



Sample Workflow Project

The sample project provided in the appendix is designed as a review of many of the concepts discussed in this book. It also demonstrates how the techniques you've learned can be combined to build a full-featured application. Instead of giving you step-by-step instructions for building the solution, the complete project is available for you to download from <http://www.apress.com>. As you'll see, this project is fairly extensive and this will save you quite a bit of time (and typing).

Project Overview

The application provides a web page in which end users can enter requests, comments, or questions. Based on the topic selected, the request is placed in one of several queues for individuals to view and respond to. A second web page is provided to show the contents of these queues. Once a queue is selected, items are presented to be worked. Each queue can be configured in one of two modes: either the oldest request is automatically presented to be worked or all requests are displayed for the user to select one.

After a request has been worked, it might require a QC review step based on rules defined in a Policy object. (The implementation is very similar to one you created in Chapter 20.) Requests can also be rerouted to a different queue if necessary. A tracking extension is used to record the various events (started, assigned, reviewed, rerouted, and so on) so you can see how a particular request made its way through the workflow.

All the workflow functionality is provided through a web service. The web site uses the native .Net membership services, which gives the site the capability to "log in." The operator information is provided to the workflow so it can track which users worked on each request. Finally, the generic workflow features and queue logic is provided by a set of workflow activities and extensions that are implemented in a separate library project. This allows you to reuse this code in other applications.

I'll explain the solution in more detail later, but first, let's run the application so you can see what it does. You'll need to download the Appendix.zip file and extract this to an appropriate location.

Configuring the Database

The AppendixData folder contains the scripts you'll need to set up the database schema. First, create a SQL Server database for this solution. Expand the Create Scripts folder and run the included scripts in the following order:

- `SqlWorkflowInstanceStoreSchema.sql`
- `SqlWorkflowInstanceStoreLogic.sql`
- `Config.sql`
- `Request.sql`
- `Tracking.sql`

The scripts used to create the membership tables and procedures assume that there is a database named `aspnetdb` on the server that the connection string is referencing. This is the default database that the ASP.Net services use. If you do not already have this setup, create a database called `aspnetdb` and then run the `InstallCommon.sql` and `InstallMembership.sql` scripts (in that order).

■ **Caution** If you need to modify the database connection for your environment, make sure that you make the change to the configuration files; there are two places. The `web.config` file in the root folder of the web site project defines the connection string for the `aspnetdb` database used by the .Net services. There is also a `web.config` file in the root folder of the `ServiceLayer` project. This is used to configure the workflow persistence store as well as the application data that your custom activities will use.

Running the Application

Once your database has been configured, from Visual Studio, press F5 to start the application. The initial page should look like the one shown in Figure A-1.



Figure A-1. Initial web page

Logging In

Click the Log In link at the top-right corner of the page. The first time you log in, you'll need to create a new account by clicking the register link. This will display the page shown in Figure A-2.

The screenshot shows a web application interface with a header bar containing the title 'Beginning WF 4.0 - Sample Application' and a '[Log In]' link. Below the header is a navigation bar with buttons for 'Home', 'About', 'Submit', and 'Process'. The main content area is titled 'CREATE A NEW ACCOUNT' and includes the instruction 'Use the form below to create a new account.' and a note 'Passwords are required to be a minimum of 6 characters in length.' The form is titled 'Account Information' and contains four input fields: 'User Name:', 'E-mail:', 'Password:', and 'Confirm Password:'. A 'Create User' button is located at the bottom right of the form.

Figure A-2. Creating a new account

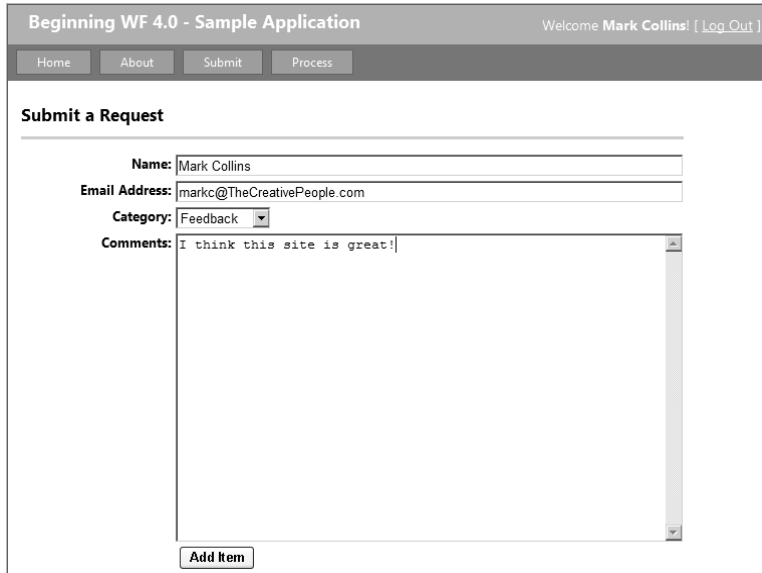
Enter your name, e-mail address, and a password (the e-mail address is not actually used to send e-mails so you can enter any text here that you want.) The next time you log in, you'll want to check the “Keep me logged in” check box, as shown in Figure A-3. If you do, you won't need to log in again even if you restart the application.

The screenshot shows the 'LOG IN' page of the same web application. The header and navigation bar are identical to Figure A-2. The main content area is titled 'LOG IN' and includes the instruction 'Please enter your username and password. [Register](#) if you don't have an account.' The form is titled 'Account Information' and contains three input fields: 'Username:' (with the text 'Mark Collins' entered), 'Password:' (with masked characters '••••••'), and a 'Keep me logged in' checkbox which is checked. A 'Log In' button is located at the bottom right of the form.

Figure A-3. Using the “Keep me logged in” option

Submitting a Request

Click the Submit link at the top of the page. This will display the page used for entering a new request/comment, which is shown in Figure A-4.



The screenshot shows a web application interface titled "Beginning WF 4.0 - Sample Application". In the top right corner, it says "Welcome Mark Collins! [Log Out]". Below this is a navigation bar with buttons for "Home", "About", "Submit", and "Process". The main content area is titled "Submit a Request". It contains a form with the following fields: "Name:" with the value "Mark Collins", "Email Address:" with the value "markc@TheCreativePeople.com", "Category:" with a dropdown menu showing "Feedback", and "Comments:" with a text area containing "I think this site is great!". At the bottom of the form is a button labeled "Add Item".

Figure A-4. Entering a new request

If you've logged in, the name and e-mail address will be filled in for you. Select the Feedback category and enter a comment. Then click the Add Item link. Figure A-5 shows a completed submit page. The comment field is cleared and the unique identifier assigned to this request is displayed at the top of the page. Enter a couple more requests using the same Feedback category.

Beginning WF 4.0 - Sample Application Welcome Mark Collins! [[Log Out](#)]

Home About Submit Process

Submit a Request

Request d87c5513-1ccb-4352-927d-d34b23f24d72 was submitted

Name:

Email Address:

Category: ▼

Comments:

Figure A-5. The completed submit page

Processing Requests

Click the Process link at the top of the page. This will display the Process page that is shown in Figure A-6.

Beginning WF 4.0 - Sample Application Welcome Mark Collins! [[Log Out](#)]

Home About Submit Process

Process a Request

Select which Queue to work

Available Queues				
	Queue	QC	# Requests	Oldest
Select	Marketing	False	3.00	1/15/2010

Figure A-6. Displaying the available queues

The grid on this page lists the queues that have requests that need to be responded to. The Feedback category was assigned to the Marketing queue so all the requests you entered are in this queue. This grid indicates how many requests are in this queue and the date of the oldest request. Click the Select link and the page should look like the one shown in Figure A-7.

Beginning WF 4.0 - Sample Application

Welcome Mark Collins! [Log Out]

Home

About

Submit

Process

Process a Request

Select which Queue to work

Available Queues

	Queue	QC	# Requests	Oldest
Select	Marketing	False	3.00	1/15/2010

Select one of the following Requests

	Type	Name	eMail	Created
Select	Feedback	Mark Collins	markc@TheCreativePeople.com	1/15/2010
Select	Feedback	Mark Collins	markc@TheCreativePeople.com	1/15/2010
Select	Feedback	Mark Collins	markc@TheCreativePeople.com	1/15/2010

Figure A-7. Selecting a request

The Marketing queue is configured to allow selection. This means that instead of automatically assigning the oldest request in the queue, all the requests are listed, and you are allowed to select the one you want to work on. Click one of the Select links and the request will be displayed, as shown in Figure A-8.

Process this Request :

Queue Type:

Feedback

Comment:

I think this site is great!

Name:

Mark Collins

Email:

markc@TheCreativePeople.com

Action Taken:

Sent to Product queue - send him a free product.

Route Next:

Product

Complete

Cancel

Figure A-8. Responding to a request

Below the request there is a place to enter the action that was taken. You can also select a queue to route this request to. You could click the Cancel button if you decided not to work on this request. That will unassign the request and put it back in the queue for someone else to work. Enter some notes in the Action Taken field, select the Product queue, and click the Complete button. The page should now look like the one shown in Figure A-9.

Beginning WF 4.0 - Sample Application Welcome Mark Collins! [[Log Out](#)]

Home About Submit Process

Process a Request

Select which Queue to work

Available Queues				
	Queue	QC	# Requests	Oldest
Select	Marketing	False	2.00	1/16/2010
Select	Product	False	1.00	1/16/2010

Select one of the following Requests

	Type	Name	eMail	Created
Select	Feedback	Mark Collins	markc@TheCreativePeople.com	1/16/2010
Select	Feedback	Mark Collins	markc@TheCreativePeople.com	1/16/2010

Figure A-9. The updated queue list

Notice that the Product queue is now displayed in the queue list, and there are only two requests to choose from in the Marketing queue. Click the Select link next to the Product queue. The page should look like the one shown in Figure A-10.

Process this Request :

Queue Type: Feedback

Comment:

Name: Mark Collins

Email: markc@TheCreativePeople.com

Action Taken:

Route Next:

Figure A-10. Working the request in the Product queue

Because there is only one item in the Product queue, it was automatically assigned. You can edit or append to the Action Taken field. You can also reroute this request to another queue, if necessary. For this request, append a note to the Action Taken field and leave the Route Next field blank. Click the Complete button. The page should look like the one shown in Figure A-11.

Beginning WF 4.0 - Sample Application

Welcome Mark Collins! [Log Out]

Home

About

Submit

Process

Process a Request

Select which Queue to work

Available Queues

	Queue	QC	# Requests	Oldest
Select	Marketing	False	2.00	1/16/2010
Select	Product	True	1.00	1/16/2010

Figure A-11. Request moved to QC mode

Notice that this request is still in the Product queue, but the QC column shows “True”. The Product queue is configured so that all requests in the Product queue must go through a QC review. Consequently, the request is put back into the queue in QC mode. Select the Product queue again and the request should be displayed; this time in QC mode, as shown in Figure A-12.

