeling Activities” section are both governed by conditional rules. The While activity’s Condition property holds a rule that determines whether the While activity should continue iterating. Common conditions in a While activity are counter < 3 or IsValid = true. In the first example, the While iterates until the counter value is 3 (or more). In the second example, it iterates while IsValid is true. The While activity’s Condition property is evaluated at each iteration to determine whether it should continue iteration. The Condition property can be set as a Declarative Rule Condition or as a Code Condition. Declarative Rule Conditions can be created using the Rule Condition Editor and are stored in an XML format in a .rules file. Figure 1.15 shows the Declarative Rule Condition being created in the Rule Condition Editor.

Figure 1.16 shows the While activity Condition property set to Declarative Rule Condition and pointed to the rule created (named CounterDeclarative) in Figure 1.15. The CounterDeclarative Declarative Rule Condition is now bound to the While activity Condition property, which ensures iteration will stop when counter is 3 or more.
Declarative Rule Conditions are stored in an XML format in a .rules file. The file is extremely verbose and hard to read. The advantage of creating rules declaratively is that they can be changed at runtime without recompilation, stored in a database, and more general tooling support exists for them.

The alternative is to set the While activity Condition property to Code Condition. It then requests a method name. You would insert counter<3 in the method and if the value is less than 3, the method would return true, otherwise it would return false. The method replaces the verbose XML file as a storage medium. Code Conditions do not receive the same level of tooling support as Declarative Rule Conditions in WF.

**RuleSets**

A RuleSet permits a collection of rules to be created that each has a corresponding action. The rules in the RuleSet can be prioritized and configured to reevaluate if a dependent item changes. Reevaluation can be explicitly set on an individual rule or defined overall by setting the RuleSet Chaining property (or both).

When the RuleSet Chaining property is set to Full, rules that have a dependent value changed downstream are automatically reevaluated. Think of a pricing scenario where a discount is applicable if the customer reaches a year-to-date-sales threshold. This discount rule can be reevaluated every time the year-to-date-sales change. The neat part is that WF can do much of this detection automatically via clever CODEDOM programming (a .NET serialization capability) in contrast to forcing the developer to attribute the dependencies.
Figure 1.17 shows a RuleSet with three rules. It is set to Full Chaining, so it will automatically reevaluate. The selected DiscountPercent rule checks if the OrderAmount is larger than the DiscountThreshold. If so, it applies a 5% discount. The other two rules in the RuleSet have their own conditions and actions. Notice that the other two rules (YearlySales and TotalOrderAmount) contain information likely related and relevant to calculating a discount. RuleSet rules also have an optional Else action that executes if the rule condition evaluates to false. A RuleSet is a collection of related rules that solve a common problem such as pricing.

In WF, RuleSets are always stored in .rules files. WF does not add any tools management, central rules storage, or other common functionality found in a commercial rules engine product. However, some add-on products provide some of these capabilities. One of the add-ons stores RuleSets in a SQL database and retrieves them at runtime. Another does some analysis on the rules (Hour 12, “Working with the WF RuleSet”).

RuleSets can be added to a workflow via the Policy activity, which has a RuleSetReference property that binds a RuleSet to a Policy activity. The Policy activity then executes the RuleSet from the workflow.

See Hour 2 for more details on Declarative Rule Conditions and Code Conditions. See Hour 12 for more details on RuleSets.
Custom Activities

This section first describes the reason for custom activities and then describes their technical characteristics.

Reason for Custom Activities

You are not limited to the OOB activities in WF. You can create your own. Creating custom activities is as core to WF as any other capability.

This section discusses three reasons to create custom activities: to improve on an OOB activity for usability reasons, to create domain specific activities, and to create custom control flow patterns.

Improve on OOB Activities

In Hour 17, “Learning Advanced Hosting,” a third-party created synchronous InvokeWorkflow activity is explored. It is looked at because the OOB InvokeWorkflow activity only calls workflows asynchronously. Many scenarios call for calling workflows and waiting for the called workflow to return a response. Some may even choose to create an entirely new set of OOB activities. Improving the OOB activities is a viable scenario for custom activity development, but is not anticipated to be the primary motivation behind creating custom activities.

Create Domain-Specific Activities

The OOB activities provide general functionality. They are host agnostic and know nothing about any vertical domains or any individual enterprise. Many who use WF will find that its value grows proportionally to the amount of domain activities added. If, for example, you want to use WF to model the credit process, you could use the OOB activities for control flow and then augment them with standard code to perform the actual credit process. Alternatively, you could create Customer, CheckCredit, SendNotification, and other custom activities that augment the credit process. Figure 1.18 shows custom activities on a toolbox, and Figure 1.19 shows the custom Customer activity on a workflow and its properties.

---

FIGURE 1.18
CustomActivities toolbox.
Create Custom Control Flow Patterns

A large portion of WF’s utensil and ability to attract a wide range of authors is predicated on there being control flow activities that simplify modeling the respective process. The CAG activity, for instance, makes it possible to model data-driven workflows. The StateMachineWorkflow (workflows are themselves activities) allows for an event-driven style of workflows to be modeled. When nondevelopers are included as authors, there needs to be control flow activities appropriate for them.

Types of Custom Activities

Five types of custom activities exist in WF: basic, long-running, event-driven, control flow, and compound.

Basic activities are similar to standard components. They are called, and then execute, complete their work, and return in a finished state. A sample basic activity is a Customer activity that retrieves customer data. When creating a basic custom activity, you will override its execute method to tell it what to do. You may optionally also customize the activity’s appearance, add it to the toolbox, and validate it. The next code snippet shows the code from the custom Customer activity discussed in the last section (variable declarations are omitted for brevity). You will create custom basic activities in Hour 20.

```csharp
[Designer(typeof(CustomerDesigner), typeof(IDesigner))]
[ToolboxBitmap(typeof(Customer), "Resources.Customer.jpg")]
{
    protected override ActivityExecutionStatus Execute(ActivityExecutionContext executionContext)
    {
```
// Perform preprocessing.
base.RaiseEvent(Customer.InvokingEvent, this, EventArgs.Empty);

SqlConnection dbConn = new SqlConnection(ConnectionString);
SqlCommand getCustomer = new SqlCommand("GetCustomer", dbConn);
getCustomer.CommandType = System.Data.CommandType.StoredProcedure;
getCustomer.Parameters.AddWithValue("@CustomerNumber", CustomerNumber);

dbConn.Open();

using (SqlDataReader custReader =
getCustomer.ExecuteReader(CommandBehavior.CloseConnection))
{
    if (custReader.Read())
    {
        CustomerName = custReader["CustomerName"].ToString().Trim();
        CustomerCreditLimit =
double.Parse(custReader["CustomerCreditLimit"].ToString());
        CustomerType = custReader["CustomerType"].ToString().Trim();
        CustomerYtdSales =
double.Parse(custReader["CustomerYtdSales"].ToString());
        CustomerHoldRules =
custReader["CustomerHoldRules"].ToString().Trim();
    }
}

Console.WriteLine
("The customer number is: " + CustomerNumber);
Console.WriteLine
("The customer name is: " + CustomerName);
Console.WriteLine
("The customer credit limit is: " + CustomerCreditLimit);
Console.WriteLine
("The customer type is: " + CustomerType);
Console.WriteLine
("The customer YTD sales is: " + CustomerYtdSales);
Console.WriteLine
("The customer hold rules is: " + CustomerHoldRules);
return ActivityExecutionStatus.Closed;
}

Following is a summary of the additional custom activity types in WF. These activities build on the steps necessary to create a basic activity.

Long-running activities do not complete on initial call. They continue processing on another thread. They return control to the workflow while the processing occurs. When the processing completes on the other thread, the long-running activity notifies the workflow that it is complete (via WF's internal queuing system). The same Customer activity can also be coded as a long-running activity. The reason is that if the
database call to retrieve the customer information was to a local database and was fast, it would make sense to do all work in the execute method. If the call was across the firewall and not so fast, it might be better to return control before the customer data is retuned. Hour 21, “Creating Queued Activities,” demonstrates modifying the Customer activity, shown in the previous code listing, that currently executes in one part to execute in two distinct parts.

Event-driven custom activities can wait for external events inside of a Listen or other activity. Custom event-driven activities are described in Hour 22, “Creating Typed Queued and EventDriven-Enabled Activities.”

Composite or control flow activities, as previously mentioned, allow you to create your own control flow patterns. These activities are responsible for determining which child activities should execute and then scheduling them for execution. Creating custom composite activities is discussed in Hour 23, “Creating Control Flow Activities Session 1,” and Hour 24, “Creating Control Flow Activities Session 2.”

A compound activity is prepopulated with other activities. It is useful when a pattern of activities is commonly used. Figure 1.20 demonstrates a CreditCheck compound activity that uses the custom Customer activity to retrieve customer data; then it uses the CheckCredit activity to determine if the customer is on credit hold. Finally, it evaluates the return results in an IfElse activity. The left branch processes the order, and the right branch rejects it. You will create this compound activity in Hour 20.

All noncomposite activities derive (directly or indirectly) from System.Workflow.ComponentModel.Activity. Composite activities all derive from
System.Workflow.ComponentModel.CompositeActivity. These are the same classes the BAL activities derive from.

Custom activities are covered in Hours 20 to 24.

**XAML Workflows and Serialization**

XAML (pronounced “ZAML”) is an XML language used in both WF and WPF. It allows a hierarchical collection of objects, their relation to .NET types, and input and output data to be described. In WF, XAML can be used to describe the tree of workflow activities, the .NET types that encapsulate the logic, and the data sent to and received from the activities. In WPF it does the same for the user interface controls.

WF supports specifying the tree of activities in both code and XAML. Why the need for XAML? As an XML dialect, XAML receives the benefits of the investment being made in XML by Microsoft and others. These investments include the capability to store XAML workflows in SQL Server. WF will load a XAML workflow at runtime without requiring precompilation. If the workflows are expressed in XAML, each workflow instance can be stored in a database. Calculating the difference to the baseline process and between individual workflows is as simple as using XSLT, XQUERY, or other XML manipulation languages.

If no compilation occurs, can’t invalid workflows be loaded? No, WF’s validation capability, which you learn about in Hours 15 and 24, validates workflows when they are loaded. It checks the workflow for structural fidelity. You are free to extend the validation on any workflow to add business-specific checks, such as if a `BankBeginTransfer` activity exists on the workflow, ensuring there is also a `BankEndTransfer` activity.

