From Technologies to Solutions

Programming

Windows Workflow Foundation

Practical WF Techniques and Examples using XAML and C#

A Concise and Practical Guide to Installation, Administration, and Customization

K. Scott Allen

PACKT PUBLISHING

Programming Windows Workflow Foundation: Practical WF Techniques and Examples using XAML and C#

A C# developer's guide to the features and programming interfaces of Windows Workflow Foundation

K. Scott Allen

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

Programming Windows Workflow Foundation: Practical WF Techniques and Examples using XAML and C#

First published: December 2006

Production Reference: 1121206

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 1-904811-21-3

www.packtpub.com

Cover Image by www.visionwt.com

Credits

Author	**Project Manager**
K. Scott Allen	Patricia Weir
Reviewer	**Project Coordinator**
Dan Kahler	Abhijeet Deobhakta
Development Editor	**Indexer**
Douglas Paterson	Bhushan Pangaonkar
Assistant Development Editor	**Proofreader**
Nikhil Bangera	Chris Smith
Technical Editor	**Layouts and Illustrations**
Viraj Joshi	Shantanu Zagade
Editorial Manager	**Cover Designer**
Dipali Chittar	Shantanu Zagade

About the Author

Scott Allen is a software architect with 14 years of experience in commercial software development. Scott has worked on platforms ranging from 8-bit embedded systems to highly scalable and distributed web applications. For the last six years, Scott has concentrated on the .NET platform, and is a Microsoft MVP. Scott presents at local code camps and national conferences.

Scott is a founder of the website `OdeToCode.com`, where he publishes articles and maintains a blog. Scott previously coauthored *Building Websites with the ASP.NET Community Starter Kit* for Packt Publishing, and has also published articles in *MSDN Magazine* and *Dr. Dobb's Journal*.

I'd like to thank Packt's Douglas Paterson for putting up with my antics over the years. I'd also like to acknowledge the rest of the Packt team, including Patricia Weir, Abhijeet Deobhakta, and Viraj Joshi, for their efforts.

Dan Kahler and Nikhil Bangera made this a better book with their diligent reviewing efforts, and I am grateful.

Finally, there is Beaker. While I was writing, and the rest of the world was sleeping, Beaker would be on my desk or curled next to my feet. She'd be sleeping too, but she was always close by until she passed away. I miss you, Beaker.

About the Reviewer

Dan Kahler is a Senior Engineer with Verizon Business. With over eight years of experience developing and administering Windows and Web-based solutions, he specializes in using Microsoft technologies to simplify and automate day-to-day system administration tasks and to integrate line-of-business applications. Dan previously contributed to the *Microsoft Log Parser Toolkit* (Syngress, ISBN: 1-932266-52-6) as a contributing author, and contributed to the *Microsoft Internet Information Services (IIS) 6.0 Resource Kit* (Microsoft Press, ISBN: 0-735614-20-2) as a technical reviewer and tester. He is active in the Baltimore .NET user group (`www.baltomsdn.com`). Dan currently resides in Eldersburg, Maryland with his wife Shannon and children Nicole and Ethan.

Table of Contents

Preface

Windows Workflow Foundation (WF) is a technology for defining, executing, and managing workflows. It is part of the .NET Framework 3.0 and will be available natively in the Windows Vista operating system.

Windows Workflow Foundation might be the most significant piece of middleware to arrive on the Windows platform since COM+ and the Distributed Transaction Coordinator. The difference is, not every application needs a distributed transaction, but nearly every application does have a workflow encoded inside.

This book will help you add that workflow power to your applications.

What This Book Covers

Chapter 1 introduces us to the concept of workflow and describes how Windows Workflow can solve the difficult problems inherent in workflow solutions. We'll become familiar with **activities** as the basic building blocks of a workflow definition and demonstrate how to author a simple workflow using Visual Studio 2005. This chapter also describes the runtime services available with WF. By the end of the chapter we will be able to identify the primary features of Windows Workflow.

Chapter 2 concentrates on authoring workflows. Specifically, we'll look at how to build workflows with C#, and with extensible application markup language (**XAML**). Looking at the workflow compiler, we'll have a better understanding of how WF uses code generation to produce classes from workflow markup, and how this generated code can combine with our hand-written code to produce a workflow type. This chapter will provide the fundamental knowledge needed to understand how WF operates during the compilation phase.