Process this Request : QC Review

Queue Type: Feedback

Comment:

I think this site is great!

Name:

Mark Collins

Email:

markc@TheCreativePeople.com

Action Taken:

Sent to Product queue - send him a free product. [Sent book xyz]

Route Next:

None

Complete

Cancel

Figure A-12. Performing QC review

In QC review, you can view and modify the Action Taken. Edit this field and click the Complete button. The request is now complete. Figure A-13 shows the updated queue list with only the remaining items in the Marketing queue.

Queue	QC	# Requests	Oldest
Marketing	False	2.00	1/16/2010

Figure A-13. Updated queue list

Tracking the Workflow

Close the web application and go back to Visual Studio. From the Server Explorer, open the contents of the QueueTrack table. The results should be similar to those shown in Figure A-14.

	QueueName	SubQueueN...	QueueInsta...	EventType	QC	OperatorKey	EventDate
►	Request	Marketing	efcbab4c-4b...	Start	False	NULL	1/9/2010 5:...
	Request	Marketing	efcbab4c-4b...	Assign	False	b4a8b43c-e...	1/9/2010 5:...
	Request	Product	efcbab4c-4b...	Route	False	NULL	1/9/2010 5:...
	Request	Product	efcbab4c-4b...	Assign	False	b4a8b43c-e...	1/9/2010 5:...
	Request	Product	efcbab4c-4b...	QC	False	NULL	1/9/2010 5:...
	Request	Product	efcbab4c-4b...	Assign	True	b4a8b43c-e...	1/9/2010 5:...
	Request	Marketing	3c97677a-c...	Start	False	NULL	1/9/2010 5:...
	Request	Marketing	4c41ce42-f8...	Start	False	NULL	1/9/2010 5:...

Figure A-14. Showing the tracking results

The QueueTrack table records the various events that occurred on each of the requests: Started, Assigned, Route, QC, and so on. The request that you just worked was started in the Marketing queue. It was then assigned to an operator and then routed to the Product queue. It was again assigned to the same operator and then put into QC mode. Finally, it was assigned to the operator for a third time this time in QC mode.

Generic Queue Logic

Using queues for managing human tasks is a common practice that can be used in many applications. I designed this solution to encapsulate the generic activities in a separate project called UserTasks. This should help you to reuse this logic more easily in your own applications.

Database Design

Figure A-15 shows the tables used by the UserTasks project. You can also view this diagram by opening the UserTasks.dbml file in the UserTasks project.

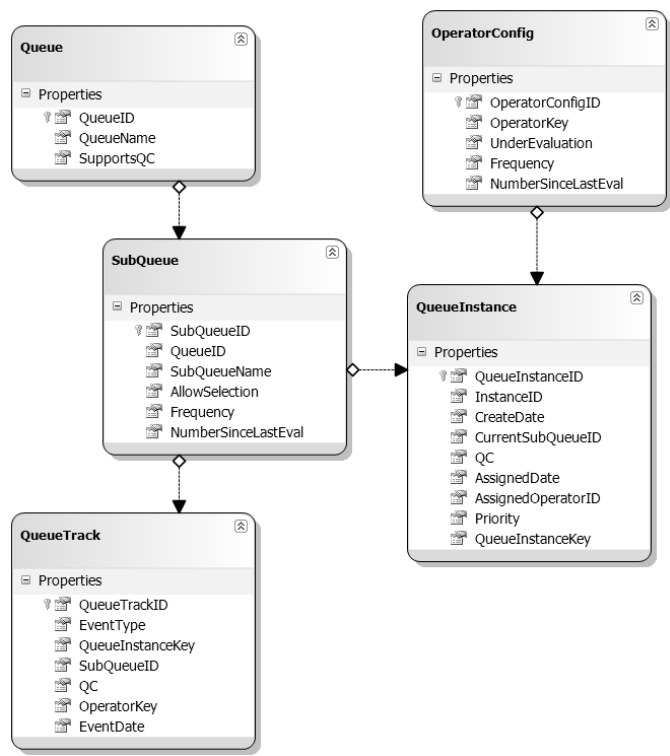


Figure A-15. Database design for the UserTasks project

The database contains both a Queue and a SubQueue table. The queues that you used, such as Marketing and Product, are really subqueues, and this solution uses a single queue called Request. This approach allows you to reuse the same tables (and workflow activities) for any number of human-centric workflow tasks.

The Queue and SubQueue tables provide configuration options such as SupportsQC (at the queue level) and AllowSelection (at the subqueue level). The SubQueue table includes the Frequency column, which defines how often requests in this subqueue need to be reviewed. The NumberSinceLastEval is used to keep track of this to know when it's time to force another review. The OperatorConfig table provides other QC-related options (see Chapter 20 for more details.)

The QueueInstance table is the main table that drives the queue logic. A record is created for every request. It keeps track of what subqueue the request is currently in, whether it's in QC mode, and who it is currently assigned to. The QueueTrack table is populated by the tracking extension in response to user-defined tracking events.

Activities

The UserTasks project is an activity library project that provides activities that can be dropped onto other workflows. The included activities are listed in Table A-1.

Table A-1. Activities provided in the UserTasks project

Activity Name	Description
AssignQueue	Moves a QueueInstance to the specified subqueue.
AssignQueueInstance	Assigns a QueueInstance to the specified operator.
CompleteInstance	Provides QC and rerouting logic.
CreateQueueInstance	Creates a new QueueInstance record.
GetQueueInstances	Returns the available QueueInstance records in the specified subqueue (if allow selection is turned off, it returns only the oldest record).
LoadQueueInstance	Loads the specified QueueInstance.
LookupQueueStats	Returns the number of records in each queue/subqueue.
RequestQC	Moves the specified QueueInstance into QC mode.
UnAssignQueueInstance	Unassigns the current operator and makes this QueueInstance available to be assigned to another operator.

All these activities, with the exception of CompleteInstance, are implemented as coded activities. I won't take the time to list all the source code here; you can browse the code in Visual Studio. They use a DBConnection extension to obtain the connections string (see Chapter 12) and use a PersistQueueInstance extension to save the changes as part of the workflow persistence (see Chapter 15). As with all the other projects in this book, they use LINQ to SQL to access the database tables.

CompleteInstance

The CompleteInstance activity is provided as a designed activity. The workflow design for it is shown in Figure A-16.



Figure A-16. Design of the `CompleteInstance` activity

After loading the `QueueInstance` record, it then checks to see whether this queue supports QC or whether the `QueueInstance` is already in QC mode. If not, it invokes the `QCPolicy` activity, which determines whether this record should be QC'ed. If QC is required, the `RequestQC` activity is executed to update the `QueueInstance`. The `Complete` output argument is passed back to the calling workflow to indicate whether the `QueueInstance` was actually completed or whether it requires a QC step.

QCPolicy

The `QCPolicy` activity is an Interop activity that invokes a `QCPolicy` activity, which is implemented in .Net 3.5. (You might want to refer to Chapter 20 for more details on using the Policy activity.) The design of the `QCPolicy` activity is shown in Figure A-17.

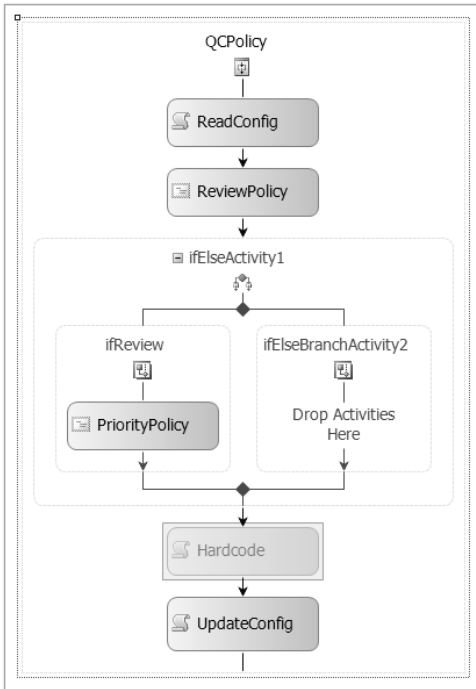


Figure A-17. QCPolicy Design (.Net 3.5)

Unlike the implementation in Chapter 20, this QCPolicy activity reads and updates the configuration data from the database. Because the Policy activity uses the .Net 3.5 version of workflow, it cannot access the DBConnection extension. Instead, the connection string is passed in as a DependencyProperty.

The ReviewPolicy activity is a Policy that determines whether the QueueInstance needs to be reviewed. The rule set used is shown in Figure A-18.

Name	Priority	Reeval...	Active	Rule Preview
No Configuration	5	Always	True	IF this._queue == null THEN this.Review = False Halt
ActivityFrequency	4	Always	True	IF this._queue.NumberSinceLastEval >= this._queue.Frequency THEN this.Review = True...
No Operator	3	Always	True	IF this._operator == null THEN this.Review = False Halt
Under Evaluation	2	Always	True	IF this._operator.UnderEvaluation == True THEN this.Review = True Halt
Operator Frequency	1	Always	True	IF this._operator.NumberSinceLastEval >= this._operator.Frequency THEN this.Review = ...

Figure A-18. ReviewPolicy rule set

If QC is needed, the PriorityPolicy is executed to determine what priority it should be given. Its rule set is shown in Figure A-19.

Name	Priority	Reevaluati...	Active	Rule Preview
Marketing	2	Always	True	IF this SubQueueName == "Marketing" THEN this Priority = 1
Evaluation	1	Always	True	IF this _operator != null && this _operator.UnderEvaluation == True THEN this.Priority = 10

Figure A-19. PriorityPolicy rule set

The Hardcode activity is used for testing only and is normally disabled. It overrides the Review and Priority properties that were set by the Policy activities. This was provided to make it easier to test both QC and non-QC scenarios.

■ **Note** WF 3.0/3.5 provided the ability to disable an activity. When the workflow is executed, disabled activities are ignored. This feature had limited usefulness and is *not* provided in WF 4.0. This particular scenario was one of the useful applications of this feature.

To test the actual Policy implementation, a separate TestQC application is provided. This is a simple workflow application with the QCPolicy activity dropped onto it.

Tracking

The ability to track workflow events was described in Chapter 13. This project relies on custom tracking events. The following code (or something similar) is included in several of the custom activities:

```
CustomTrackingRecord userRecord = new CustomTrackingRecord("Route")
{
    Data =
    {
        {"QueueInstanceKey", qi.QueueInstanceKey},
        {"SubQueueID", qi.CurrentSubQueueID}
    }
};

// Emit the custom tracking record
context.Track(userRecord);
```

This causes a custom tracking event to be generated, which is received and processed by the QueueTracking extension. The implementation of the Track method of this extension is shown in Listing A-1.