As a whole, storing workflow models in a database and then retrieving them at runtime to execute is a compelling possibility. The capability to then store each workflow instance is also compelling.

No matter what format you choose, the graphical workflows are saved in a format that can be seen and edited. This means you can modify the code directly if needed.

The next listing shows a simple workflow with one `Code` activity expressed in XAML. The one immediately following shows the same workflow expressed in code. If you are familiar with Windows Forms development, you will notice that the code representation matches the format of a Windows Forms application.

```xml
<SequentialWorkflowActivity x:Name="Workflow2XOMLOnly"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
```
<DelayActivity TimeoutDuration="00:00:00" x:Name="delayActivity1" />
</SequentialWorkflowActivity>.

The same workflow expressed in code:

```csharp
this.CanModifyActivities = true;
this.delayActivity1 = new System.Workflow.Activities.DelayActivity();
 //
 // delayActivity1
 //
 this.delayActivity1.Name = "delayActivity1";
 this.delayActivity1.TimeoutDuration = System.TimeSpan.Parse("00:00:00");
 //
 // Workflow2
 //
 this.Activities.Add(this.delayActivity1);
 this.Name = "Workflow2";
 this.CanModifyActivities = false;
```

XAML workflows and serialization are covered in Hour 2.

**Dynamic Update**

Two of WF’s primary goals are to support long-running processes and to provide increased process agility. One common denominator to both these goals is the ability to evolve processes. It is a simple fact that processes change. If software cannot handle it, the changes are implemented out-of-band. This not only diminishes the efficiency of the process itself but reduces reporting accuracy because the data used to compile reports does not include the out-of-band operations. With dynamic update, activities can be added and removed from workflows (a change can be implemented via an add-remove combination). Declarative rules can also be changed.

The next code listing demonstrates adding a Delay activity to a workflow. The details will be covered in Hour 15, “Working Dynamic Update,” but this is all the code necessary to change a running workflow.

```csharp
// use WorkflowChanges class to author dynamic change and pass
// it a reference to the current Workflow Instance
WorkflowChanges workflowTochange = new WorkflowChanges(this);

// Create a new Delay, initialized to 2 seconds
DelayActivity delayOrder = new DelayActivity();
delayOrder.Name = "delayOrder";
delayOrder.TimeoutDuration = new TimeSpan(0, 0, 2);

// Insert the Delay Activity to the TransientWorkflow collection
// (the workspace) between the two Code Activities
workflowTochange.TransientWorkflow.Activities.Insert(1, delayOrder);
```
// Replace the original Workflow Instance with the clone
this.ApplyWorkflowChanges(workflowTochange);

Although the capability to change a running workflow is powerful, it is also unnerv-
ing. WF has controls in place to make changing running workflows more palatable. The first is that each workflow can be set to stipulate when and if dynamic update should be permitted. The default is always. Workflows have a DynamicUpdateCondition property (Figure 1.21 shows it set to a Declarative Rule Condition). This property can be set to determine when and if dynamic update should be permitted. If you want to disallow it completely, return false from the Condition property. If you want to selectively permit it, for example, to be allowed at one part of the workflow that changes from instance to instance, apply the appropriate condition.

![Properties](image)

The second is that two ways exist to store per-workflow instance changes. If using XAML workflows, each instance can be saved to a database and changes can be com-
pared to other instances and to the baseline process. If trends are noticed, the base-
line process can be changed. If using tracking, it will store the changes, and the mechanism you use for displaying tracking information can include them. The Work-
flowMonitor SDK application, for instance, will show changes made when viewing the workflow instances. It is not always appropriate to view each and every workflow. However, if you store each XAML workflow instance in a database or use tracking, the information is there and available to report on.

The third is that WF’s validation is called when dynamic update changes are applied. This prevents erroneous changes from being made to running workflow instances.

Dynamic update is an extremely powerful feature that goes hand-in-hand with WF’s per-instance tracking capabilities. It seems feasible that processes will continually be changed going forward with WF and controlled with strong reporting tools that lever-
age tracking. The capability to change running workflows is common in BPMSs, and...
WF provides a solid set of tools to do it and build even more powerful solutions to change running workflows and track these changes.

XAML and dynamic update can appear to overlap. XAML can be loaded at runtime without precompilation but cannot be changed on a running workflow. XAML is good for tools and for database loading. Dynamic update is used to change an actual running process. Dynamic update can be performed on workflows expressed in either XAML or code. Rules are a little different. Declarative rules (those expressed in .rules files) are also more tool friendly than their code counterparts and can also be loaded without precompilation, which makes database storage compelling. Only declarative rules can be changed with dynamic update, though. Hour 13 demonstrates loading declarative rules from a database. Hour 15 covers Dynamic Update.

**WF and WCF**

The first subsection in this section provides an overview of WCF and conceptually describes using WF and WCF together. The second subsection discusses the product features that integrate the two.

**WF and WCF: Conceptual Overview**

WCF, WF’s counterpart in the .NET 3.x Framework, is Microsoft’s preferred technology for hosting and accessing network endpoints. WCF and WF have a symbiotic relationship. By leveraging WCF’s core strengths, WF workflows can be securely and reliably exposed across the network. WF workflows can also access network endpoints. Let’s look at a couple of examples. If you have a workflow that performs a credit check, and you want to expose it to clients within and across the firewall, you can expose it as a WCF service. Likewise, if you wanted to access a remote service to receive a client’s credit score from a workflow, you would use WCF to access the remote service from the workflow. A workflow hosted by a WCF host is referred to as a WorkflowService.

WCF provides one programming model and runtime for distributed computing on the Microsoft platform. It subsumes all previous Microsoft distributed technologies, including web services, web service enhancements, .NET remoting, and enterprise services. WCF supplies the most thorough web service standard support on the Microsoft platform. WCF can listen for and access network endpoints via HTTP, TCP, named pipes, and just about any other protocol.

WCF services (endpoints) are protocol and host agnostic. For example, a service communicating across the firewall can choose the WCF HTTP binding, which sends stan-
standard SOAP over HTTP using basic security. If the service needs additional security, it can use WCF’s sibling HTTP binding that includes WS* (special web service security) security. Finally a service communicating behind the firewall can utilize the TCP binding that uses compiled SOAP. Therefore, the same service can support all three communication patterns. There simply needs to be one WCF endpoint created for each communication pattern the service supports.

WCF separates the service (the application logic) from the endpoint. The service can be implemented in standard .NET code or as a WF workflow. It is up to you to choose whether to use standard .NET code or a workflow to provide the service logic. Leveraging WCF, WF could expose the same workflow over HTTP to remote clients and TCP to local clients.

Like WF, WCF can be hosted in any .NET 2.0 plus application domain, such as a Windows Service, IIS, or Windows Activation Service (IIS in Windows Server 2008) process. Unlike WF, there is a WCF project type for IIS that simplifies hosting in IIS. This enables WCF services to take advantage of IIS’s message-based activation, security, process pooling, and other capabilities. There really is no reason not to use WCF for distributed computing on the Microsoft platform.

WF and WCF: Integration Specifics

Two activities are designed to support WF-WCF integration: Receive and Send. The first is used when exposing a WF workflow as a WCF endpoint. The second is used to access a remote client from a WF workflow. The Send activity is host agnostic—it can be called from any WF host. The Receive activity is not. It can be used only when WF is hosted by WCF.

The remote client does not actually have to be a WCF client. It just must be exposed using a protocol and binding that WCF can communicate with (see Hour 19).

There are also two WCF workflow projects in Visual Studio 2008 (Figure 1.22). The first contains a sequential workflow project and a WCF contract (interface). The second holds state machine workflows and an accompanying contract. These project types are relevant when exposing a workflow as a service.

The Send and Receive activities are very powerful. They both provide synchronous and asynchronous communication capabilities. The Send activity has a very similar structure to a method call: expected return type, method to call, and parameters to pass. The Receive activity has a very similar structure to a called method. The WF
communication activities made available in the .NET Framework 3.0 do not offer such a natural way to call out from and into a workflow.

Many prefer to host all WF workflows in WCF for the following reasons: the robust communication implementation of the Send and Receive activities, the IIS and Windows Activation Services hosting support offered to WCF, Microsoft’s apparent emphasis toward WCF as WF’s hosting apparatus, especially going forward and the expectation that WF and WCF will continue to be interweaved until they look like one.

WCF and WF integration is discussed in Hour 19, “Learning WF-WCF Integration.”

**SharePoint Workflow**

This section on SharePoint workflow and the next on designer rehosting and external modeling discuss topics not covered in later hours. They are discussed in this hour because they are important to the overall WF vision and understanding them will help you understand WF’s goals. SharePoint workflow is not covered because it is not part of the base WF product. It is built on top of the base WF. Following the hands-on labs would also require that you install SharePoint, which is no small task. The reasons for not including designer rehosting and external modeling in a subsequent hour are described in the next section.

This section contains three subsections. The first provides a conceptual overview of SharePoint workflow, the next describes the Visual Studio version of SharePoint workflow, and the last the SharePoint Designer version.

**SharePoint Workflow Overview**

SharePoint is Microsoft’s collaboration, document management, and general information environment. Many companies use SharePoint to implement departmental and event-specific sites for groups of people to share information and work together.
A common site may have document libraries, tasks, form libraries, a calendar, and other data specific to the site’s purpose (Figure 1.23). SharePoint can also be described as a portal. SharePoint also features business intelligence and much more. For our purposes, SharePoint is an information hub where groups of people collaborate.

Workflow is a common requirement for SharePoint because the vast information in it needs to be routed. Expense reports and documents stored in it, for example, need to be approved, rejected, and escalated. SharePoint tasks (which can be integrated with Outlook tasks) also provide a great destination for workflows to assign work. If Jane, for instance, must approve John’s expense report, a task can be assigned to Jane. Jane can then access this task in her SharePoint or Outlook task list, or, depending on configuration, it may be emailed to her.

SharePoint can be customized by both technically savvy business users and developers. One of its selling points is that its sites and other items can be created and configured without the need for IT. There are three levels of customization performed in SharePoint. The first is performed via SharePoint configuration forms, the second is performed in the SharePoint Designer, and the third in code. For the most part, both users and developers can configure SharePoint via the forms. SharePoint Designer (FrontPage’s replacement) configuration is generally performed by power users. Changes requiring code are performed by developers.