In *Chapter 3*, we will turn our attention to sequential workflows. We will examine the `SequenceActivity` and learn about the events fired by the workflow runtime during the life of a workflow instance. Using Visual Studio, we will build workflows that accept parameters and communicate with a host process by invoking methods and listening for events. The chapter concludes with a workflow example that raises an exception and uses a fault handler.

Chapter 4 examines each activity in the WF base activity library. We will look at the control flow activities, communication activities, and transaction-oriented activities. The chapter also examines web service activities, rule-centric activities, and state activities. The goal of this chapter is to make us aware of all the capabilities provided by the base activity library, with an eye towards understanding how each activity can solve a particular problem.

With an understanding of what is available in the base activity library, we can look at building our own custom activities in *Chapter 5*. This chapter examines the motivations for building custom activities, and provides examples of building a custom activity using both a compositional approach and a derivation approach. We'll see how to build a custom validator and designer for our activity, and also understand the advantages of using dependency properties. The chapter ends by covering the execution context, which we must understand to build robust activities.

Chapter 6 covers the workflow runtime, workflow diagnostics, and the out-of-the-box services provided for WF by Microsoft. The chapter demonstrates how to configure services both declaratively and programmatically. We'll see examples of how to use a scheduling service, persistence service, and tracking service. The chapter provides enough information to allow a developer to select and configure the services needed for a wide variety of scenarios and environments.

Chapter 7 focuses on building event-driven workflows using state machines. We'll see how WF models the traditional state machine using activities, and we will build a workflow to handle external events and react with state transitions. We'll also see how to track and examine the history of state machine execution. The chapter ends with an examination of a hierarchical state machine, which provides all the knowledge we need to tackle tough problems with event-driven workflows.

Chapter 8 is dedicated to workflow communications. The chapter explains how to use correlated local services for communication with a host process, and web service activities for communication across a network. By the end of the chapter we'll uncover the queuing service that is used behind the scenes of a workflow to coordinate and deliver messages.

Finally, *Chapter 9* is about rules and conditions in Windows Workflow. This discusses the role of business rules in software development and provides examples of how WF's rules engine can take away some of the burden of rule development. The chapter takes an in-depth look at rule execution in the `PolicyActivity`, and recording diagnostic information about rule evaluation. We'll come away with the knowledge we need to build rule-based solutions using Windows Workflow.

What You Need for This Book

Windows Workflow Foundation is one part of the .NET 3.0 framework. To run Windows Workflow, you'll need to download and install the .NET 3.0 redistributable (see the links below):

.NET 3.0 (x86): `http://go.microsoft.com/fwlink/?LinkID=70848`

.NET 3.0 (x64): `http://go.microsoft.com/fwlink/?LinkID=70849`

Visual Studio 2005 extensions for .NET Framework 3.0 (Windows Workflow Foundation):

```
http://www.microsoft.com/downloads/details.aspx?FamilyId=
5D61409E-1FA3-48CF-8023-E8F38E709BA6&displaylang=en
```

The .NET 3.0 runtime requires Windows Server 2003 SP1, Windows XP SP2, or Windows Vista. To develop Windows Workflow solutions you'll need to download the Visual Studio 2005 extensions for .NET Framework 3.0.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

There are three styles for code. Code words in text are shown as follows:

"The `codeActivity1_ExecuteCode` method is here and waiting for us to provide an implementation"