Listing A-1. Implementation of the Track Method

```

protected override void Track(TrackingRecord record, TimeSpan timeout)
{
    CustomTrackingRecord customTrackingRecord =
        record as CustomTrackingRecord;

    if (customTrackingRecord != null)
    {
        if (customTrackingRecord.Name == "Start" ||
            customTrackingRecord.Name == "Route" ||
            customTrackingRecord.Name == "Assign" ||
            customTrackingRecord.Name == "UnAssign" ||
            customTrackingRecord.Name == "QC")
        {
            QueueTrack t = new QueueTrack();

            // Extract all the user data
            if ((customTrackingRecord != null) &&
                (customTrackingRecord.Data.Count > 0))
            {
                foreach (string key in customTrackingRecord.Data.Keys)
                {
                    switch (key)
                    {
                        case "QueueInstanceKey":
                            if (customTrackingRecord.Data[key] != null)
                                t.QueueInstanceKey =
                                    (Guid)customTrackingRecord.Data[key];
                            break;
                        case "SubQueueID":
                            if (customTrackingRecord.Data[key] != null)
                                t.SubQueueID = (int)customTrackingRecord.Data[key];
                            break;
                        case "QC":
                            if (customTrackingRecord.Data[key] != null)
                                t.QC = (bool)customTrackingRecord.Data[key];
                            break;
                        case "OperatorKey":
                            if (customTrackingRecord.Data[key] != null)
                                t.OperatorKey =
                                    (Guid)customTrackingRecord.Data[key];
                            break;
                    }
                }
            }

            if (t.SubQueueID != null && t.QC == null)
                t.QC = false;
        }
    }
}

```

```

        t.EventType = customTrackingRecord.Name;
        t.EventDate = DateTime.UtcNow;

        // Insert a record into the TrackUser table
        UserTasksDataContext dc =
            new UserTasksDataContext(_connectionString);
        dc.QueueTracks.InsertOnSubmit(t);
        dc.SubmitChanges();
    }
}

```

This first checks to see whether this is one of the events that should be processed. Specifically, it is looking for the following:

- Start: A new QueueInstance is created.
- Route: A QueueInstance is placed in a queue.
- Assign: The QueueInstance is assigned to a specific operator.
- UnAssign: The QueueInstance is unassigned.
- QC: The QueueInstance is placed to QC mode.

■ **Note** As I demonstrated in Chapter 13, the events that should be tracked can be configured instead of written in code. This is the preferred approach. However, this tracking extension is writing to a database table specifically designed for these events. It doesn't know how to track other events. In this case, it is appropriate to put the filter in code. This still allows you to configure the events that are actually tracked within this set of supported events.

The various data records are extracted from the event to populate the corresponding columns in the table.

Service Layer

As I mentioned previously, all the workflow functionality is provided by a web service. You created a fairly simple web service in Chapter 10. This implementation is significantly more complex.

Service Contract

In Chapter 10, you used both the traditional method of defining a service contract and a declarative style provided by WF. In this project, I used the later method exclusively. On each of the Receive activities, the OperationName and appropriate input parameters are defined. Similarly, on the SendReply activities the output parameters are defined.

Also, on the Receive activities, the interface name is specified. You don't have to define this interface; just specify the name to use when one is created for you. If you want all the methods on the same interface, use the same interface name on all the Receive activities. You can also use different names, which will result in multiple interfaces. For this project, I used `IProcessRequest` for all the methods. Figure A-20 shows the methods that are implemented in the `IProcessRequest` interface.

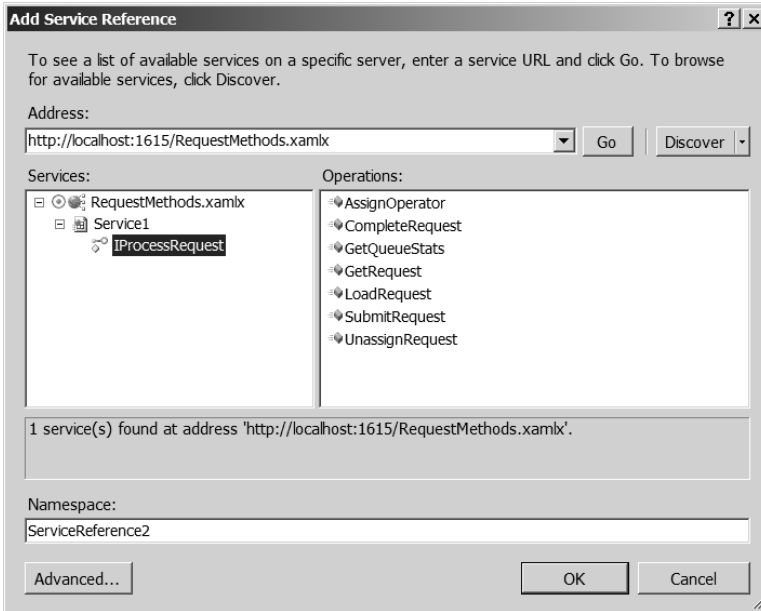


Figure A-20. Web service methods

When a service reference is added to the web application, all the necessary web service details are generated for you, including the Web Service Definition Language (.wsdl) file that explicitly defines the web service and the methods provided. You can open `Service1.wsdl` and see what this looks like. It's a little cryptic for the human reader. It also generates several .xsd files that define the data types and data contracts that are used in the input and output messages. Figure A-21 shows one of these files displayed in the XML Schema Explorer.

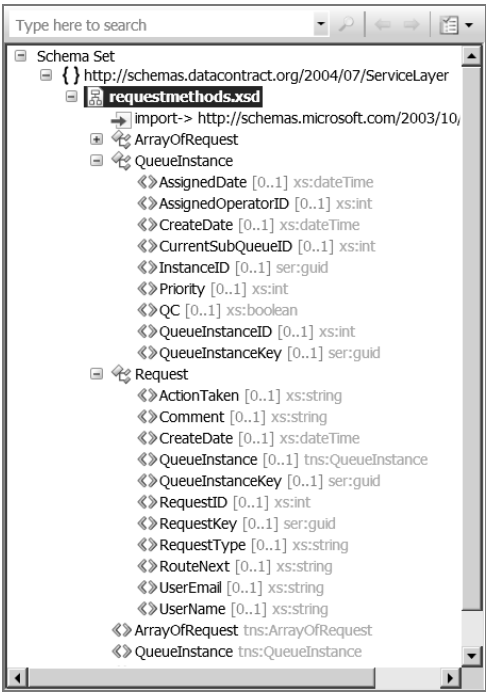


Figure A-21. XML Schema Explorer

Database Design

Because the queue logic is provided by the UserTasks project, the service layer can focus on request-specific design elements. Consequently, the data model is quite simple, as shown in Figure A-22.

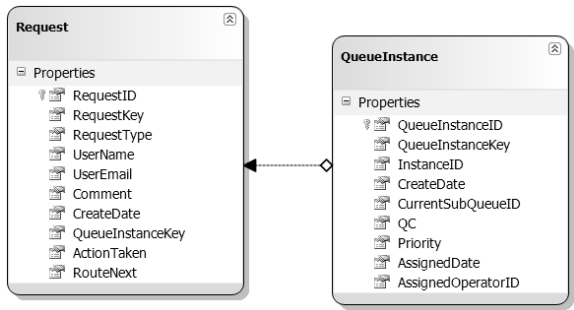


Figure A-22. Service layer data model

The Request table contains the data, such as name, e-mail, and comment, entered by the user who submitted the request. It also records the action taken, which is entered by the operator who worked the request. It has a reference to a QueueInstance record. The QueueInstance record handles all the queue details such as the current queue, to whom it's assigned, and so on. This design keeps the Request table clear of all these “plumbing” details.

Activities

The ServiceLayer project implements some custom activities, which are listed in Table A-2.

Table A-2. Activities provided in the ServiceLayer project

Activity Name	Description
BuildRequestList	Maps the data returned by GetQueueInstances into a list of Request objects.
CreateRequest	Creates a new Request record.
LoadRequest	Load the specified Request from the database.
UpdateRequest	Updates a Request.

The CreateRequest and UpdateRequest use the PersistRequest extension to perform the database update when the workflow is persisted. The BuildRequestList activity is the only one that is particularly interesting. The GetQueueInstances activity (provided in the UserTasks project) handles all the logic to determine which records are available to be worked, but it knows nothing about requests. It uses only the QueueInstance table as well as the Queue and SubQueue setup tables. BuildRequestList takes the list of QueueInstance objects returned by GetQueueInstances and maps them to a list of Request objects. The implementation is shown in Listing A-2.

Listing A-2. BuildRequestList Implementation

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Activities;
using UserTasks;
using UserTasks.Extensions;

namespace ServiceLayer.Activities
{
    public sealed class BuildRequestList : CodeActivity
    {
```

```

public InArgument<UserTasks.QueueInstance[]> QueueInstanceList { get; set;}
public OutArgument<Request[]> RequestList { get; set; }

protected override void Execute(CodeActivityContext context)
{
    // Get the connection string
    DBConnection ext = context.GetExtension<DBConnection>();
    if (ext == null)
        throw new InvalidProgramException("No connection string available");

    RequestDataContext dc = new RequestDataContext(ext.ConnectionString);

    // Get the list of QueueInstances
    UserTasks.QueueInstance[] qiList = QueueInstanceList.Get(context);
    if (qiList != null && qiList.Count() > 0)
    {
        // Build a list of Request objects
        Request[] rList = new Request[qiList.Count()];
        int i = 0;
        foreach (UserTasks.QueueInstance qi in
            QueueInstanceList.Get(context))
        {
            Request r = dc.Requests.SingleOrDefault
                (x => x.QueueInstanceKey == qi.QueueInstanceKey);
            rList[i++] = r;
        }