SharePoint workflow features both Visual Studio and SharePoint Designer workflow authoring options. Additional custom activities are targeted at the SharePoint host in both offerings. A tighter relationship exists between the workflow and the forms used to collect its data. The most common forms are task and initiation. The first is used to collect information for tasks that are assigned by the workflow. Remember that approval requests, for example, are assigned via tasks. The second is used when the workflow is started to collect information. SharePoint Workflow (in the Microsoft Office SharePoint Server offering) also includes a number of OOB workflows that support approval and other common scenarios. They are highly flexible and good to work
with to get an idea of the types of workflows that can be created in SharePoint workflow.

**SharePoint Workflow Visual Studio**

In Visual Studio SharePoint workflow, developers use the same Visual Studio environment used in standard WF development. The main difference is a collection of additional activities tailored to the SharePoint host. A couple of project templates also reference SharePoint assemblies and simplify deployment to the SharePoint host. Figure 1.24 demonstrates the Visual Studio SharePoint workflow. The toolbox shown is specifically for SharePoint. It contains a number of task-centric and other activities specific to the SharePoint host.

The capability to add a collection of custom activities specific to a domain, in this case the SharePoint domain, makes WF much more useful. It is also demonstrates how effectively you can extend WF by adding a collection of activities specific to your business and/or host. The SharePoint custom activities are used in addition to the BAL activities. WF’s BAL activities provide the general control flow and other generic capabilities, and the SharePoint custom activities provide host-specific functionality.

**SharePoint Workflow SharePoint Designer**

The Visual Studio version of SharePoint workflow allows developers to create workflows to run in SharePoint. However, SharePoint allows technically savvy business
users to customize many aspects of SharePoint. Consistent with this premise, workflows can also be designed in the SharePoint Designer. The SharePoint Designer includes a similar collection of activities to that found in Visual Studio SharePoint workflow and packages them in a different design environment. Figure 1.25 shows the SharePoint Designer workflow designer.

It is rules driven because Microsoft thought it preferable to a graphical one, which is a controversial decision. However, the big picture is that it takes a similar set of base functionality and exposes it in a different designer targeted at a less technical user. Custom activities can be added to the SharePoint Designer as well. Adding a collection of domain-specific custom activities to this designer significantly increases the potential for technically savvy business people to create workflows by themselves. The SharePoint Designer saves the workflow in a XAML format, which is then loaded by WF and executed.

**Designer Rehosting and External Modeling**

At times you may need to graphically create workflows outside of Visual Studio and graphically monitor and interact with them at runtime. The most common reason to author workflows outside of Visual Studio is to allow nondevelopers to author workflows. There will almost always be a need for a collection of domain-specific custom
activities in addition to the custom authoring experience if business users are to create workflows. There are two ways workflows can be authored outside of Visual Studio. The first, referred to as designer rehosting, permits the design tools used to create workflows in Visual Studio to be rehosted in another application. The second is to use an external modeling tool to produce a workflow model that can be executed by the WF runtime.

Figure 1.26 shows the workflow designer hosted in a Windows Forms application. If the current designer is sufficient and you want the authoring performed outside of Visual Studio, rehosting the designer is likely appropriate. The design experience consists of a combination of the designer and the activities used in the designer. For instance, moving the Code activity to a custom designer does not eliminate the need to write code to support its execution. You should therefore use or create activities appropriate for those targeted to use the rehosted designer. You can even create custom control flow activities if you think the configuration requirements of the OOB ones are not appropriate to your target audience. The WorkflowMonitor application (looked at in the “Tracking” section) demonstrates rehosting the designer for monitoring and interacting with running and previously executed workflows.

If there is a need for an entirely difference experience, then using a different tool altogether is probably more appropriate than rehosting the designer. Maybe Visio has the required authoring tools because that is what the business people know. Maybe a rules-driven designer, such as the SharePoint Designer, is called for.
When rehosting the designer, you can also include its serialization tools to create the XAML or code, depending on which format you choose. When using external tools, it is up to you to serialize the information, which will almost always be to XAML.

Designer rehosting and using an external modeling tool are not covered in this book. They are included in this hour because they are important to WF's overall vision, but they are advanced topics, especially external modeling. By the end of this book, you will have experience in custom activity authoring (to create your domain activities) and WF in general, readying you for one of these two alternative authoring options. I will post a sample of external modeling that I have worked on some on my blog at www.reassociates.net and will try to do the same for designer rehosting. You can also see http://msdn.microsoft.com/en-us/library/aa480213.aspx for an example of designer rehosting.

The business users do not have to create an executable workflow. It is possible to allow them to lay out the process using a rehosted designer or external modeling tool. Their work can then be loaded by a developer and completed. This may be a better approach, in some cases, than producing static requirements that are not leverageable by the developer who created the process or not connected to the final executable application in any way.

### By the Way

This hour first outlined workflow in general, discussed common workflow categorizations, and looked at BPMSs. It then provided a brief overview of WF. Then it covered a number of WF's capabilities. Let's now look at six of the main benefits offered by WF, having gone through its capabilities:

- **Design-time transparency**—This is intrinsic because the workflow that executes also graphically describes the process.

- **Runtime transparency**—The tracking infrastructure and tools built to leverage the infrastructure supply visibility to running processes.

- **Runtime flexibility**—XAML allows processes to be run without precompilation and be retrieved from databases, and dynamic update allows processes to be changed at runtime.

- **State management**—The workflow keeps track of the current step, idles and persists as necessary, restarts when appropriate, and can even skip or redo steps.

- **Domain-specific languages**—Domain-specific languages can be created by adding a collection of custom activities and potentially a customer designer to go along.
Participate in network—WF’s integration with WCF permits it to expose itself across the network and to access network (cloud) services securely and reliably.

These benefits combine to form a better way to create application logic.

The rest of this book consists of explanations and accompanying labs that walk you through most areas of WF.

**Installation Instructions**

You should install Visual Studio 2008 and .NET Framework 3.5 to follow along with the hands-on labs in this book. If you are using Visual Studio 2005 and .NET Framework 3.0, you will not be able to complete the hands-on exercises in Hour 19, “Learning WF-WCF Integration.” There are also other minor incompatibilities. The two known differences, creating projects and dependency properties, are pointed out the first time they are encountered.

You should also install the WF, WCF, and CardSpace samples that can be found at http://msdn2.microsoft.com/en-us/library/ms741706.aspx. These samples include the aforementioned WorkflowMonitor application that is used in exercises in various hours. These samples are also generally useful for you to explore and are referred to for topics not covered in the book or to obtain more information on topics that are covered.

To follow the hands-on labs, download the samples from my website at www.reassociates.net. I recommend that you unzip them to your root directory to create the following directory structure: c:\SamsWf24hrs\Hours. Each hour and the associated labs will be appended to the base directory. Retaining this structure will make it easier to follow the labs. Each hour will have at least one completed solution in a Completed subdirectory. Many will have more granular solutions to help you follow along throughout the hour. See the readme.txt file included in the root directory of each hour’s labs for details for each hour.

See the next section for Visual Studio 2005 installation instructions and the one immediately following for Visual Studio 2008 instructions.

**Visual Studio 2005 and .NET Framework 3.0 Installation Directions**

Visual Studio 2005 and the .NET Framework 3.0 require the following to be installed:

- Windows 2003, XP, or Vista. (The .NET Framework 3.0 should already be installed if you have Vista.)
Installation Instructions

- SQL Server or Express 2000 or later.
- SQL Server Management Studio for SQL Express or SQL Server.

Visual Studio 2008 and .NET Framework 3.5 Installation Requirements/Directions

- Windows 2003, XP, or Vista.
- SQL Server or Express 2000 or later.
- SQL Server Management Studio for SQL Express or SQL Server.
- Visual Studio 2008 and the .NET Framework 3.5. (The .NET Framework 3.5 should be included with Visual Studio 2008.)
A

ABCs of WCF, 426
aborting workflow, AdvancedHostingForms, 374
accessing properties in tracking, 295-296
actions, attributing methods called from, 269
ActiveDirectoryRole, 307
activities, 16-18
AddApprover EventDriven, 224
  adding to Declarative Rules, 60-61
to event handlers view (EventHandlingScope), 227
to extend workflow, 123-124
to sequential view (EventHandlingScope), 227
to toolboxes across projects, 480-481
to workflow, 54
to workflow projects, 49-51
ApprovalReceived HandleExternalEvent, 126
basic activities, 455
BasicQueued, 493
CallExternalMethod, 18
  adding, 103-108
  tracking, 292
CancelWorkflow EventDriven, 224
CheckCredit
  binding Customer properties to, 466-467
  configuring, 125-127
  running workflows with, 467
Code, 18
  configuring, 447
  Replicatorworkflow, 207
CompensatableSequence, 357
CompensatableTransaction Scope, 366
compound activities, 456, 470
CompoundCreditCheck, 471
ConditionedActivityGroup, 19
ConditionedActivityGroup (CAG) activity, 22-23
configuring for ParallelActivityDesigner, 540
configuring for Replicator workflow, 206
  ActivityExecutionContext, 208-209
  CallExternalMethod, 206
activities