A block of code will be set as follows:

```
using System;
using System.Workflow.Activities;

namespace chapter2_library
{
  public sealed partial class PureCode: SequentialWorkflowActivity
  {
    public PureCode()
    {
      InitializeComponent();
    }

  }
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
using System;
using System.Workflow.Activities;

namespace chapter2_library
{
  public sealed partial class PureCode: SequentialWorkflowActivity
  {
    public PureCode()
    {
      InitializeComponent();
    }
  }
}
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "Right-click the workflow and select the **Delete** option".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to feedback@packtpub.com, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or email suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the Example Code for the Book

Visit `http://www.packtpub.com/support`, and select this book from the list of titles to download any example code or extra resources for this book. The files available for download will then be displayed.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **Submit Errata** link, and entering the details of your errata. Once your errata have been verified, your submission will be accepted and the errata added to the list of existing errata. The existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with some aspect of the book, and we will do our best to address it.

1 Hello, Workflow

…thoughts arrive like butterflies — Gossard / Vedder

Windows Workflow might be the most significant piece of middleware to arrive on the Windows platform since COM+ and the Distributed Transaction Coordinator. The difference is, not every application needs a distributed transaction, but nearly every application does have a workflow encoded inside. To understand the types of problems Windows Workflow is designed to solve, let's talk about workflow in a generic sense.

What is a workflow? A simple definition would say a workflow is the series of steps, decisions, and rules needed to complete a specific task. Think of the workflow that takes place when you order food at the local pizza shop. You tell the cashier the type of pizza you want. The cashier passes this information to the cook, who gathers ingredients and puts a pizza in the oven. The cook hands a finished pizza to the cashier, who collects payment and completes the workflow by handing over your pizza. The work *flows*, to the cashier, then to the cook, and then back again.

During each of these steps, all parties are also evaluating rules and making decisions. Before accepting the order, the cook has to compare the order against the ingredients in stock. The cashier has to validate and process any coupons you might present, and notify the manager if you pay with a counterfeit looking bill.

Not every workflow has to involve humans (which is good, because humans complicate even the simplest process). A workflow can take place between two distributed software applications. For example, two content management applications might need to follow a specific set of steps and rules when synchronizing content in the middle of the night.

Most workflows are stateful, and often run for a relatively long time. Hopefully, your pizza will be ready within 30 minutes. During those 30 minutes, state information about your order, like the toppings you selected, has to be available. A different workflow happens when the pizza shop orders cheese. The cheese supplier might not deliver the mozzarella for 30 hours, and the pizza shop may not pay the cheese supplier for 30 days. During those 30 days, something needs to maintain the state of the workflow for a purchase.

A workflow may spend a large portion of its lifetime waiting for events to happen in the world around it. A workflow may be idle when waiting for a delivery, or waiting for a payment, or waiting for a pizza to finish in the oven. During these wait times, the workflow is idle and no resources are required by the workflow.

A workflow, then, is a series of steps to finish a task. A workflow is often long running and stateful, and often needs to wait on events and interact with humans. You can see workflows everywhere you look in the world. As software developers, we often have to codify the workflows around us into software applications.

Building Workflow Solutions

We've all been involved with software projects that try to improve a business process. The process might have involved pizza orders, or financial transactions, or health care. Invariably, the word *workflow* will arise as we talk about these projects. While the workflow might sound simple, we know the devil is always in the details. We'll need database tables and data access classes to manage the workflow state. We'll need components to send emails and components to wait for messages to arrive in a queue. We will also need to express the workflow itself for the computer to execute. Let's look at a theoretical implementation of a workflow:

```
// The workflow for a newly submitted Purchase Order
class PurchaseOrderWorkflow
{
   public void Execute(PurchaseOrder order)
   {
      WaitForManagerApproval(order);
      NotifyPurchaseManager(order);
      WaitForGoods(order);
   }

   …

}
```

Assuming we have definitions for the three methods inside of `Execute`, can a workflow really look this simple? The answer is no. We'll have to add code for exception handling, logging, and diagnostics. We'll need to raise events and provide hooks to track and cancel a running workflow. Also, this workflow will be idle and waiting for an external event to occur, like the arrival of the purchased goods, for the majority of the time. We can't expect to block a running application thread for days or weeks while waiting for a delivery. We'll need to provide a mechanism to save the workflow's state of execution to a persistent data store and remove the running workflow instance from memory. When a significant event occurs, we'll need to restore the workflow state and resume execution.

Unfortunately, we will have so much code in and around the workflow that we will lose sight of the workflow itself. All the supporting code will hide the process we are trying to model. A non-technical businessperson will never be able to look at the code and see the workflow. A developer will need to dig through the code to find the workflow inside.

An improved workflow design will try to separate the definition of a workflow from the engine and supporting code that executes the workflow. This type of approach allows a developer, or even a businessperson, to express *what* the workflow should be, while the workflow engine takes care of *how* to execute the workflow. These days, many workflow solutions define workflows inside the loving embrace of angled brackets. Let's look at some theoretical XML for a workflow definition:

```
<Workflow Name="PurchaseOrderWorkflow">
   <Steps>
         <WaitForTask Event="ManagerApproval"/>
         <NotifyTask Target="PurchaseManager"/>
         <WaitForTask Event="Delivery"/>
   </Steps>
   <Parameters>
         <Parameter Type="PurchaseOrder" Name="order"/>
   </Parameters>
</Workflow>
```

Let's ask the question again — can a workflow really look this simple? The answer is yes; what we will need is a workflow engine that understands this XML, and can transform the XML into instructions for the computer. The engine will include all the required features like exception handling, tracking, and enabling cancellations.

The C# code we saw earlier is an example of **imperative** programming. With imperative programming, we describe *how* to perform a task by providing a series of instructions to execute. The XML markup above is an example of **declarative** programming. With declarative programming, we describe *what* the task looks like, and let other software determine the steps required to complete the task. Most of the commercial workflow solutions on the market allow a declarative definition of workflow, because the declarative approach doesn't become cluttered with exception handling, event raising, and other lower-level details.

One of the benefits to using XML is the large number of tools with the ability to read, modify, create, and transform XML. XML is tool-able. Compared to parsing C# code, it would be relatively easy to parse the XML and generate a visualization of the workflow using blocks and arrows. Conversely, we could let a business user connect blocks together in a visual designer, and generate XML from a diagram.

Let's think about what we want in a workflow solution. We want to specify workflows in a declarative manner, perhaps with the aid of a visual designer. We want to feed workflow definitions into a workflow engine. The engine will manage errors, events, tracking, activation, and de-activation.

Enter Windows Workflow Foundation.

A Windows Workflow Tour

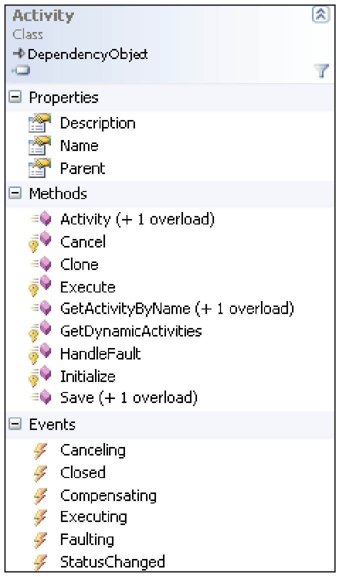
Microsoft's Windows Workflow Foundation is one piece of the new .NET 3.0 platform. The other major additions in .NET 3.0 include Windows Presentation Foundation, or WPF, and Windows Communication Foundation, or WCF. Microsoft will support Windows Workflow (WF) on Windows XP, Windows Server 2003, and Windows Vista.

Support for current and future Microsoft platforms means WF could reach near ubiquity over time. We can use WF in smart client applications, and in simple console-mode programs. We can also use WF in server-side applications, including Windows services, and ASP.NET web applications and web services. WF will make an appearance in several of Microsoft's own products, including Windows SharePoint Services and Microsoft Biztalk Server. We will now look at an overview of the essential features of Windows Workflow.

Activities

The primary building block in Windows Workflow is the **activity**. Activities compose the steps, or tasks in a workflow, and define the workflow. We can arrange activities into a hierarchy and feed activities to the workflow engine as instructions to execute. The activities can direct workflows involving both software and humans.

All activities in WF derive from an `Activity` base class. The `Activity` class defines operations common to all activities in a workflow, like `Execute` and `Cancel`. The class also defines common properties, like `Name` and `Parent`, as well as common events like `Executing` and `Closed` (the `Closed` event fires when an Activity is finished executing). The screenshot below shows the **Activity** class in the Visual Studio 2005 class designer:


Activity
Class
DependencyObject
Properties
Description
Name
Parent
Methods
Activity (+ 1 overload)
Cancel
Clone
Execute
GetActivityByName (+ 1 overload)
GetDynamicActivities
HandleFault
Initialize
Save (+ 1 overload)
Events
Canceling
Closed
Compensating
Executing
Faulting
StatusChanged

WF ships with a set of ready-made activities in the **base activity library**. The primitive activities in the library provide a foundation to build upon, and include control-flow operations, like the `IfElseActivity` and the `WhileActivity`. The base activity library also includes activities to wait for events, to invoke web services, to execute a rules engine, and more.

Custom Activities

Windows Workflow allows developers to extend the functionality of the base activity library by creating custom activities to solve problems in their specific domain. For instance, pizza delivery workflows could benefit from custom activities like `SendOrderToKitchen` or `NotifyCustomer`.

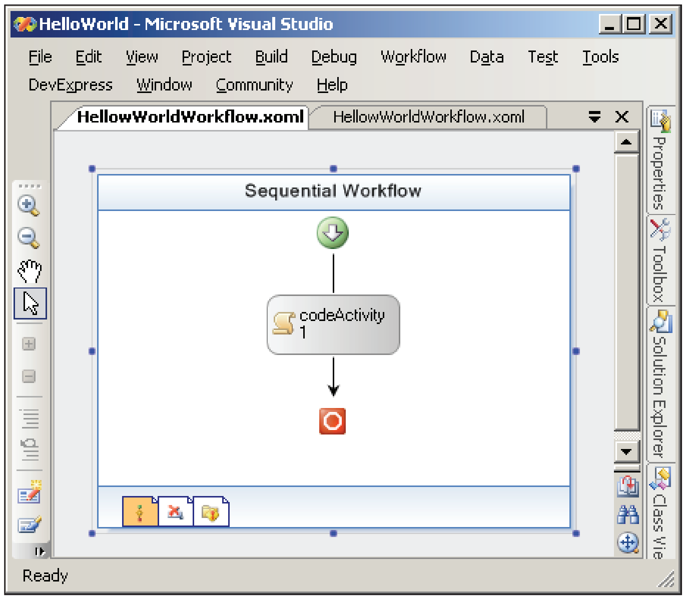
All custom activities will also ultimately derive from the base `Activity` class. The workflow engine makes no special distinction between activities written by Microsoft and custom activities written by third parties.

We can use custom activities to create domain-specific languages for building workflow solutions. A domain-specific language can greatly simplify a problem space. For instance, a `SendOrderToKitchen` custom activity could encapsulate a web service call and other processing logic inside. This activity is obviously specific to the restaurant problem domain. A developer will be more productive working with this higher-level abstraction than with the primitive activities in the base activity library. Even a restaurant manager will understand `SendOrderToKitchen` and might arrange the activity in a visual workflow designer. It will be difficult to find a restaurant manger who feels comfortable arranging `WhileActivity` and `InvokeWebServiceActivity` objects in a workflow designer.

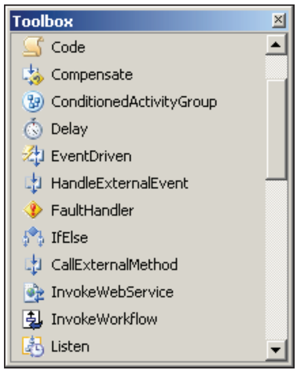
C#, VB.NET, and XML are *general-purpose* languages and have the ability to solve a wide array of different problems. We can use C# to develop solutions for pizza restaurants as well as hospitals, and the language works equally well in either domain. A *domain-specific* language excels at solving problems in a particular area. A domain-specific language for restaurant workflow would boost productivity when writing software for a restaurant, but would not be as effective when writing software for a hospital.

Visual Studio 2005 Extensions

Microsoft also provides the Microsoft Visual Studio 2005 Extensions for Windows Workflow. These extensions plug into Visual Studio to provide a number of features, including a visual designer for constructing workflows. A screenshot of the visual designer is shown on the next page.



The designer uses the same windows we've come to love as Windows and web form developers. The **Toolbox** window will list the activities available to drag onto the design surface. We can add our own custom activities to the **Toolbox**. Once an activity is on the design surface, the **Properties** window will list the activity's properties that we can configure, and the events we can handle. The **Toolbox** window is shown below:



Windows Workflow and XAML

The WF designer can generate C# and Visual Basic code to represent our workflow. The designer can also read and write eXtensible Application Markup Language (XAML, pronounced zammel). XAML files are valid XML files. XAML