        RequestList.Set(context, rList);
    }
}
}
}

```

Workflow Design

This web service is implemented using the workflow designer, which produces an .xamlx file. Figure A-23 shows the overall design with some of the activities collapsed to fit the diagram on a page.

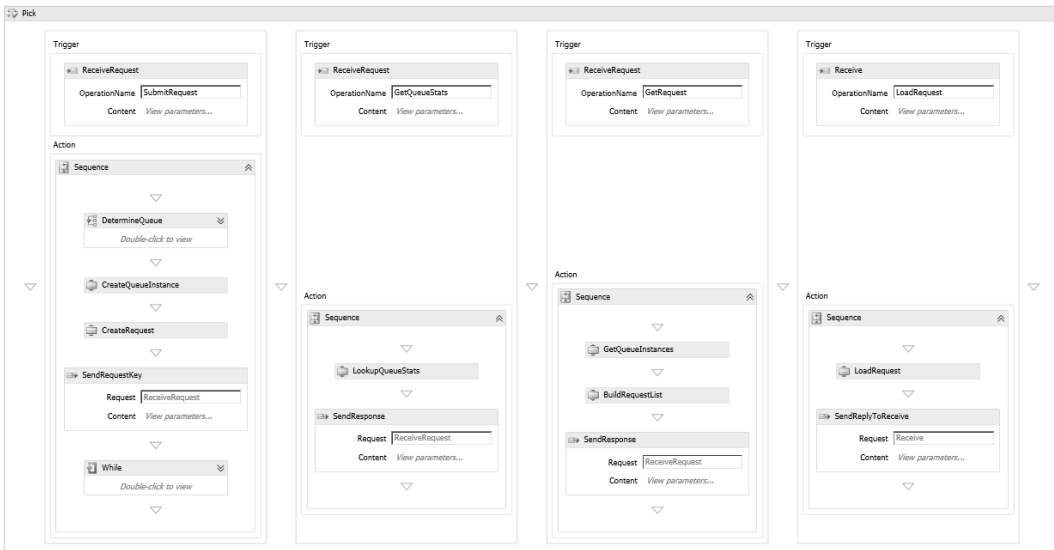


Figure A-23. Overall workflow design

The workflow uses the Pick activity that was introduced in Chapter 10. To review, a Pick activity contains one or more PickBranch activities, in which each contains a Trigger activity and an Action sequence. When the Trigger is executed, the associated Action sequence is started, and all other PickBranch activities are cancelled. In this case there are four branches; the Trigger for each one contains a Receive activity that waits for a specific web service method.

The four methods are as follows:

- **SubmitRequest:** Initiates a new request.
- **GetQueueStats:** Gets the number of request in each queue.
- **GetRequest:** Gets the available request(s) from the specified queue.
- **LoadRequest:** Returns the request details of the specified request.

The last three are fairly straightforward. They perform a database operation and return the appropriate results back to the caller using a SendReply activity. When the application wants to get the current statistics about the queues, it calls the `GetQueueStats()` method of the web service. This will create a new workflow instance that is completed as soon as the response is sent back to the application. This workflow is very short-lived. The instance store is configured to remove the record from the `InstancesTable` when the workflow is completed. So using a workflow instance to perform a simple task does not leave any artifacts behind.

SubmitRequest

When `SubmitRequest` is called, it first determines the correct subqueue using a Switch activity (see Chapter 4 for details). It then creates a `QueueInstance` record and a `Request` record and returns data back

to the caller using `SendReply`. So far, this is similar to the other branches. However, the workflow continues after the `SendReply` activity. It is followed by a `While` activity; the design of this is shown in Figure A-24.

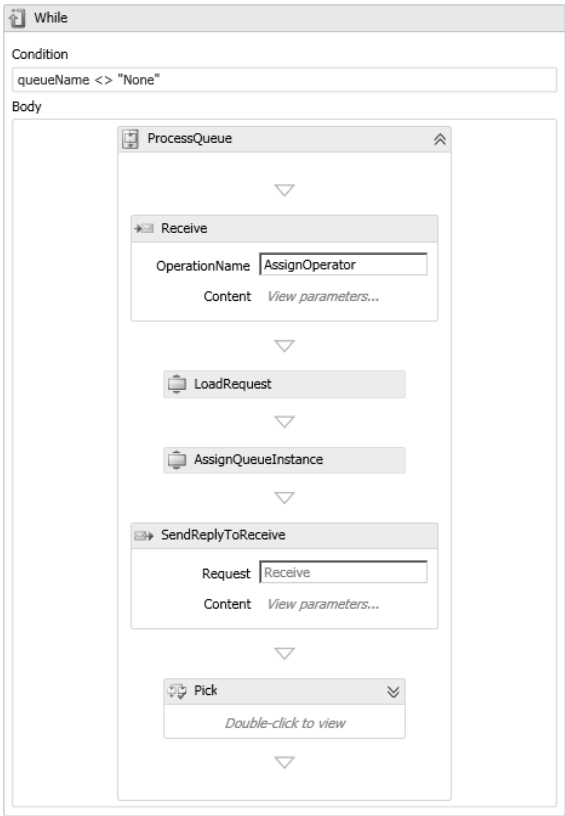


Figure A-24. While activity

The Condition on the While activity is as follows:

```
queueName <> "None"
```

This means that the While activity will continue to execute as long as the request is assigned to a queue. When the request is completed, without forwarding the request to another queue, the `queueName` is set to “None”, causing the While activity to complete. The logic inside the While activity starts with a `Receive` activity that waits for the `AssignOperator` method. A request must be assigned to an operator before it can be worked. This helps ensure that two people are not working on the same request.

Processing a Request

When the `AssignOperator()` method is called, the `QueueInstance` associated with that request is updated to record the assignment and the response is sent back. The workflow then continues with another `Pick` activity that is shown in Figure A-25.

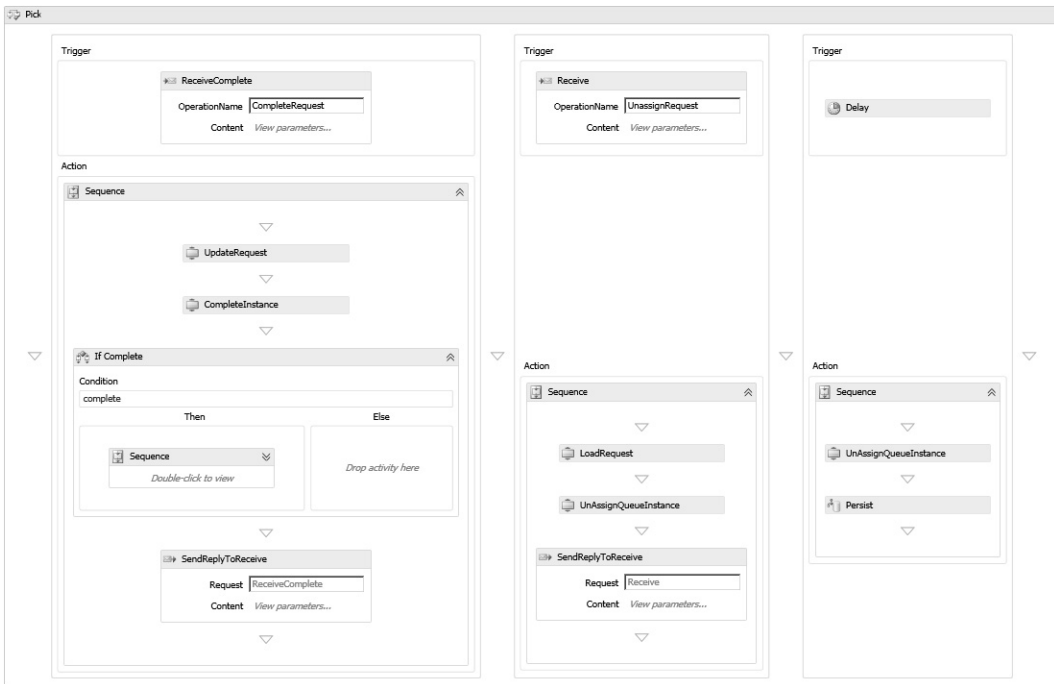


Figure A-25. The final `Pick` activity

When a request has been assigned to an operator, one of three things can happen:

- The request is completed.
- The page is cancelled forcing the request to be unassigned.
- Neither.

The three branches represent these scenarios. The first branch completes the request, and the second branch unassigns the request so someone else can work it. The third branch uses a `Delay` activity to wait for five minutes. If nothing has been done within that time, the request is automatically unassigned. If an operator has a request assigned to them and they decide to leave for the day and simply close their browser, the request would be left assigned to them. This means that no one else could work that request. This third branch was added to take care of that scenario.

If the `CompleteRequest()` method is called, the first branch updates the request with the data provided. It then executes the `CompleteInstance` activity that was described earlier. This activity

determines whether the item in the queue needs to be reviewed in QC mode. It outputs a `Complete` argument to indicate whether the item is complete or whether it needs to be reviewed. Figure A-26 shows the sequence that is executed if the request is complete.

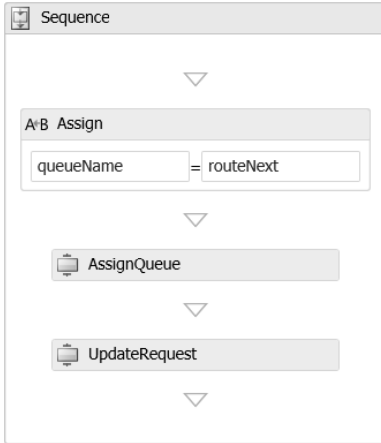


Figure A-26. The sequence for completed requests

■ **Note** When I say the request is complete, I'm referring to that particular task for that request is complete. The request might need to be worked in other queues before the workflow is completed.

The `routeNext` variable is specified in the `CompleteRequest()` method; it is determined by the operator working the request. This is copied to the `queueName` variable. If no queue was selected, this will signal that the workflow is done, and the `While` activity will complete. The `AssignQueue` activity is called to update the current subqueue for the `QueueInstance`. The `UpdateRequest` activity will clear the `RouteNext` field.

Correlation

The concept of correlation was introduced briefly in Chapter 8. A typical workflow will have hundreds, perhaps thousands, of workflow instances executing simultaneously. When a workflow design includes sending messages to (and between) instances, correlation provides the mechanism to ensure that messages are sent to the correct instance. There are three types of correlation provided by WF 4.0.

The first (and probably easiest) correlation is called request reply correlation. It is used when you have a two-way communication between workflow activities. For example, you send a request and wait for the response. You used this in Chapter 8 (and others). By placing the `Send` and `ReceiveReply` activities within a `CorrelationScope` activity, the workflow took care of the details for you. In this case, correlation was accomplished through the communication channel that was established between the sender and the receiver.

The other two types support more complex scenarios in which there are multiple messages between the workflows. The first is referred to as context correlation. In this case, the client sends a request, and the server includes a context ID with the response. This context ID must be included with all subsequent messages from the client. This is used by the server to associate the same instance that responded to the first message. This approach requires logic on both the client and server. This approach also requires that the first message be a two-way message; the server has to send back a response that includes the context ID.

The last approach, query correlation, is accomplished on the server side only. In a sense, this is actually very similar to context correlation. There is some sort of key that is included with each request that identifies the corresponding workflow instance. With context correlation, this key is generated by the server on the initial request. However, with query correlation, this key is based on data included in the message.

This project uses query correlation and the common key is the RequestKey, which is a Guid generated by the application. The RequestKey is provided as one of the parameters on every call to the service. When the first message is received, the RequestKey is mapped to that workflow instance. This is done through a correlation initializer, which sets up the mapping between the RequestKey and the associated workflow instance. On subsequent calls, this is extracted (queried) from the data in the incoming message and the mapping is used to determine the workflow instance.

Query correlation is accomplished through the Receive activity. There are three properties on the Receive activity that support correlation, as shown in Figure A-27.

System.ServiceModel.Activities.Receive	
Search: <input type="text"/> Clear	
Correlations	
CorrelatesOn	(Collection) ...
CorrelatesWith	requestIDHandle ...
CorrelationInitializers	(Collection) ...
Misc	
Content	(Content) ...
DisplayName	ReceiveRequest
OperationName	SubmitRequest
ServiceContractName	{http://tempuri.org/}IProces
More Properties	
Action	
CanCreateInstance	<input checked="" type="checkbox"/>
KnownTypes	(Collection) ...
ProtectionLevel	(null)
SerializerOption	DataContractSerializer

Figure A-27. Properties of a Receive activity

The CorrelatesWith property defines the handle, which is specified as a CorrelationHandle type. This is a workflow variable that is persisted with the workflow. A CorrelationInitializer is then added to the first Receive activity, as shown in Figure A-28.

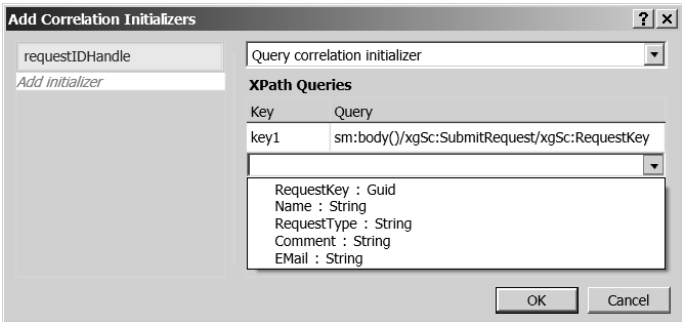


Figure A-28. A correlation initializer

The XPath query might seem a little cryptic, but don't worry; the Visual Studio takes care of this for you. You only need to select the appropriate property from the drop-down menu. The drop-down menu lists all the parameters in the incoming message. When the RequestKey is chosen, the query is generated automatically.

On subsequent Receive activities, instead of setting the CorrelationInitializer, the CorrelatedOn property is set. This is done by selecting the correct parameter; a query is then generated for you, as shown in Figure A-29.

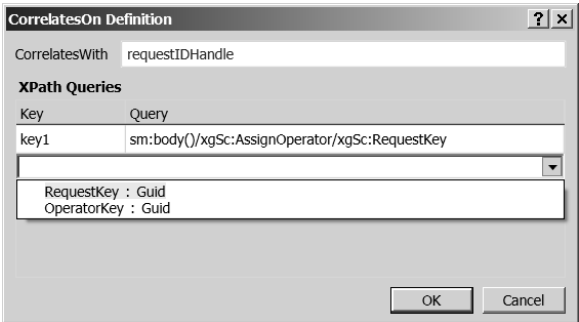


Figure A-29. The CorrelatesOn property

Using WorkflowServiceHost

This project uses several workflow extensions that were introduced in previous chapters (persistence in Chapter 11, sharing configuration data in Chapter 12, tracking in Chapter 13, and custom persistence in Chapter 14). In those projects there was a console or WPF application that configured these extensions and added them to the workflow instances as they were created. In this project, this is done by the WorkflowServiceHost.

Writing Extensions

To add extensions when using the `WorkflowServiceHost`, they must be configured in the `web.config` or `app.config` file. This requires some extra steps when writing the extensions. The modified implementation of `DBConnection.cs` is shown in Listing A-3.

Listing A-3. Implementation of the `DBConnection` Extension

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.Configuration;
using System.Web.Configuration;
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Channels;
using System.ServiceModel.Configuration;
using System.ServiceModel.Description;

namespace UserTasks.Extensions
{
    /*****
    // The extension class is used to define the behavior
    *****/
    public class DBConnectionExtension : BehaviorExtensionElement
    {
        public DBConnectionExtension()
        {
            Console.WriteLine("Behavior extension started");
        }

        [ConfigurationProperty("connectionStringName", DefaultValue = "",
            IsKey = false, IsRequired = true)]
        public string ConnectionStringName
        {
            get { return (string)this["connectionStringName"]; }
            set { this["connectionStringName"] = value; }
        }

        public string ConnectionString
        {
            get
            {
                ConnectionStringSettingsCollection connectionStrings =
                    WebConfigurationManager.ConnectionStrings;
                if (connectionStrings == null) return null;
                string connectionString = null;
                if (connectionStrings[ConnectionStringName] != null)
                {