Code, 207
HandleExternalEvent, 206
Replicator ChildInitialized property, 209-210
Replicator Initialized property, 207-208
configuring in
  event handlers view
  (EventHandlingScope), 229
sequential view
  (EventHandlingScope), 228
custom activities, 453
event-driven custom activities, 32
reasons for, 29-30
types of, 30-33
Delay, 374, 383
domain-specific activities, 454
event-driven activities, 149, 513
  adding to state machine workflows, 151-152
state machine workflows, 162-164
updating state machine workflows, 153-159
EventHandlingScope, 18, 219
FaultHandlers, 345
HandleExternalEvent, 18, 124
  tracking, 290-291
HandleExternalMethod, 101-102
IfElse, 17, 350
  adding, 103
InvokeWebService activity, 18
calling web services from workflows, 405
InvokeWorkflow, 384-385
  host prematurely exits, 385-386
host waits, 387
synchronous calls and parameters, 387-394
Listen, 17, 119
  extending workflow, 123
  Timeout branch, 127
More Info
  CallExternalMethod, 125
multiburst activities, 483
Parallel, 18
placing on
  Replicator workflow, 205
Policy, creating RuleSet, 260-264
queued activities, 484
Receive, 18
  ContractsAndWorkflows (WCF), 428-429
  WCF workflows, 438-441
RejectionReceived
  HandleExternalEvent, 126
removing from workflows, 54
Replicator, 203
Replicator activity, 18
Send, 18
  configuring in WCF, 446-447
Sequence, 16
SetState, 150
single-burst activities, 483
State, 149
Statefinalization, 150
Statelnitialization, 150, 155
StateMachine
  WorkflowActivity, 149
SynchronizationScope, 185-186
TransactionScope, 344, 365
WebServiceFault, 407-409
configuring, 409
WebServiceInput, 18, 398
WebServiceOutput, 18, 398
While, 17, 203, 213
XAML + code, adding and configuring activities, 64-65
Activity Designer, 475-476
activity handlers, 354-355
activity life cycles, 568-570
activity programming model, 475
  ActivityDesignerTheme, 476-478
designer components, 475-476
activity validation, 475
ActivityCondition type
  GeneralControlFlow activity, 561-563, 567
ActivityDesignerTheme, 476-478
ActivityExecutionContext, 208-209, 485, 530
ActivityExecutionContext class, 485
ActivityExecutionStatus, 459, 568-570
ActivityListenerEvent,
  CustomerEventDrivenActivity, 519
ActivityValidator, 557
AddApprover EventDriven activity,
  EventHandlingScope, 224
adding
deborative Rules, 60-61
to event handlers view
  (EventHandlingScope), 227
to extend workflow, 123-124
to sequential view
  (EventHandlingScope), 227
to toolboxes across projects, 480-481
to workflow, 54
to workflow projects, 49-51
XAML + code workflow, 64-65
CAG, to workflows, 245-246
CallExternalMethod activities, 103-108
cancellation and early completion, GeneralControlFlow activities, 542-545
cancellation handlers, CancellationWorkflow, 356-357
ChildActivityContinuation handler, 531
code to QueueItemAvailable handler, basic custom queued activities, 490-491
code-beside
to Approver1Lane, 247-249
to Approver2Lane, 250-251
to Approver3Lane, 252-253
for Replicator activities (EventHandlingScope workflow), 230-231
to update workflow midflight (EventHandlingScope workflow), 231-232
for While activities, 213-215
Compensate activity to fault handlers, CompensationWorkflow, 361-362
compensation handlers, CompensationWorkflow, 359-360
controls, to forms, 109
custom activities, to workflows, 459-460
custom designers to
GeneralControlFlow activity, 552
custom Policy activity to toolbox, external RuleSet, 275
CustomerEventArgs class, to CustomerQueuedFromService Activity, 508-509
DependencyProperty type, to host-workflow data exchange, 99-101
DependencyProperty, to customer custom activities, 460-462
DiscountPercent rule, 260-262
embedded CustomerState class, CustomerQueuedFromService, 496
event handlers to activities, customer custom activity, 469-470
events, state machine workflows, 160-162
existing custom CreditCheck activity, 465-468
FaultHandlers, CompensationWorkflow, 361
GetCustomer method, CustomerQueuedFromService, 495
GetCustomerOnWorkerThread method, CustomerQueuedFromService, 496-498
graphics as images and associating classes, ToolboxBitmap, 479
hosting logic, 76-77
IfElse, to evaluate credit hold status, 467
IfElse activities and rules, 103
Level2Role support, 315-317
member variables
to Replicator workflow, 204-205
state machine workflows, 170
parallel execution options, GeneralControlFlow activities, 536-542
parallel support, to Replicator, 215-216
parameters, to InvokeWorkflow activity, 389-391
persistence, to basic and escalation forms, 131-132
roles, ASP.NET role providers, 310-311
rule conditions, 54
RuleSet tracing, 265
service references to WCF hosts, 434-435
single child activity execution, GeneralControlFlow activities, 529-532
states, state machine workflows, 160-162
States, to state machine workflows, 151-152
total order validation, to concurrent approval workflow, 199
TotalOrderAmount rule, 262
tracking to basic and escalation forms, 130-131
user tracking data, UserTrackingRecords, 299
variable declarations, to CAG projects, 244
WF runtime, to forms/hosts, 109-112
workflow-level FaultHandlers, 345-347
YearlySales rule, 262-263
AdvancedHostingForms project, running, 372-373
aborting workflow, 374
suspending and resuming workflow, 374
terminating workflow, 374
Annotate and Match Derived Types options, TrackingProfileDesigner, 303
App.config file, ContractsAndWorkflows (WCF), 430-431
approval, 187
ApprovalReceived HandleExternalEvent activity, 126
Approved event, 21
Approver1Lane, configuring for CAG, 246-249
Approver2Lane, configuring for CAG, 249-251
Approver3Lane, configuring for CAG projects, 252-253
ASMX file, 411
ASP.NET hosting, 413-414
  creating ASP.NET web forms, 415-416
  instantiating WorkflowRuntime, 414-415
  running workflows, 418
  starting WorkflowInstance, 416-417
ASP.NET role providers, installing, 308-309
  adding roles and users, 310-311
attached dependency properties, creating, 562-563
attributing methods
  called from actions, 269
  called from conditions, 270-271

B
BAL (Base Activity Library), 16
BAM (business activity monitoring), 13, 283-285
base activities, setting, 457-458
Base Activity Library, 16
basic activities, 455
  overview, 456
basic custom queued activities, 487
  adding code to
    QueueItemAvailable handler, 490-491
  overriding
    Execute method, 489-490
    Initialized method, 487-488
    Uninitialize method, 492
performing preliminary custom activity setup, 487
running workflow, 493
updating hosts to send data to queues, 492
basic forms
  adding persistence to, 131-132
  adding tracking to, 130-131
basic order forms, implementing
  MoreInfo method, 129-130
BasicHostForm, 114
BasicHostingProject, creating, 74
BasicQueued activity, 493
binding Customer properties to
  CheckCredit activities, 466-467
bindings, WCF, 426-427
BPM (Business Process Management), 285
BPMS (business process management system), 13
BranchCondition rule, 332
BranchesToExecute, 557
business activity monitoring, 13, 283
Business Process Management, 285
business process management system, 13

C
CAG (ConditionedActivityGroup), 241-244
  adding to workflows, 245-246
  adding variable declarations, 244
  configuring Approver1Lane, 246-249
  configuring Approver2Lane, 249-251
  configuring Approver3Lane, 252-253
running with three lanes, 254
running with two lanes, 251
CallExternalMethod activity, 18
  adding, 103-108
  configuring, 391
    for approver, 193, 196
    for Replicator workflow, 206
  tracking, 292
calling
  methods from rules, 267-271
  XAML-only workflow from host, 68-69
calling web services from workflows, 404-406
Canceling state, 569
cancellation, GeneralControlFlow activities, 542-545
cancellation handlers, 355-356
Cancellationhandlers, 343
CancellationWorkflow, 355
  adding cancellation handlers, 356-357
  modeling, 356
CancelWorkflow EventDriven activity, EventHandlingScope, 224
CanInsertActivities method, GeneralControlFlow activity, 553
changes in workflow, Dynamic Update (sample), 337-340
changing
  declarative rule conditions, Dynamic Update, 329-332
  RuleSets
    CodeDom, 332-334
    .rules file, 334-337
ChannelToken property, 447
CheckApprovalStatus, 407
Condition property