```

```

        connectionString =
            connectionStrings[ConnectionStringName].ConnectionString;
    }
    if (connectionString == null)
    {
        throw new ConfigurationErrorsException
            ("Connection string is required");
    }
    return connectionString;
}
}

public override Type BehaviorType
{
    get { return typeof(DBConnectionBehavior); }
}
protected override object CreateBehavior()
{
    return new DBConnectionBehavior(connectionString);
}
}

/*****
// The behavior class is used to create an extension
// for each new instance
*****/
public class DBConnectionBehavior : IServiceBehavior
{
    string _connectionString;

    public DBConnectionBehavior(string connectionString)
    {
        this._connectionString = connectionString;
    }

    public virtual void ApplyDispatchBehavior
        (ServiceDescription serviceDescription, ServiceHostBase serviceHostBase)
    {
        WorkflowServiceHost workflowServiceHost
            = serviceHostBase as WorkflowServiceHost;
        if (null != workflowServiceHost)
        {
            string workflowDisplayName
                = workflowServiceHost.Activity.DisplayName;

            workflowServiceHost.WorkflowExtensions.Add((
                => new DBConnection(_connectionString));
        }
    }
}

public virtual void AddBindingParameters

```

```

        (ServiceDescription serviceDescription,
         ServiceHostBase serviceHostBase,
         Collection<ServiceEndpoint> endpoints,
         BindingParameterCollection bindingParameters)
    {
    }

    public virtual void Validate
        (ServiceDescription serviceDescription,
         ServiceHostBase serviceHostBase)
    {
    }
}

/*****
// This is the actual extension class
*****/
public class DBConnection
{
    private string _connectionString = "";

    public DBConnection(string connectionString)
    {
        _connectionString = connectionString;
    }

    public string ConnectionString { get { return _connectionString; } }
}
}

```

There are three classes implemented in this file:

- DBConnectionExtension
- DBConnectionBehavior
- DBConnection

DBConnectionExtension is derived from the BehaviorExtensionElement class. It specifies the configuration values that are supported. In this case, there is only one: connectionStringName. It also provides a ConnectionString() method that obtains the connection string from the configuration file using the connectionStringName parameter. Finally, it overrides the CreateBehavior() method, which creates a DBConnectionBehavior object passing in the connection string to the constructor.

The DBConnectionBehavior class implements the IServiceBehavior interface. This interface defines an ApplyDispatchBehavior() method that creates an extension and adds it to a workflow instance. This is roughly equivalent to the SetupInstance() method you wrote in Chapter 12. When the WorkflowServiceHost is started, it looks for all the configured extensions and obtains an IServiceBehavior interface for each. As each workflow instance is created, it calls the ApplyDispatchBehavior() method on each of the IServiceBehavior interfaces. The ApplyDispatchBehavior() method creates a DBConnection class, passing in the connection string to the constructor and then adds it to the WorkflowExtensions collection.

The `DBConnection` class in same implementation you created in Chapter 12. It provides a `ConnectionString` property that supplies the connection string to any activity that needs it.

Configuring Extensions

A subset of the `web.config` file is shown in Listing A-4.

Listing A-4. A Portion of the `web.config` file

```
<configuration>
  <connectionStrings>
    <add name="Request" connectionString=
      "Data Source=localhost;Initial Catalog=Appendix;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.serviceModel>
    <extensions>
      <behaviorExtensions>
        <add name="dbConnection"
          type="UserTasks.Extensions.DBConnectionExtension, UserTasks,
            Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      </behaviorExtensions>
    </extensions>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <dbConnection connectionStringName="Request"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

First, a connection string named “Request” is defined in the `connectionStrings` section. This allows you to reference it by name in various places in the `web.config` file. The advantage to this approach is that the actual connection string is specified only once. If you need to modify it later, you have to change it in only one place.

Next, an extension named “dbExtension” is added to the `behaviorExtensions` section. Note that the actual class that is referenced is the `DBConnectionExtension` class, not the `DBConnectionBehavior` or `DBConnection` classes. This extension is then configured in the `behaviors` section. The extension is specified by name, `dbConnection`, and its configuration values are defined. There is only one, `connectionStringName` and it is set to “Request” to use the connection string defined earlier.

Configuring Persistence

The persistence extension, `SqlWorkflowInstanceStore`, is configured in the `behavior` section as well. You do not need to add anything to the `behaviorExtensions` section. The subset of the `web.config` file is shown in Listing A-5.

Listing A-5. Configuring Persistence

```

<behaviors>
  <serviceBehaviors>
    <behavior>
      <sqlWorkflowInstanceStore
        connectionStringName="Request"
        instanceCompletionAction="DeleteAll"
        instanceLockedExceptionAction="NoRetry"
        instanceEncodingOption="GZip"
        hostLockRenewalPeriod="00:00:30" />
      <workflowIdle
        timeToUnload="00:00:10"
        timeToPersist="00:00:05" />
    </behavior>
  </serviceBehaviors>
</behaviors>

```

Notice that it uses the same `connectionStringName`. The code in Listing A-5 also configures the `workflowIdle` behavior. The `timeToUnload` property is set to 10 seconds. This will keep the instance in memory for 10 seconds after it has entered the Idle state.

Configuring Tracking

To add the tracking extension `QueueTracking`, the entries are added to extensions and behavior sections just as it was for the `DBExtension` discussed previously. In addition, a tracking section is added to specify queries used to define the tracking events that are to be included. Refer to Chapter 13 for more information about tracking queries. The `web.config` entries are shown in Listing A-6.

Listing A-6. Configuring Tracking

```

<configuration>
  <system.serviceModel>
    <extensions>
      <behaviorExtensions>
        <add name="tracking"
          type="UserTasks.Extensions.QueueTrackingExtension, UserTasks,
            Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      </behaviorExtensions>
    </extensions>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <tracking connectionStringName="Request"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <tracking>
      <profiles>

```

```

    <trackingProfile name="Queue_Tracking">
      <workflow>
        <customTrackingQueries>
          <customTrackingQuery name="*" activityName="*" />
        </customTrackingQueries>
      </workflow>
    </trackingProfile>
  </profiles>
</tracking>
</system.serviceModel>
</configuration>

```

The complete web.config file is shown in Listing A-7.

Listing A-7. Complete web.config File

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="Request" connectionString=
      "Data Source=localhost;Initial Catalog=Appendix;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.serviceModel>
    <extensions>
      <behaviorExtensions>
        <add name="persistRequest"
          type="ServiceLayer.Extensions.PersistRequestExtension, ServiceLayer,
            Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
        <add name="persistQueueInstance"
          type="UserTasks.Extensions.PersistQueueInstanceExtension, UserTasks,
            Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
        <add name="dbConnection"
          type="UserTasks.Extensions.DBConnectionExtension, UserTasks,
            Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
        <add name="tracking"
          type="UserTasks.Extensions.QueueTrackingExtension, UserTasks,
            Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      </behaviorExtensions>
    </extensions>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- To avoid disclosing metadata information, set the value below to
            false and remove the metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="True"/>
          <!-- To receive exception details in faults for debugging purposes,
            set the value below to true. Set to false before deployment to

```



```

        avoid disclosing exception information -->
<serviceDebug includeExceptionDetailInFaults="True"/>
<!-- This line configures the persistence service -->
<sqlWorkflowInstanceStore
  connectionStringName="Request"
  instanceCompletionAction="DeleteAll"
  instanceLockedExceptionAction="NoRetry"
  instanceEncodingOption="GZip"
  hostLockRenewalPeriod="00:00:30" />
<workflowIdle
  timeToUnload="00:30:00"
  timeToPersist="00:00:05" />
<!-- Configure the connection string for the persistence extensions-->
<dbConnection connectionStringName="Request"/>
<persistRequest connectionStringName="Request"/>
<persistQueueInstance connectionStringName="Request"/>
<tracking connectionStringName="Request"/>
</behavior>
</serviceBehaviors>
</behaviors>
<tracking>
  <profiles>
    <trackingProfile name="Queue_Tracking">
      <workflow>
        <customTrackingQueries>
          <customTrackingQuery name="*" activityName="*" />
        </customTrackingQueries>
      </workflow>
    </trackingProfile>
  </profiles>
</tracking>
</system.serviceModel>
<system.webServer>
  <modules runAllManagedModulesForAllRequests="true"/>
</system.webServer>
</configuration>

```

Summary

This sample project is just one way workflow can be used to implement a solution. There are other approaches, such as the one described in Chapter 7 in which workflow was used to organize a processing algorithm. My goal throughout this book was to give you a variety of applications. Hopefully one or more of these will resemble a project you are currently working on.

You now have the tools to use the capabilities provided by Workflow Foundation. I wish you great success as you add this to your repertoire of software design patterns.

Index

A

- Action property, 173
- Action Taken field, 411, 412, 413
- Action<T> class, 219
- activities
 - compound, 11
 - PerformLookup, 158–161
 - Pick, 173–174
 - QCPolicy, 416–418
 - ReceiveRequest, 155–158
 - SendResponse, 155–158
- Activities member, 26
- Activities property, 28
- Activity class, 21, 25
- activity counters, incrementing, 396–399
- Activity template, 397
- Activity1 file, 379
- ActivityConfig class, 379, 395, 396, 398, 400, 402
- ActivityContext class, 27
- ActivityData property, 398
- ActivityName property, 236
- ActivityStateQuery class, 235
- ActivityStateRecord class, 232, 245
- Add activity, 53, 54
- Add Existing Item dialog, 124, 125
- Add Existing Project dialog, 60, 61, 70, 80
- Add Item link, 408
- Add new catch link, 322, 373
- Add New Item dialog, 46, 47, 368
- Add new parameter link, 164
- Add New Project dialog, 362, 363
- Add Rule link, 385, 391
- AddAssignment() method, 286
- AddBindingParameters() method, 302
- AddComment activity, 216–218
- AddComment class, 216
- AddComment() method, 216, 218
- AddEvent() method, 128, 131, 232
- AddLead class, 239
- AddLead() method, 233, 280, 281, 283
- AddLead.xaml file, 178
- AddLead.xaml.cs, 247, 264, 265, 294
- AddNewRequest() method, 141, 145
- AddToCollection activity, 348
- AllowSelection option, 414
- Always value, 389
- Amount property, TransactionConfig class, 379
- anonymous class instances, 26
- app.config file, 290, 312–313, 431
- AppendixData folder, 405
- Appendix.zip file, 405
- application configuration (app.config) file, 196
- Application Configuration File template, 101, 312
- ApplicationInterface class, 131, 134, 141, 181, 311
 - methods, 144–148
 - static reference, 128–129

- applications, running in client workflow, 172
- ApplyDispatchBehavior() method, 302, 433
- appSettings section, 102
- App.xaml file, 291
- Argument type, 170
- arguments
 - configuring, 400
 - persisting, 207
- arguments, passing, 45–58
 - creating new solution, 45–48
 - implementing workflow, 48–56
 - defining arguments, 49–51
 - designing workflow, 51
 - expression activities, 53–56
 - Switch activity, 51–52
 - invoking workflow, 56–58
 - running application, 58
- Arguments window, VS 2010 IDE, 5
- ArgumentType drop-down menu, VS 2010, 50
- aspnetdb database, 406
- Assign activity, 11, 21, 22, 28, 83, 110
- Assign class, 21, 28
- Assign To property, 164
- AssignedTo argument, 259, 261
- AssignedTo property, 260, 273
- AssignLead activity, 259, 261, 262, 273, 283, 287, 308
- AssignLead class, 258–260
- Assignment class, 257, 262, 294, 313
- Assignment record, 308
- Assignment table, 256–262, 311, 316
 - adding LINQ to SQL class, 256–258
 - AssignLead class, 258–260
 - CreateAssignment class, 260–262
- Assignment.dtt query, 316
- AssignOperator() method, 426, 427–428
- AssignQueue activity, 415, 428
- AssignQueueInstance activity, UserTasks project, 415
- asterisk (*), 235

B

- BeginInvoke() method, 131
- BeginOnSave() method, 278, 281
- behavior section, 434
- BehaviorExtensionElement class, 433
- behaviorExtensions section, 434
- behaviors, 301
- binding WCF endpoint, 100
- Blank Solution template, 59, 69, 79, 209
- Body activity, 324
- Body property, 300, 395
- Body section, 12, 323, 324, 325, 326, 327, 329
- Body sequence, 325, 327
- BookInfo classes, 162
- BookInfo objects, 172
- BookInfo.cs class, 152
- BookInfoList class, 155, 156, 158, 166
- BookInventory assembly, 156, 163
- BookInventory project, 152, 158, 169
- BookInventory2.xamlx file, 163, 166, 167, 174
- BookInventory.BookSearch property, 157
- BookInventory.xaml file, 152
- BookInventory.xamlx file, 155, 160
- BookList property, 161
- BookLookup class, 168
- BookLookup.ServiceReference1.Activities namespace, 170
- BookLookup.ServiceReference1.BookInventory assembly, 170
- BookmarkResumptionQuery class, 235
- BookmarkResumptionRecord class, 232, 245
- BookSearch class, 155
- bool type, 27
- Branch class, 96, 97, 99, 100, 105
- Browse and Select a .Net type dialog, 366, 367
- btnAddLead_Click event handler, 210
- btnAddLead_Click() method, 211, 215
- btnAssign_Click event handler, 210
- btnAssign_Click() method, 211, 263, 264
- Budget property, 351

- BuildRequestList activity, ServiceLayer project, 423
- built-in activities, extending, 79–92
 - custom activities, 80–85
 - implementing, 80–82
 - LookupItem activity, 82–85
 - running application, 85
 - InvokeMethod activity, 86–92
 - adding discount, 91
 - OrderDiscount class, 86–87
 - Parameters property, 89
 - Result property, 89–90
 - running application, 91–92
 - TargetObject property, 88
- reusing chapter 6 project, 79

C

- CacheMetadata method, 217
- CallItOffException class, 321, 322
- cancellation handler, 336
- Cancellation Handler section, 324, 325, 328, 329
- Cancellation section, 323
- CanCreateInstance property, 108, 164, 311
- Catch activity, 76, 323, 373
- Catch section, 342
- Catches section, TryCatch activity, 70
- Category property, CustomerConfig class, 379, 403
- chaining feature, 388
- Chaining option, 389
- Chaining property, 389
- CheckStock activity, 70–76
 - Catch activity, 76
 - defining exceptions, 70–72
 - ForEach activity, 72
 - If activity, 72–73
 - running application, 76
 - Throw activity, 73
 - TryCatch activity, 70
- classes
 - adding to projects, 46
 - copying from LeadGenerator, 294

- defining, 47
- ClearCollection activity, 355–360
- Click event, 142
 - _Click event handler, 258
- "Click to browse" link, 366
- client workflow, 168–172
 - defining, 170–171
 - implementing host application, 171–172
 - running application, 172
- ClientService class, 133–134
- Code Activity template, 166, 397
- CodeActivity class, 22, 80, 364, 368
- CodeActivityContext class, 236
- coded workflows, 23–32
 - creating console application, 23–24
 - defining workflow, 24–29
 - running application, 29
- Collection property, AddToCollection activity, 348
- collections, 345–360
 - ClearCollection activity, 355–360
 - creating, 345–349
 - AddToCollection activity, 348
 - defining, 346
 - initial workflow, 347–348
 - invoking workflow, 348
 - running application, 349
 - printing, 349–351
 - searching, 353–355
 - ExistsInCollection activity, 354
 - overriding Equals() method, 353–354
 - RemoveFromCollection activity, 355
 - sorting, 351–353
- CollectionsWF() method, 352, 354
- CollectionWF() method, 347, 348, 350
- CollectValues() method, 216
- CommentExtension class, 215, 216, 278
- Comments property, 219
- Compare() method, IComparer interface, 352
- CompensableActivity object, 323, 324
- Compensate activity, 341, 342, 344
- compensation

- customizing, 337–342
 - handlers, 332–336
- Compensation Handler section, 324, 326, 333, 334
- Compensation section, 323
- CompensationToken class, 337, 338
- Complete argument, 428
- Complete button, 315, 411, 412, 413
- Complete output argument, 416
- CompleteAssignment class, 304
- Completed event, 219
- Completed property, 219
- CompleteInstance activity, 427
- CompleteInstance activity, UserTasks project, 415–416
- CompleteRequest() method, 427, 428
- CompletionCondition property, 331, 332
- compound activities, 11
- ComputeDiscount() method, 87, 89
- Condition action, 384, 387
- Condition element, If activity, 11
- Condition property, 27, 36, 73, 341, 354, 390
- Confirm activity, 339, 340, 344
- confirmation
 - activities, 328–329
 - customizing, 337–342
- Confirmation Handler section, 324, 326, 328, 329
- Confirmation section, 323, 328, 329
- ConnectionString argument, 214
- connectionString attribute, 210, 256
- _connectionString member, 302
- ConnectionString() method, 433
- ConnectionString property, 434
- connectionStrings section, 434
- connscionStringName parameter, 433
- console application, implementing, 402
- Console Application template, 95
- Content link, 165
- Content property, 104, 107, 108, 110, 157, 308
- context correlation, 429
- Continue() method, 136
- CorrelatedOn property, 430
- CorrelatesOn property, 430
- CorrelatesWith property, 108, 429
- correlation, in workflow project, 428–430
- correlation initializer, 429, 430
- CorrelationHandle type, 429
- CorrelationScope activity, 106, 308, 428
- counter variable, 25
- Create Argument link, 394
- Create New SQL Server Database dialog, 185
- Create Scripts folder, 405
- Create variable link, 337, 338, 395
- CreateAssignment activity, 261, 273, 287, 294, 308, 311
- CreateAssignment class, 260–262, 286, 287
- CreateAssignment.cs file, 294
- CreateBehavior() method, 433
- CreateBookmark() method, 136
- CreateLead activity, 191–192, 213, 214, 282–283
- CreateLead class, 236, 238
- CreateQueueInstance, activity, UserTasks project, 415
- CreateRequest activity, 104–106, 141, 423
- CreateRequest class, 104
- CreateResponse activity, 108–110
- CreateSqlTrackingParticipant() method, 246
- CreateTrackingParticipant() method, 233, 234
- CreateWorkflow() method, 22, 24, 25
- custom activities, 80–85
 - creating workflow application, 393–396
 - executing (3.5), 367–374
 - creating custom activity, 368–370
 - invoking custom activity, 371–373
 - running application, 374
 - throwing exception, 370–371
 - implementing, 80–82
 - LookupItem activity, 82–85
 - running application, 85
- CustomActivity class, 369, 370
- CustomerConfig class, 379, 394, 400, 402
- CustomTrackingQuery class, 236