CheckCredit activity
  binding Customer properties to, 466-467
  configuring, 472
  running workflows with, 467
CheckForFaults, 407
  child activities
    executing in GeneralControlFlow activities, 532-535
    validating in GeneralControlFlow activity, 559-560
  child activity execution logic, 529
ChildActivityContinuation handler, 539
  adding, 531
  updating, 534
    to check for early, 543
children, 354-355
  classes
    creating classes that extend IServiceProvider, 564
    custom designers,
      GeneralControlFlow activity, 552
    custom validation,
      GeneralControlFlow activity, 557
GeneralControlFlowToolBoxItem class, 556
  ToolBoxItem, GeneralControlFlow activity, 554
  code-beside
    adding
      to Approver1Lane, 247-249
      to Approver3Lane, 252-253
    for Replicator activities (EventHandlingScope),
      230-231
    to update workflow midflight (EventHandlingScope),
      231-232
    for While activities, 213-215
      Level2Role, adding, 315
      updating for roles, 313-314
code-only workflow, 63-64
  CodeDom, changing RuleSets, 332-334
  combo boxes, SetState, 174
  CompensatableSequence, 357
  CompensatorTransaction activity, 366
  Compensate activity, adding to fault handlers (CompensationWorkflow),
    361-362
  Compensating state, 570
  compensation, 357
    implementing, 570-571
  compensation handlers, adding to CompensationWorkflow, 359-360
  compensation, overriding (default compensation handlers), 570-571
CompensationHandlers, 344
  CompensationWorkflow
    adding
      Compensate activity to fault handlers, 361-362
      compensation handlers, 3 59-360
    FaultHandlers, 361
    modeling, 359
    moving Throw activity to CompensatableSequence activity, 362
    performing preliminary setup, 358
    registering SQL persistence service, 358-359
  completed event, 23
  components of state machine workflows, 149-150
  CompositeActivity base type, 528
  CompositeActivityDesigner, 535
  Compound activities, 456, 470
    creating and modeling, 470-474
  CompoundCreditCheck activities, 471-473
  concurrent approval workflow,
    191-196
    adding activities, 197-198
    adding total order validation, 199
    configuring CallExternalMethod activity for approver, 193, 196
    configuring HandleExternalMethod activity for approver, 194, 197
    running, 200
  concurrent processing, correlation, 187
  Condition property, 128
    adding to
      GeneralControlFlow, 545
conditional rules, 26-27
ConditionedActivityGroup, 241
ConditionedActivityGroup (CAG) activity, 19, 22-23
conditions applying to tracking, 296-297
attributing methods called from, 270-271
Conditions property, RuleSet, 328
configuring activities
  in event handlers view (EventHandlingScope), 229
  in sequential view (EventHandlingScope), 228
  XAML + code, 64-65
activities for Replicator workflow, 206
  ActivityExecutionContext, 208-209
  CallExternalMethod, 206
  Code, 207
  HandleExternalEvent, 206
  Replicator ChildInitialized property, 209-210
  Replicator Initialized property, 207-208
Approver1Lane for CAG, 246-249
Approver2Lane for CAG, 249-251
Approver3Lane CAG projects, 252-253
CallExternalMethod activity, 391
CallExternalMethod activity for approver, 193, 196
CheckCredit activities, 125-127, 472
Code activities, 55-56, 447
Customer activities in CompoundCreditCheck activities, 473
EventDriven activities, state machine workflows, 162-164
HandleExternalMethod activity for approver, 194, 197
IfElse activity, 401
scheduling service projects, 379-380
TransactionWorkflow, 364-365
Visual Studio debugger to not trap CLR exceptions, 349-350
WebServiceFault activity, 409
WebServiceInput activity, 400-401
WebServiceOutput activity, 402-403
workflow to use external rules, 276-277
connecting to WCF endpoints from WF, 445
updating workflows, 445-448
Console Application
  creating BasicHostingProject, 74
  creating hosts, 75
    adding hosting logic, 76-77
    adding monitoring events, 83-84
    adding persistence service, 80-83
  pausing hosts, 77-78
  persistence and tracking databases, 85-86
  registering closed and terminated events with hosts, 78-80
  running the workflow, 80, 86-87
  updating hosts via configuration, 87-89
  creating simple workflows, 75
  creating solution and projects, 73
  creating WorkflowsProject, 74
console application clients (WCF), 434
  adding client code to call the service, 436-437
  service references to WCF hosts, 434-435
  generated files, 435
  running, 437
console host, 52
ConsoleApplicationWcfWorkflowHost, 443
  retrieving WorkflowRuntime, 443-444
  running, 444
ContextToken, 429
ContractsAndWorkflows (WCF), 427
  App.config file, 430-431
  modeling workflow, 429
  Receive activity, 428-429
  reviewing interfaces, 427-428
control flow activities, 525-526
  GeneralControlFlow activities, 527
    adding cancellation and, 542-545
    adding parallel execution option, 536-542
    adding single child activity, 529-532
    creating, 545-548
    creating nonexecuting control flow, 527-529
    designer hierarchy, 535-536
    executing child activities, 532-535
    GeneralControlFlow activity, 526
ControlFlowCustomActivitySolution, 527
controls, adding to forms, 109
cookie usage, 411
correlation, 187
  concurrent processing, 187
  CorrelationWorkflows project, 188
  event payload (EventArgs), 190
  local services, 188-189
  CorrelationToken.OwnerActivityName
  property, 193
  CorrelationWorkflows project, 188
  CreateQueue method,
  CustomerEventDrivenActivity, 520
  credit hold status, adding IfElse to
  evaluate, 467
  CreditCheck activity
  adding, 465-468
  importing, 465
  custom activities, 453
  adding to workflows, 459-460
  creating domain-specific
  activities, 454
  custom control flow patterns, 454
  custom customer activity, 457
  event-driven custom activities, 32
  improving on OOB activities, 454
  reasons for, 29-30
  technical overview, 455-456
  types of, 30-33
  custom activities that access typed
  services, 509-513
    CustomerQueuedFromService-
    Activity, 507-508
      adding, 508-509
    CustomerQueuedFromTypeService, updating, 513
  custom control flow patterns, 454-455
  custom designers
    adding to GeneralControlFlow
    activity, 552-554
  modifying in GeneralControlFlow
  activity, 565
  custom validation, GeneralControlFlow
  activity, 556-557
    associating with activities, 560
    classes, 557
    overriding Validate method,
    557-558
    testing validation, 561
    validating child activities, 559-560
  Customer activity
    configuring in
    CompoundCreditCheck
    activity, 473
    updating to retrieve information
    from databases, 462-464
  customer custom activity, 457
    adding event handlers, 469-470
    adding existing custom
    CreditCheck activity, 465-468
    adding to workflows, 459-460
    overriding Execute method,
    458-459
    setting base activities, 457-458
    updating customer activities to
    receive input, 460-461
    updating Customer activity to
    retrieve information from,
    462-464
  CustomerEventArgs class, adding,
  508-509
  CustomerEventDrivenActivity, 515-516
    ActivityListenerEvent, 519
    CreateQueue method, 519
    creating DoSubscribe
    method, 518
    DeleteQueue method, 520
    DoUnSubscribe method, 519
    implementing Subscribe
    handler, 517
    implementing UnSubscribe
    handler, 518
    overriding Execute method,
    516-517
    overriding Initialize method, 516
    ProcessQueueItem method, 520
    running the workflow, 521-522
  CustomerNumber dependence
  property, adding, 516
  CustomerNumber property, 461
  CustomerQueuedFromService,
  494-495
  CustomerQueuedFromServiceActivity,
  498-499, 507-508
    adding CustomerEventArgs class,
    508-509
    overriding Execute method,
    499-500
    overriding Initialized method, 499
    QueueItemAvailable handler, 500-503
    running workflow, 504
    updating hosts to use new ser-
    vices, 503-504
  CustomerQueuedFromTypedService
    modifying, 510-513
    updating hosts, 513
  CustomerQueuedFromTypedService-
  Activity, modifying, 509-510
  customers, retrieving from
databases, 463
  CustomerState class, adding to
  CustomerQueuedFromService, 496
  CustomerState object, updating, 510

D
  data-driven workflows, 243
  debugging workflow projects, 52-53

How can we make this index more useful? Email us at indexes@samspublishing.com
Declarative Rule Conditions, 26-27
updates, 35
declarative rule conditions, changing with Dynamic Update, 329-332
Declarative Rules, 60-62
default compensation handlers, overriding, 570-571
DefaultWorkflowSchedulerService, running, 381
Delay activity, 374
modifying workflow to use, 383-384
workflow, 34-35
DeleteQueue method, CustomerEventDrivenActivity, 520
dependency properties
adding, 462
calling web services from workflows, 404
creating, 398
DependencyProperty, 127, 336
adding to customer custom activities, 460
DependencyProperty type, adding to host-workflow data exchange, 99-101
design-time transparency, 11
derigner components, activity programming model, 475-476
derigner hierarchy, GeneralControlFlow activities, 535-536
derigner rehosting, 42-43
DiscountPercent rule, adding, 260-262
distributed transaction coordinator (DTC), 132
DLL file, 411
domain-specific activities
creating, 454
custom activities, 29
DoSubscribe method, CustomerEventDrivenActivity, 518
DoUnSubscribe method, CustomerEventDrivenActivity, 519
downloading
external RuleSet application, 271-272
WorkflowMonitor, 141-143
DTC (distributed transaction coordinator), 132
dynamic update, 34-36, 319-320
Dynamic Update
applying from the inside, 321-322
applying from the outside, 324-327
changing declarative rule conditions, 329-332
changing rules in a rules file, 332
changing RuleSets, via .rules files, 334-337
exploring workflow changes sample, 337-340
selecting which version to run, 329
TransientWorkflow property, 320
WorkflowChanges, 320
WorkflowInstances, 321
dynamic update, XAML, 36
dynamically updating rules, 327-328
DynamicUpdateCondition, 35
DynamicUpdateCondition property, 319
DynamicUpdateFromInside workflow, running, 323-324
DynamicUpdateFromOutside, 326
settings, WCF, 431
WorkflowServiceHost, 432
enhancing state machine workflow, 165-166
escalation forms
adding persistence to, 131-132
adding tracking to, 130-131
implementing new interface members, 128-129
escalation local services, 121
escalation workflow
adding
persistence to basic and escalation forms, 131-132
tracking to basic and escalation forms, 130-131
extending, 122
adding activities to, 123-124
configuring CheckCredit activities, 125-127
updating workflow code-beside file, 127-128
retrieving tracking data, 132-135
retrieving workflow from persistence, 135-137
running the solution, 137-140
updating forms
implementing MoreInfo method in basic order, 129-130
implementing new interface members, 128-129
updating local services, 120-122
evaluating
attached properties with Execute method (GeneralControlFlow activity), 566
post-execution results, RuleSet, 267
pre-execution results, RuleSet, 265-266
event handlers, adding to customer custom activities, 469-470
event handlers view
  adding activities to (EventHandlingScope), 227
  configuring activities in (EventHandlingScope), 229
event handling activities, configuring event handling scope, 224
event mapping to local services, 162
event payload (EventArgs), correlation, 190
event-driven activities, adding to state machine workflows, 151-152
event-driven custom activities, 32
EventArgs, 97-99, 508-509
  correlation, 190
EventArgs class, 389
EventDriven activities, 149, 513-514
  configuring for state machine workflows, 162-164
CustomerEventDrivenActivity, 515-516
  updating for state machine workflows, 153-159
EventHandlingScope activity, 18
EventHandlingScope, WCA.exe, 233-234
EventHandlingScope activity, 219-220
EventHandlingScope workflow
  configuring event handling activities, 224
  configuring sequential activities, 223
  placing activities on event handlers section, 221-223
  placing activities on sequential section, 220-221
running, 225
  strongly typed activities, 234-237
XAML, 225-226
EventHandlingScope workflow (advanced), 226
  adding activities to event handlers view, 227
  adding activities to sequential view, 227
  adding code-beside for Replicator activities, 230-231
  adding code-beside to update workflow, 231-232
  configuring activities in event handlers view, 229
  configuring activities in sequential view, 228
  preparatory setup, 230
  running, 232-233
events
  adding to state machine workflows, 160-162
  raising for Windows Forms host, 112-113
examining
  project files, 63-64
  workflow, XAML, 65-66
exception handler, 345
exception handling, 344-345, 348
  adding workflow-level FaultHandlers, 345-347
  configuring Visual Studio debugger to not trap CLR exceptions, 349-350
  hierarchical exception handling and throw activity, 350
  catching, 351-352
  handling, 352-353
  reconfiguring, 353-354
  Throw activity, 353
  updating, 350-351
  modeling ExceptionWorkflow, 345
  StateMachineWorkflows, 345
exceptions
  catching in IfElse activity, 351-352
  handling in IfElse activity, 352-353
ExceptionWorkflow
  Modeling, 345
  Updating, 350-351
Execute method
  GeneralControlFlow activity, evaluating attached properties, 566
  overriding
    in basic custom queued activities, 489-490
    in customer custom activity, 458-459
    inCustomerEventDrivenActivity, 516-517
    inCustomerQueuedFromServiceActivity, 499-500
  updating
    to process multiple children, 533
    updating customer activity to receive input, 461
executing
  child activities,
    GeneralControlFlow activities, 532-535
  standard parallel activities, 182-183
  synchronized parallel activities, 185-186
Executing state, 569
ExecutionMode property, 557
expense report workflow (example), 10-12