CustomTrackingRecord class, 232, 236–238, 245

■ D

data context class, 260
 data structures, defining, 376–379
 data, using parameters to pass, 162–168
 modified PerformLookup activity, 166
 second workflow service, 163–165
 testing, 167–168
 database design, UserTasks project, 414
 database, in workflow project
 configuring for, 405–406
 design of service layer, 422–423
 design of UserTasks project, 414
 DataContext class, 281
 DataElements.cs file, 376, 393
 DateTime class, 10
 DateTime.Now assembly, 387
 DBConnection class, 433, 434
 DBConnection extension, 415, 417, 431
 dbConnection extension, 434
 DBConnectionBehavior class, 433, 434
 DBConnection.cs, 431
 DBConnectionExtension class, 433, 434
 DBExtension class, 212, 259, 302
 dbExtension extension, 434
 DBExtensionBehavior class, 301–302, 304
 DBExtension.cs file, 301
 Default property, 9, 338
 Delay activity, 13, 110, 173, 174, 273, 325, 327, 328, 336, 339, 427
 dependency properties, adding, 380–382
 DependencyProperty object, 395
 Deposit activity, 333
 Design tab, 293
 Details tab, Event Viewer application, 240, 241
 determining priority, 390
 Dictionary object, 58, 141, 172, 197, 215, 216, 402
 DisplayName argument, 83
 DisplayName property, 6, 323, 400

DoWhile activity, 12
 Duration property, 13, 29, 173, 327, 328

■ E

Else action, 384, 387, 388, 391
 Else element, of If activity, 11
 Endpoint property, 300
 Enlist() method, 260, 261
 EnlistTransaction() method, 281
 EnterLead workflow modifications activity, 304–308
 Equals() method, overriding, 353–354
 Error List, VS 2010 IDE, 4
 ETW. *See* Event Tracing for Windows
 EtwTrackingParticipant class, 238–241
 running application, 239–241
 setting up extension, 238
 TrackingProfile class, 239
 event handlers, 142–143, 219–220
 Event Tracing for Windows (ETW), 238–241
 configuring TrackingProfile class, 239
 running application, 239–241
 setting up extension, 238
 Event Viewer application, running, 239, 240
 _eventLog private member, 232
 exception handling, 69–78
 CheckStock activity, 70–76
 Catch activity, 76
 defining exceptions, 70–72
 ForEach activity, 72
 If activity, 72–73
 running application, 76
 Throw activity, 73
 TryCatch activity, 70
 exceptions, 77–78
 reusing chapter 5 project, 69–70
 Exception property, 334, 335, 341
 Execute() method, 82, 105, 136, 192, 213, 218, 259, 260, 261, 262, 294, 352, 397
 ExecutionPropertyName property, 260

- ExistsInCollection activity, 354
- Explicit Update Only option, 389
- Explicit Update Only value, 389
- expression activities, 53–56
- Expression editor, 9
- Expression property, 38, 390
- expressions, 9, 27–28
- ExpressionServices class, 27
- eXtensible Application Markup Language (xaml), 4
- extensions, 209–221
 - configuring, 212
 - implementing simple, 211–212
 - persistence, 215–220
 - accessing extension from application, 218–219
 - AddComment activity, 216–218
 - creating extension, 215–216
 - event handler syntax, 219–220
 - IPersistenceParticipant interface, 216
 - modifying workflow, 218
 - running application, 220–221
 - setting up solution, 209–211
 - running application, 211
 - setting up database, 210
 - SetupInstance class, 210
 - updating application, 214–215
 - using in activities, 213–214
 - in workflow project
 - configuring, 434
 - persistence extension, 434–435
 - tracking extension, 435–437
 - writing, 431–434
- Extensions folder, 294

F

- Facility activity, 333
- False branch, FlowDecision activity, 36
- FalseLabel property, 36
- FaultType property, 370
- Feedback category, 408, 410
- Finally section, TryCatch activity, 70

- Find() method, 260
- Flowchart activity, 34
- flowchart workflow, 33–42
 - creating, 33–37
 - defining connections, 34–35
 - designing flowchart, 34
 - FlowDecision activity, 35–37
 - running application, 37
 - FlowSwitch activity, 38–40
 - adding, 38
 - adding FlowStep activities, 39–40
 - running application, 40
 - Parallel activity, 40–42
 - adding, 40
 - adding branches, 41–42
 - running application, 42
- Flowchart Workflow Console Application
 - template, Visual Studio 2010, 33
- Flowers activity, 332
- FlowStep activities, 39–40
- FlowSwitch activity, 38–40
- FollowUpLead.xaml.cs file, 291, 294
- ForEach activity, 61–64, 72, 77, 78, 400
- Frequency column, 414
- Frequency property, 385
- Frequency rule, 403
- FromSeconds() method, 29
- Full Chaining value, 389
- functional construction, 26

G

- Get() method, Variable class, 27
- GetComments activity, 218
- Get(env) method, 27, 29
- GetEventListBox() method, 129
- GetHashCode() method, 354
- GetQueueInstances activity, UserTasks
 - project, 415, 423
- GetQueueStats() method, 425
- GetRequest method, 425
- GetValue() method, 82
- Greeting activity, 15

■ H

Halt command, 384, 385, 389
 Hardcode activity, 418
 host application
 communicating with, 123–150
 creating WPF project, 123–127
 implementing application, 141–148
 implementing workflows, 131–141
 running application, 148–150
 TextWriter class, 127–131
 implementing in client workflow, 171–172
 Hour member, DateTime class, 10

■ I, J

IActivityExtensionProvider interface, 218
 IBookInventory class, 155
 ICollection interface, 352
 IComparer interface, 352
 Idle state, 435
 If activity, 10–11, 72–73, 77, 341, 354, 396, 398
 ifReview branch, 390
 IfReview value, Name property, 390
 if-then-else statement, 387
 Implementation property, 106
 implementing console application, 402
 in FlowDecision activity, 36, 38, 39
 InArgument class, 28
 InArgument<string> class, 29
 InArgument<T> class, 27
 IncrementEvalCount() method, 398
 InsertOnSubmit() method, 192, 281
 InstallCommon.sql, 406
 InstallMembership.sql, 406
 InstanceData table, 206, 257
 InstanceStore extension, 301
 integrated development environment (IDE), VS Studio 2010, 4–5
 Interop activity, 365, 366–367, 371, 395, 416
 Interop Properties window, 396
 interoperability with Workflow 3.5, 361
 4.0 Workflow, 361–367

 creating 3.5 workflow, 362–365
 Interop activity, 366–367
 modifying Program class, 362
 running application, 367
 executing custom 3.5 activity, 367–374
 creating custom activity, 368–370
 invoking Custom Activity, 371–373
 running application, 374
 throwing exception, 370–371
 InvalidProgramException, 370, 371, 373
 Invitations activity, 328–329, 332, 334, 339, 343
 invitationsToken variable, 338
 Invoke () method, WorkflowInvoker class, 58, 349
 InvokeMethod activity, 86–92, 141, 195, 198, 218, 273, 311
 adding discount, 91
 OrderDiscount class, 86–87
 Parameters property, 89
 Result property, 89–90
 running application, 91–92
 TargetObject property, 88
 IPersistenceParticipant interface, 216, 278
 IProcessRequest interface, 421
 ISBN argument, 172
 IServiceBehavior interface, 302, 433
 Item property, AddToCollection activity, 348

■ K

Keep me logged in check box, 407

■ L

lambda expressions, 27, 28
 lambda operator, 27
 Language-Integrated Query (LINQ), 313–314
 Lead class, 257, 263, 273
 Lead object, 313
 LeadDataDataContext class, 191
 LeadDataDataContext constructor, 213

- LeadData.Designer.cs file, 314
 - Lead.dtq query, 276
 - LeadGenerator project, 177, 207, 209–221, 229, 255, 276, 277–287
 - application changes, 262–264
 - adding workflow event handlers, 264
 - removing database updates, 263–264
 - updating list of leads, 262–263
 - Assignment table, 256–262
 - adding LINQ to SQL class, 256–258
 - AssignLead class, 258–260
 - CreateAssignment class, 260–262
 - configuring, 212
 - creating application, 177–184
 - defining window form, 178–180
 - renaming window, 178
 - TextWriter class, 181–184
 - designing workflow, 191–195
 - CreateLead activity, 191–192
 - defining workflow activities, 193–195
 - WaitForInput activity, 193
 - EtwTrackingParticipant class, 238–241
 - running application, 239–241
 - setting up extension, 238
 - TrackingProfile class, 239
 - implementing application, 195–200
 - app.config file, 196
 - assigning leads, 198–200
 - creating leads, 197–198
 - loading existing leads, 200
 - implementing simple, 211–212
 - IPersistenceParticipant interface, 278
 - ListBoxTrackingParticipant class, 230–238
 - configuring, 233–234
 - CustomTrackingRecord class, 236–238
 - overriding Track() method, 232–233
 - running application, 238
 - TrackingProfile class, 234–236
 - PersistAssignment extension, 284–286
 - persistence, 215–220
 - accessing extension from application, 218–219
 - AddComment activity, 216–218
 - creating extension, 215–216
 - event handler syntax, 219–220
 - IPersistenceParticipant interface, 216
 - modifying workflow, 218
 - persisting arguments and variables, 207
 - PersistLead extension, 278–284
 - connecting to database, 281
 - modifying AssignLead activity, 283
 - modifying CreateLead activity, 282–283
 - updates, 281
 - running application, 204–205, 220–221, 276, 287
 - setting up database, 185–191
 - installing schema, 185–188
 - LINQ to SQL classes, 188–191
 - setting up solution, 209–211, 229–230, 255–256, 277–278
 - running application, 211
 - setting up database, 210, 230
 - SetupInstance class, 210
 - tracking participants, 230
 - SqlTrackingParticipant class, 241–247
 - configuring, 245–246
 - implementing, 243–245
 - running application, 247
 - setting up database, 241–242
 - updating application, 214–215
 - using in activities, 213–214
 - workflow changes, 272–273
- LeadGenerator\Activities folder, 294
- LeadGenerator.Assignment class, 314
- LeadGeneratorWF class, 193, 273
- LeadGeneratorWF.cs file, 214, 305
- LeadID argument, 261
- leadID parameter, 308
- LeadID property, 313, 314
- LeadResponse application, 305
- LeadResponse folder, 294
- LeadResponse project
 - adding app.config file to, 312–313