How can we make this index more useful? Email us at indexes@samspublishing.com
extending workflow

extending workflow, 122
adding activities to, 123-124
configuring CheckCredit activities, 125-127
updating workflow code-beside file, 127-128
external data exchange, 92
external modeling, 42-43
external RuleSet application, 271
adding custom Policy activity to toolbox, 275
configuring workflow to use external rules, 276-277
downloading, 271-272
preparing workflow to use external rules, 275-276
running sample workflow, 272-273
running workflow with external rules, 277
uploading rules to external databases, 273-274
ExternalDataExchange, 484
ExternalDataExchangeService, registering with hosts, 393-394

F

fault handler, 345
FaultHandler, 344
FaultHandlerActivity.Fault, 348
FaultHandlers, 343
adding, 345-347
Faulting state, 569
FaultMessage, 429
first-level approval or rejection, running solutions, 137
flow patterns, custom activities, 30
form code-behind logic, updating to work with
StateMachineWorkflowInstance members, 170-178
forms
adding controls to, 109
adding WF runtime to, 109-112
basic forms, 130-132
basic order forms, implementing MoreInfo method, 129-130
escalation forms, 128-129
updating for state machine workflows, 170
FreeformActivityDesigner, 535
FreeFormDesigner, 535

G

GeneralControlFlow activities, 527
adding cancellation and early completion, 542-545
adding parallel execution option, 536-542
adding single child activity execution, 529-532
creating
GeneralControlFlowBranch and adding conditions, 545-548
creating nonexecuting control flow activities, 527-529
designer hierarchy, 535-536
executing child activities, 532-535
GeneralControlFlow activity, 526
ActivityCondition type, 561-563
adding attached properties, 561-563
creating classes that extend, 564
Execute method, 566
GetCondition, 564
IExtenderProvider.CanExtend, 565
modifying custom designer, 565
removing, 567
SetChild method, 564
updating workflow, 568
adding custom designers, 552-553
classes, 552
overriding CanInsertActivities, 553
overriding OnCreateNewBranch, 552
testing designers, 553-554
custom validation, 556-557
associating with activities, 560
classes, 557
overriding Validate method, 557-558
testing validation, 561
validating child activities, 559-560
ToolBoxItem, 554-556
classes, 554
GeneralControlFlowToolBoxItem, 556
testing, 556
GeneralControlFlowBranch, 557
creating, 545-548
GeneralControlFlowBranch validation, removing, 567
generated files, console application clients (WCF), 435
GetCondition, GeneralControlFlow activity, 564
GetCustomer method, adding to CustomerQueuedFromService, 495
GetCustomerCalledFromActivity method, updating, 511
GetCustomerOnWorkerThread, updating, 511
GetCustomerOnWorkerThread method, 495
adding to CustomerQueuedFromService, 496-498
GetWorkflowRuntime, 111
graphics, adding as images (ToolboxBitmap), 479

H

HandleExternalEvent, 124
HandleExternalEvent activities, tracking, 290-291
HandleExternalEvent activity, 18
HandleExternalMethod activity configuring for approver, 194, 197
host-workflow data exchange, 101-102
HandleExternalEvent activity, configuring for Replicator workflow, 206
handling exceptions, 348
hierarchical exception handling, throw activity and, 350
catching exceptions in IfElse activity, 351-352
handling exceptions in IfElse activity, 352-353
reconfiguring, 353-354
Throw activity, 353
updating ExceptionWorkflow, 350-351
hierarchical states, state machine workflows, 159-160
historical states, 175
host forms, updating for Replicator workflow, 210-212
host-workflow communication, 93
host-workflow data exchange, 91-93
local service, 93
local service interface, 96-97
local service projects, 94
payloads (EventArgs), creating, 97-99
solution and projects, creating, 94
Windows Forms projects, creating, 95-96
workflow, creating, 99
adding DependencyProperty type, 99-101
HandleExternalMethod activity, 101-102
workflow projects, creating, 95
hosting, 23
ASP.NET, 413-414
creating ASP.NET web forms, 415-416
instantiating WorkflowRuntime, 414-415
running workflows, 418
starting WorkflowInstance, 416-417
completed event, 23
idled event, 23-24
overview of, 72
terminated event, 23
hosting logic, adding, 76-77
hosts
adding WF runtime to, 109-112
building from scratch
adding monitoring events, 83-84
adding persistence service, 80-83
creating
BasicHostingProject, 74
creating hosts, 75-80
creating simple workflows, 75
creating solution and projects, 73
creating WorkflowProject, 74
persistence and tracking databases, 85-86
running the workflow, 86-87
updating hosts via configuration, 87-89
calling XAML-only workflow, 68-69
creating, 75
adding hosting logic, 76-77
adding monitoring events, 83-84
adding persistence service, 80-83
pausing hosts, 77-78
persistence and tracking databases, 85-86
registering closed and terminated events with hosts, 78-80
running the workflow, 80, 86-87
updating hosts via configuration, 87-89
pausing, 77-78
in workflow projects, 51-52
registering closed and terminated events with, 78-80
ExternalDataExchange-Service, 393-394
running the workflow, 80
updating for CustomerQueuedFromTyped Service, 513
to pass in OrderAmount, 465

How can we make this index more useful? Email us at indexes@samspublishing.com
for roles, 311-312
to send data to queues
(basic custom queued activities), 492
updating to run three workflows,
scheduling service projects, 380
updating to use
ManualWorkflowScheduler-Service, 381-383
updating to use new services,
CustomerQueuedFromServiceActivity, 503-504
WCF, 431
adding service references to,
434-435
running, 433
starting, pausing, and closing,
432-433
WorkflowServiceHost, 432
Windows Forms host, 108
adding controls to, 109
adding WF runtime to,
109-112
creating workflow instances and raising initial events,
112-113
implementing local service,
113-116
running the project, 116
human workflow, 12

I-J-K
IActivityEventListener, 515
ICompensateableActivity-Compensate, 570
ICompensateableActivityCompensate method, 570
idled event, hosting, 23-24
IEventActivity, 515
IEventActivityQueueName, 515
IExtenderProviderCanExtend method,
GeneralControlFlow activity, 565
IfElse
adding activities and rules, 103
adding to evaluate credit hold status, 467
configuring for Compound activities, 471
IfElse activity, 17
catching exceptions in, 351-352
configuring, 401
exception handling, 350
handling exceptions in, 352-353
implementing
compensation, 570-571
interface members in escalation forms, 128-129
local services for Windows Forms host, 113-114, 116
MoreInfo method in basic order form, 129-130
Subscribe handler,
CustomerEventDrivenActivity, 517
UnSubscribe handler in
CustomerEventDrivenActivity, 518
importing CreditCheck activity, 465
improving
on OOB activities, 454
transparency, 57
Initialize method, overriding
in basic custom queued activities,
487-488
in CustomerQueuedFromServiceActivity, 499
in CustomerEventDrivenActivity, 516
installing
ASP.NET role providers, 308-309
adding roles and users, 310-311
.NET Framework 3.0, 44
.NET Framework 3.5, 45
Visual Studio 2005, 44
Visual Studio 2008, 45
WorkflowMonitor, 141-143
integration, WF and WCF, 37-38
integration-centric workflow systems, 12
interfaces
ContractsAndWorkflows (WCF), reviewing, 427-428
modifying in WCF workflows, 438
InvokeWebService activity, 18
calling web services from workflows, 405
InvokeWorkflow activity, 384-385
host prematurely exits, 385-386
host waits, 387
synchronous calls and parameters, 387-394
IServiceProvider, creating classes that extend, 564

L
Leve2Role support, 315-317
life cycle topics, 484
ActivityExecutionContext, 485
WorkflowQueue, 484-485
life cycles, activity life cycles
ActivityExecutionStatus, 568-570
OnActivityExecutionContextLoad, 570
OnActivityExecutionContextUnload, 570
Listen activity, 17, 119
  extending workflow, 123
  Timeout branch, 127
loading service, 377
loading workflows, TrackingProfileDesigner, 288-289
local service, 92
  host-workflow data exchange, 93
local service interface
  creating, 389
  host-workflow data exchange, 96-97
local service project, creating, 388-389
  host-workflow data exchange, 94
local services
  correlation, 188-189
  escalation workflow, updating, 120-122
  event mapping to, 162
  implementing for Windows Forms host, 113-116
LocalService.cs, 97
LocalServiceEscalationEventArgs.cs, 121
LocalServiceWorkflow, 106

M
ManualWorkflowScheduler-Service, 378
  running, 382-383
  updating hosts, 381-382
member variables
  adding
    to Replicator workflow, 204-205
    to state machine workflows, 170
    updating for roles, 312
methods, calling from rules, 267-268
  attributing methods called from actions, 269
  attributing methods called from conditions, 270-271
modeling
  CancellationWorkflow, 356
  CompensationWorkflow, 359
  Compound activities, 470-474
  ExceptionWorkflow, 345
  scheduling service projects, 379-380
  TransactionWorkflow, 364-365
  workflow
    ContractsAndWorkflows (WCF), 429
    While activity for sequential processing, 213
  workflows, calling web services from workflows, 404-405
  workflows as web services, 399
  XAML-only workflow, 67-68
modifying
  custom designer,
    GeneralControlFlow activity, 565
  CustomerQueuedFromTyped-Service, 510-513
  CustomerQueuedFromTyped-ServiceActivity, 509-510
  state machine workflows, 160
  adding new states and events, 160-162
  configuring EventDriven activities, 162-164
  TrackingProfiles, 300-301
  workflow to use Delay activities, 383-384
monitoring
  RuleSet, 264
  adding tracing, 265
  evaluating post-execution results, 267
  evaluating pre-execution results, 265-266
  events, adding to hosts, 83-84
MoreInfo CallExternalMethod activity, 125
MoreInfo method, implementing in basic order form, 129-130
moving Throw activity into CompensatableSequence activity, 362
multiburst activities, 483
  EventDriven activities, 513-514
  CustomerEventDrivenActivity, 515-522
  WorkflowQueue, 484

N
.NET 3.5, 423
.NET Framework 3.0, 13
  installing, 44
.NET Framework 3.5, 13
  installing, 45
nonexecuting control flow activities, creating, 527-529
NullCustomerRule, 409

O
OnActivityResultExecutionContextLoad, 570
OnActivityResultExecutionContext-Unload, 570
OnCreateNewBranch method, GeneralControlFlow activity, 552

How can we make this index more useful? Email us at indexes@samspublishing.com
OOB (out-of-the-box)

custom activities, 29
SqlTrackingService, 135
persistence service, 81

opening TrackingProfileDesigner, 288-289
OperationValidation, 429
OrderAmount, updating hosts to pass, 465
OrderEscalationForm running, 314
updating
code-beside, role processing, 313-314
hosts, role processing, 311-312
member variables, role processing, 312
workflow, role processing, 312
workflow model, role processing, 313

OrderInitEventArgs, 98
out-of-the-box, 81
out-of-the-box (OOB) activities, improving on, 454

overriding
CanInsertActivities method, GeneralControlFlow activity, 553
default compensation handlers, 570-571

Execute method
basic custom queued activities, 489-490
customer custom activity, 458-459
CustomerEventDrivenActivity, 516-517
CustomerQueuedFromServiceActivity, 499-500

Initialize method
basic custom queued activities, 487-488
CustomerEventDrivenActivity, 516
CustomerQueuedFromServiceActivity, 499
OnCreateNewBranch method, GeneralControlFlow activity, 552
states, 177

Uninitialize method, basic custom queued activities, 492
Validate method, GeneralControlFlow activity, 557-558

pausing hosts, 77-78
WCF, 432-433
workflow projects, 51-52

payloads (EventArgs), host-workflow data exchange, 97-99

persistence
adding to basic and escalation forms, 131-132
retrieving workflow from, 135-137

persistence databases, creating, 85-86
persistence service, adding to hosts, 80-83
placing activities on Replicator workflow, 205

Policy activity
adding to toolbox, external RuleSet, 275
creating RuleSet, 260-264

preparations for state machine workflows, 151

preparatory setup,
EventHandlingScope workflow, 230
preparing XAML-only workflow, 67-68
ProcessQueueItem method,
CustomerEventDrivenActivity, 520
profiles, uploading to database (TrackingProfile), 293

Program.cs, 52
project files, examining, 63-64

properties
accessing in tracking, 295-296
attached dependency properties, creating, 562-563
ChannelToken, 447
Condition, 128
CorrelationToken.OwnerActivityName, 193
Customer, binding to CheckCredit activities, 466-467
CustomerNumber, 461
dependency properties, 404
Performing preliminary custom activity, 487
running workflow, 493
updating hosts to send data to queues, 492
QueueItemAvailable handler
adding code to, in basic custom queued activities, 490-491
CustomerQueuedFromService-Activity, 500-503
Rebuilding client proxies, WCF workflows, 441-442
Receive activity, 18
ContractsAndWorkflows (WCF), 428-429
WCF workflows, 438-441
receiving SOAP faults in client workflows, 409-410
reconfiguring Throw activity, 353-354
registering
closed and terminated events with hosts, 78-80
ExternalDataExchangeService with hosts, 393-394
SQL persistence service, 358-359
RejectionReceived
HandleExternalEvent activity, 126
removing
activities from workflows, 54
GeneralControlFlowBranch validation, 567
Replicator activity, 18, 203
sequential processing, 213
Replicator ChildInitialized property, configuring, 209-210
Replicator Initialized property, configuring for Replicator activity, 207-208
Rule, 257
rule conditions, adding to workflows, 54
Reconfiguring Throw activity, 353-354
registering
closed and terminated events with hosts, 78-80
ExternalDataExchangeService with hosts, 393-394
SQL persistence service, 358-359
RejectionReceived
HandleExternalEvent activity, 126
removing
activities from workflows, 54
GeneralControlFlowBranch validation, 567
Replicator activity, 18, 203
sequential processing, 213
Replicator ChildInitialized property, configuring, 209-210
Replicator Initialized property, configuring for Replicator activity, 207-208
Replicator workflow
adding member variables to, 204-205
configuring activities, 206-209
placing activities on, 205
running parallel Replicator, 216-217
updating host forms, 210
adding ViewQueues method, 210-211
passing number of reviewers to workflow, 211-212
running, 212-213
updating to run in parallel, 215
resuming workflow,
AdvancedHostingForms374
retrieving
customers from databases, 463
tracking data, 132-135
TrackingProfiles, 299-300
workflow from persistence, 135-137
WorkflowRuntime, ConsoleApplicationWcfWorkflowHost, 443-444
ReturnValue, 428
roles, 307-308
adding to ASP.NET role providers, 310-311
code-beside, updating, 313-314
hosts, updating, 311-312
Level2Role, 315-317
member variables, updating, 312
setting up ASP.NET role providers, 308-309
workflow, updating, 312
workflow model, updating, 313
Rule, 257
rule conditions, adding to workflows, 54

How can we make this index more useful? Email us at indexes@samspublishing.com
RuleDefinitions, 327, 332, 336
RuleDefinitionsProperty, 327
.rules file, changing with RuleSets, 334-337
rules, 258
calling methods from, 267-268
attributing methods called from actions, 269
attributing methods called from conditions, 270-271
condition rules, 26-27
Declarative Rules, creating, 61-62
DiscountPercent, 262
RuleSets, 27-28
TotalOrderAmount, 262
updating dynamically, 327-328
uploading to external databases, 273-274
YearlySales, 262-263
RuleSet, 27-28, 257-259
creating project, 260
creating via Policy activity, 260-264
external RuleSet application, 271
adding Policy activity to toolbox, 275
configuring workflow to use external rules, 276-277
downloading, 271-272
preparing workflow to use external rules, 275-276
running sample workflow, 272-273
running workflow with external rules, 277
uploading rules to external databases, 273-274
monitoring, 264
adding tracing, 265
evaluating post-execution results, 267
evaluating pre-execution results, 265-266
terminology, 258
RuleSet Chaining, 27
RuleSets, 327
changing with CodeDom, 332-334
running
AdvancedHostingForms project, 372-374
CAG projects
with three lanes, 254
with two lanes, 251
concurrent approval workflow, 200
console application clients (WCF), 437
ConsoleApplicationWcfWorkflowHost, 444
DefaultWorkflowSchedulerService, 381
DynamicUpdateFromInside workflow, 323-324
DynamicUpdateFromOutside, 326
EventHandlingScope workflow, 225
EventHandlingScope workflow (advanced), 232-233
hosts, WCF, 433
ManualWorkflowSchedulerService, 382-383
OrderEscalationForm, 314
parallel Replicator, 216-217
parallel workflows, 183-184
Replicator workflow, 212-213
solutions
first-level approval or rejection, 137
second-level approval and tracking, 139-140
second-level approval or rejection, 138
state machine workflows, 159, 178-179
TransactionWorkflow, 365-366
Windows Forms host projects, 116
workflow
basic custom queued activities, 493
external RuleSet application, 272-273
RuleSet, 263-264
with external rules, 277
WorkflowMonitor SDK sample, 141
downloading and installing WorkflowMonitor, 141-143
workflows
ASP.NET, 418
building hosts from scratch, 86-87
CustomerEventDrivenActivity, 521-522
CustomerQueuedFromService Activity, 504
TrackingProfiles, 293, 297-298
with CheckCredit activities, 467
with data coming from the databases, 464
XAML + code workflow, 66
running SQL script to create
TestTransaction database, TransactionWorkflow, 365
runtime services, 375-377
custom activity queued from service, 493
CustomerQueuedFromService, 494
state machine workflows

sequential view
adding activities to (EventHandlingScope), 227
configuring activities in (EventHandlingScope), 228
sequential workflow, 19-20
Sequential Workflow Console
Application projects, creating, 48-49
Sequential Workflow project, Parallel activity tester project, 182
Sequential Workflow Service Library project, 425
sequential workflows, 243
creating, 473
serialization, XAML, 33
ServiceOperationInfo, 428
ServicesExceptionNotHandled, 370
SetChild method, GeneralControlFlow activity, 564
SetState activity, 150
SetState combo box, 174
SharePoint, WF, 15
SharePoint Designer, 41
SharePoint workflow, 38
overview, 38-39
SharePoint Designer, 41
Visual Studio, 40
ShowTracking method, 134
simple workflows, creating, 75
single child activity execution, adding to GeneralControlFlow activities, 529-532
single-burst activities, 483
SOAP faults, 407-409
configuring conditions, 409
receiving in client workflows, 409-410
running workflow solution, 410
WebServiceFault activity, configuring, 409
SQL persistence service, registering, 358-359
SQL script
running to create SQL components, 462
running to create TestTransaction database, TransactionWorkflow, 365
SqlTrackingQuery, 130, 298
SqlTrackingService, 119, 130, 135
SqlWorkflowPersistenceService, 81
standard parallel activities, executing, 182-183
Started, 370
starting hosts, WCF, 432-433
State activity, 149
state introspection
state machine workflows, 159-160
StateMachineWorkflowInstance and, 165
state machine workflows, 20-21, 243
adding States and event-driven activities, 151-152
components of, 149-150
creating the project, 150
enhancing, 165-166
hierarchical states, 159-160
modifying, 160
adding new states and events, 160-162
configuring EventDriven activities, 162-164
overview, 147-149
preparatory work, 151
running, 159, 178-179
state introspection, 159-160
StateMachineWorkflowInstance and state introspection, 165
How can we make this index more useful? Email us at indexes@samspublishing.com
state machine workflows

updating
- EventDriven activities, 153-159
- form code-behind logic to work with, 170-178
- forms and adding member variables, 170

state management, 11
state transitions, 176
StateFinalization activity, 150
StateInitialization activity, 150
StateMachineWorkflowActivity, 149
StateMachineWorkflowInstance, state introspection and, 165
StateMachineWorkflowInstance members, updating form code-behind logic, 170-178
StateMachineWorkflows, exception handling, 345
states
- adding to state machine workflows, 160-162
- overriding current state, 177
States activities, adding to state machine workflows, 151-152
Stopped, 370
strongly typed activities,
- EventHandlingScope, 234-237
StructuredCompositeActivity-Designer, 535
Subscribe method, implementing in CustomerEventDrivenActivity, 517
suspending workflow,
- AdvancedHostingForms, 374
SynchronizationScope activity,
- 185-186
synchronized parallel activities, executing, 185-186
synchronous calls, InvokeWorkflow, 387-394

T

terminated event, hosting, 23
terminated events, registering with hosts, 78-80
terminating workflow,
- AdvancedHostingForms, 374
testing
- designers, GeneralControlFlow activity, 553-554
- GeneralControlFlowToolBoxItem class, 556
- validation, GeneralControlFlow activity, 561
TestTransaction database, running SQL script, 365
Thread.Sleep statements, 383
Throw activity
- hierarchical exception handling and, 350-353
- moving into CompensatableSequence activity, 362
- reconfiguring, 353-354
Timeout branch, configuring Listen activity, 127
ToolboxBitmap, 478
- adding graphics as images and associating classes, 479
toolboxes, adding activities to toolboxes across projects, 480-481
ToolBoxItem, GeneralControlFlow activity, 554-556
- classes, 554
- testing GeneralControlFlowToolBoxItem, 556
total order validation, adding to concurrent approval workflow, 199
TotalOrderAmount rule, adding, 262
tracking, adding RuleSet tracing, 265
tracking architecture, 284-285
tracking data, retrieving, 132-135
tracking databases, creating, 85-86
tracking tables and logic, tracking, 86
TrackingChannel, 285
TrackingProfileDesigner, 286-287
- Annotate and Match Derived Types, 303
- examining how profiles are processed, 294
- opening and loading workflows, 288-289
- tracking CallExternalMethod activities, 292
- tracking HandleExternalEvent activities, 290-291
- tracking workflow level events, 289-290
- uploading profiles to database and running workflow, 293

TrackingProfiles, 285
- conditions, applying, 296-297
- modifying, 300-301
- retrieving, 299-300
- saving profiles and running workflows, 297-298
- saving to files, 299-300
TrackingProfileDesigner, 286-287
examining how profiles are processed, 294
opening and loading workflows, 288-289
tracking CallExternalMethod activities, 292
tracking HandleExternalEvent activities, 290-291
tracking workflow level events, 289-290
uploading profiles to database and running, 293
uploading, 301-302
TrackingRecords, 285
TrackingService, 285
transactions, 363
TransactionScope, 344
TransactionScope activity, 365
TransactionWorkflow
modeling, 364-365
performing preliminary setup, 363
running SQL script to create TestTransaction database, 365
TransientWorkflow property, Dynamic Update, 320
transitions, state transitions, 176
transparency, improving in workflows, 57

U
Ul, updating, 177-178
uncommenting, UserTrackingRecords query code, 302-303
Uninitialize method, overriding in basic custom queued activities, 492
UnSubscribe handler, implementing CustomerEventDrivenActivity, 518

UntilCondition event, 355
UntilCondition property, 254
UpdateControls method, 171
updates, dynamic updates, 34-36
uploading, 301-302
host forms, Replicator workflow, 210-212
hosts
CustomerQueuedFromTypedService, 513
for roles, 311-312
to pass in OrderAmount, 465
to use new services (CustomerQueuedFromTypedServiceActivity), 503-504
hosts to run multiple workflows, scheduling service projects, 380
hosts to send data to queues, basic custom queued activities, 492
hosts to use
ManualWorkflowSchedulerService, 381-382
running, 382-383
hosts via configuration, building hosts from scratch, 87-89
local services, escalation workflow, 120-122
member variables, for roles, 312
Replicator workflow, to run in parallel, 215-216
rules dynamically, 327-328
Ul, 177-178
workflow for roles, 312
workflow code-beside file, 127-128
workflow midflight, EventHandlingScope, 231
workflow model for roles, 313
workflows
connecting to WCF endpoints from WF, 445-448
GeneralControlFlow activities, 568
uploading

profiles to database, TrackingProfile, 293
rules to external databases, external RuleSet application, 273-274
TrackingProfiles, 301-302
user tracking data, adding to workflow, 299
users, adding to ASP.NET role providers, 310-311
UserTrackingRecords
adding user tracking data, 299
uncommenting, 302-303

V

Validate method, overriding in GeneralControlFlow activity, 557-558
validating child activities, GeneralControlFlow activity, 559-560
validation
removing GeneralControlFlowBranch validation, 567
testing in GeneralControlFlow activity, 561
Value, 429
variable declarations, adding to CAG projects, 244
ViewQueues method, updating host forms (Replicator workflow), 210-211
Visual Studio 2005, installing, 44
Visual Studio 2008, installing, 45
Visual Studio debugger, configuring to not trap CLR exceptions, 349-350
Visual Studio SharePoint workflow, 40 .vscontent file, 480 .vsi files, 480-481

W

WCA.exe, 233-234
WCF (Windows Communication Foundation), 14, 424
ABCs of WCF, 426
bindings, 426-427
connecting endpoints from WF, 445
console application clients, 434
adding client code to call the service, 436-437
adding service references to WCF hosts, 434-435
generated files, 435
running, 437
ConsoleApplicationWcfWorkflowHost, 443
retrieving WorkflowRuntime, 443-444
running, 444
ContractsAndWorkflows, 427
App.config file, 430-431
modeling workflow, 429
Receive activity, 428-429
reviewing interfaces, 427-428
creating solutions, 425-426
endpoints and hosts, 431
running, 433
starting, pausing and closing, 432-433
WorkflowServiceHost, 432
overview, 421-422
Send activity, configuring, 446-447
WF and, 36-37
integration, 37-38
overview, 422-424
workflows
modifying interfaces, 438
rebuilding client proxies, 441-442
Receive activity, 438-441
updating clients to invoke new methods, 442
WCF hosting, 423
web forms, creating with ASP.NET, 415-416
web services, 398
calling from workflows, 404
creating dependency properties, 404
InvokeWebService activity, 405
modeling workflows, 404-405
running, 406
configuring
IfElse activity, 401
WebServiceImpl activity, 400-401
WebServiceOutput activity, 402-403
creating interfaces to produce WSDL, 399
dependency properties, 398
modeling and publishing workflows, 399
publishing workflows as, 403
Web.config file, 411
WebServiceFault activity, 407-409
WebServiceImpl activity, 18, 398
configuring, 400-401
WebServiceInvokeActivity, WebServiceProxy property, 412
WebServiceOutput activity, 18, 398
configuring, 402-403
WebServiceProxy property, 413
adding code-beside for Replicator activities, 230-231
adding code-beside to update workflow, 231-232
configuring activities in event handlers view, 229
configuring activities in sequential view, 228
preparatory setup, 230
running, 232-233
expense report workflow (sample), 10, 12
human workflow, 12
improving transparency, 57
integration-centric workflow systems, 12
modeling
ContractsAndWorkflows (WCF), 429
  While activity for sequential processing, 213
overview, 10
passing parameters to, 58-60
preparing to use external rules, external RuleSet, 275-276
Replicator workflow, 215
  adding member variables to, 204-205
  configuring activities, 206-210
  placing activities on, 205
  running, 212-213
  updating host forms, 210-212
retrieving from persistence, 135-137
RuleSet, running, 263-264
running
  CustomerEventDrivenActivity, 521-522
  CustomerQueuedFromService Activity, 504
running external rules, external RuleSet, 277
segmentations, 12
sequential workflows, 243
SharePoint workflow, 38
  overview, 38-39
  SharePoint Designer, 41
  Visual Studio, 40
state machine workflow, 243
TransactionWorkflow, 363
updating
  GeneralControlFlow activities, 568
  for roles, 312
WCF
  modifying interfaces, 438
  rebuilding client proxies, 441-442
  Receive activity, 438-441
  updating clients to invoke New method, 442
with CheckCredit activities, running, 467
WorkflowRuntimeService, 375
XAML, 33-34
  examining, 65-66
  XAML + code, running, 66
  XAML + code workflow, 64
  adding and configuring activities, 64-65
  XAML-only workflow, 66-67
  calling from host, 68-69
  modeling and preparing for execution, 67-68
workflow code-beside files, updating, 127-128
workflow designer, 16-18
workflow events, sample application, 372
workflow monitoring
  WorkflowMonitor (sample), 25
workflow projects
  adding activities to, 49-51
  creating, 47-49
  host-workflow data exchange, 95
debbuging, 52-53
efficiency, 53
  adding and removing activities, 54
  adding rule conditions, 54
  configuring Code activities, 55-56
  improving transparency, 57
  passing parameters, 58-60
  pausing the host, 51-52
workflow queuing service, 378-379
workflow styles, 19
  CAG (ConditionedActivityGroup) activity, 22-23
  sequential workflow, 19-20
  state machine workflow, 20-21
workflow-level FaultHandlers, adding, 345-347
WorkflowAborted, 370
WorkflowChanges, Dynamic Update, 320
WorkflowChanges.TransientWorkflow. Activities.Insert method, 322
WorkflowCompleted, 370
WorkflowCompleted handler, 444
WorkflowCreated, 370
WorkflowIdled, 370
WorkflowInstance
  ASP.NET hosting, 416-417
  Dynamic Update, 321
WorkflowLoaded, 371
WorkflowLoaderService, 377
WorkflowMarkupSerializer, 336
WorkflowMonitor SDK sample, running, 141
  downloading and installing WorkflowMonitor, 141-143
WorkflowQueue, 484-485
WorkflowQueuingService, 378-379
WorkflowResumed, 371
WorkflowRoleCollection, 307
WorkflowRuntime, 24, 111
  ASP.NET hosting, 414-415
  retrieving in ConsoleApplicationWcfWorkflow Host, 443-444
WorkflowRuntime events, 370-371
WorkflowRuntimeService, 375
workflows
  ManualWorkflowSchedulerService, 382
parallel workflows, running, 183-184
  building hosts from scratch, 86-87
  TrackingProfile, 293
running with data coming from databases, 464
  simple workflows, creating, 75
  TrackingProfiles, running, 297-298
WorkflowServiceHost, 432, 442
WorkflowsProject, creating 74
WorkflowStarted, 371
WorkflowSuspended, 371
WorkflowTerminated, 371
WorkflowUnloaded, 371
WPF (Windows Presentation Foundation), 14
WSDL, creating interfaces to produce, 399

X

XAML, 33-34
  dynamic update, 36
  EventHandlingScope workflow, 225-226
XAML + code workflow, 64
  activities, adding and configuring, 64-65
  running, 66
XAML workflow, examining, 65-66
XAML-only workflow, 66-67
  calling from host, 68-69
  modeling and preparing for execution, 67-68

Y-Z

YearlySales rule, adding, 262-263

How can we make this index more useful? Email us at indexes@samspublishing.com