- adding to WorkflowServiceHost, 290–294
 - Left property, Add activity, 54
 - LibraryReservation project, 95–122, 123–150
 - creating, 95–102
 - application configuration, 101–102
 - defining messages between applications, 96–101
 - creating WPF, 123–127
 - defining workflows, 102–107
 - implementing application, 114–118, 141–148
 - ApplicationInterface class methods, 144–148
 - event handlers, 142–143
 - maintaining workflow instances, 141–142
 - WorkflowInvoker class, 116–118
 - WorkflowServiceHost class, 114–115
 - implementing workflows, 131–141
 - bookmarks, 135–136
 - listening for messages, 132–135
 - ProcessRequest workflow, 139–141
 - SendRequest workflow, 137–139
 - running application, 118–122, 148–150
 - configuring library branch, 118–120
 - expected results, 120–122
 - TextWriter class, 127–131
 - ListBoxTextWriter class, 129–131
 - providing static reference, 128–129
 - LINQ (Language-Integrated Query), 313–314
 - LINQ to SQL
 - classes, 188–191
 - LINQ to SQL class, 256–258
 - List class, 352
 - ListBox control, 232
 - ListBoxTextWriter class, 128, 129–131, 139, 183
 - ListBoxTextWriter.cs file, 294
 - ListBoxTrackingParticipant class, 230–238
 - configuring, 233–234
 - CustomTrackingRecord class, 236–238
 - overriding Track() method, 232–233
 - running application, 238
 - TrackingProfile class, 234–236
 - ListItem class, 346, 348, 353, 354
 - ListItem object, 354
 - ListView control, 198, 200, 263
 - Load operation, 278
 - Loaded event handler, 135, 212, 233, 238, 245
 - LoadQueueInstance activity, UserTasks project, 415
 - LoadRequest activity, ServiceLayer project, 423
 - LoadRequest method, 425
 - Log In link, 407
 - LookupBook class, 155
 - LookupBook() method, 155, 161
 - LookupBook2 activity, 170
 - LookupBook2() method, 167
 - LookupItem activity, 82–85
 - LookupItem class, 82
 - LookupItem.cs class, 80
 - LookupQueueStats activity, UserTasks project, 415
 - IstEvents control, 131
 - IstLeads control, 263
 - IstLeast_SelectionChanged() event handler, 263
- M**
- MainWindow.xaml file, 291
 - Marketing queue, 410, 411, 413
 - Menu activity, 327
 - Message data property, 157
 - Message property, 369, 372, 373
 - Message type property, 158
 - MessageBodyMember attribute, 99, 155
 - MessageContract attribute, 99–100, 155, 162, 168
 - MessageOut property, 372, 373
 - MinimumAmount property, ActivityConfig class, 379
 - mscorlib assembly, 51, 370, 373
 - Multiple startup projects radio button, 314

MyActivity Properties window, 401
 MyActivity sequence, 399
 MyActivity.xaml file, 398

N

Name property, 235, 236, 390, 395, 400
 NativeActivity base class, 136
 NativeActivityContext class, 259, 260
 navigation bar, Workflow, 326
 Never value, 389
 NewRequest() method, 141
 NoPersistScope activity, 311
 numberBells variable, 25
 NumberSinceLastEval property, 385, 397

O

Object Relational Designer (O/R Designer), 189, 190, 242, 256, 257
 OnIdle event handler, 198
 OperationName class, 164
 OperationName property, 104
 OperatorConfig class, 379, 395, 396, 398, 400, 402
 OperatorConfig table, 414
 OperatorData property, 398
 O/R Designer (Object Relational Designer), 189, 190, 242, 256, 257
 Order class, 47, 48, 50, 58
 Order Flowers activity, 336
 Order object, 58
 OrderDiscount class, 86–87
 OrderItem object processing, 61–68
 adding OrderItem object, 65
 ForEach activity, 61–64
 ParallelForEach activity, 68
 running application, 66–68
 OrderProcess assembly, 50, 51
 OrderProcess project, 59–92
 CheckStock activity, 70–76
 Catch activity, 76
 defining exceptions, 70–72
 ForEach activity, 72

 If activity, 72–73
 running application, 76
 Throw activity, 73
 TryCatch activity, 70
 custom activities, 80–85
 implementing, 80–82
 LookupItem activity, 82–85
 running application, 85
 exceptions, 77–78
 InvokeMethod activity, 86–92
 adding discount, 91
 OrderDiscount class, 86–87
 Parameters property, 89
 Result property, 89–90
 running application, 91–92
 TargetObject property, 88
 OrderItem object processing, 61–68
 adding OrderItem object, 65
 ForEach activity, 61–64
 ParallelForEach activity, 68
 running application, 66–68
 passing arguments, 45–58
 creating new solution, 45–48
 implementing workflow, 48–56, 49–51, 51, 51–52, 53–56
 invoking workflow, 56–58
 running application, 58
 reusing chapter 4 project, 59–61
 reusing chapter 5 project, 69–70
 reusing chapter 6 project, 79
 OutArgument class, 28

P

Parallel activity, 40–42, 323–332, 333
 adding, 40
 adding branches, 41–42
 running application, 42
 ParallelForEach activity, 68
 Parameters dialog, 89
 Parameters property, 89, 308
 parameters, using to pass data, 162–168
 modified PerformLookup activity, 166
 second workflow service, 163–165

- testing, 167–168
- PerformLookup activity, 158–161, 166
- PerformLookup2 activity, 166
- PerformLookup2.cs class, 166
- PerformLookup.cs class, 158
- Persist activity, 311
- PersistableIdle event handler, 210, 211, 220, 247
- PersistAssignment extension, 284–286, 304
- PersistAssignmentBehavior class, 303–304
- PersistAssignment.cs class, 294
- PersistAssignment.cs file, 294, 303
- persisted workflow, lifecycle of, 247
- persistence, 277–287. *See also* SQL persistence
 - IPersistenceParticipant interface, 278
 - LeadGenerator project, 215–220
 - accessing extension from application, 218–219
 - AddComment activity, 216–218
 - creating extension, 215–216
 - event handler syntax, 219–220
 - IPersistenceParticipant interface, 216
 - modifying workflow, 218
 - PersistAssignment extension, 284–286
 - PersistLead extension, 278–284
 - connecting to database, 281
 - modifying AssignLead activity, 283–284
 - modifying CreateLead activity, 282–283
 - updates, 281
 - running application, 287
 - setting up solution, 277–278
- PersistQueueInstance extension, 415
- PersistRequest extension, 423
- Pick activity, 173, 425, 427
- PickBranch activities, 425
- Policy object, 405
- PolicyActivity
 - adding dependency properties, 380–382
 - defining data structures, 376–379
 - defining rules, 384–386
 - overview, 375–379, 392–404
- priority, 390–392
- rule sets
 - chaining, 388–389
 - creating, 383
 - Halt, 389
 - overview, 387–390
 - rules, 387–388
 - rules file, 389–390
 - Update, 389
- port access, allowing, 121, 122
- printing collections, 349–351
- PrintList class, 349, 350
- priority
 - determining, 390
 - entering rules, 391–392
- Priority property, 418
- priority variable, 396
- PriorityPolicy activity, 391, 417, 418
- procedural elements, adding, 7–15
 - Assign activity, 11
 - Delay activity, 13
 - If activity, 10–11
 - Sequence activity, 13
 - variables, 8–9
 - While activity, 12
- Process link, 409
- Process page, 409
- ProcessRequest class, 107–115
 - CreateResponse activity, 108–110
 - Receive activity, 108
 - SendReply activity, 110–114
- Product queue, 411, 412, 413
- Program class, modifying, 362
- Projects tab, 291
- Properties property, 260
- Properties window, VS 2010 IDE, 4–11
 - for Assign activity, 11
 - defining If activity in, 11
 - entering properties in, 9
 - of selected variable, 8–9
- properties windows, If ElseBranch activity, 390
- Propertiew window, VS 2010 IDE, 6
- Property window, MyActivity, 400

PublishValues() method, 216

Q

QC column, 412
 QC mode, 412, 413
 QC review, 412, 413
 QCPolicy activity, 379, 393, 395, 396, 403, 416–418
 QCPolicy assembly, 395, 400
 QCPolicy custom activity, 391
 QCPolicy.cs file, 379, 380, 390
 QCPolicy.rules file, 389
 QPolicy activity, 396
 Queries property, 234
 query correlation, 429
 queue list, 411, 413
 Queue table, 414, 423
 QueueInstance objects, 423
 QueueInstance record, 416, 417, 423, 425
 QueueInstance table, 414, 423
 queueName variable, 428
 queues, displaying, 409
 QueueTrack table, 414
 QueueTracking extension, 418, 435

R

Receive activity, 95–122, 168, 310, 311, 425, 426, 429
 creating project, 95–102
 application configuration, 101–102
 defining messages between applications, 96–101
 defining workflows, 102–114
 ProcessRequest class, 107–114
 SendRequest class, 102–107
 implementing application, 114–118
 WorkflowInvoker class, 116–118
 WorkflowServiceHost class, 114–115
 ProcessRequest class, 108
 running application, 118–122
 configuring library branch, 118–120
 expected results, 120–122

ReceiveReply activities, 428
 ReceiveReply activity, 107, 308
 ReceiveRequest activity, 155–158
 Reception activity, 327, 328, 332, 333, 334, 339, 343
 Reception compensation, 334
 Reevaluate property, 389
 Reevaluation property, 389
 Refresh() method, 259
 Remarks section, 315
 RemoveFromCollection activity, 355
 renaming workflow files, 49
 replicated activities, 59–68
 OrderItem object processing, 61–68
 adding OrderItem object, 65
 ForEach activity, 61–64
 ParallelForEach activity, 68
 running application, 66–68
 Replicator activity, 64
 Request objects, 423
 Request property, 107
 Request queue, 414
 Request record, 425
 request reply correlation, 428
 Request table, 423
 request variable, 107
 requestAddress variable, 103
 RequestBook() method, 101
 requestHandle variable, 107, 108
 RequestKey, 429, 430
 RequestQC activity, 415, 416
 requests in workflow project
 processing, 409–413
 submitting, 408
 ReservationRequest class, 96, 97, 99, 103, 105
 ReservationResponse class, 96, 99, 100
 reserved variable, 107
 ResetEval() method, 398
 RespondToRequest() method, 101, 144
 response variable, 107
 Result property, 54, 89–90, 339
 ResumeBookmark() method, 143, 263
 Rethrow activity, 342
 rethrown exception, 343

- Review property, 390, 398, 418
- review variable, 396
- ReviewPolicy activity, 390, 417
- Right property, Add activity, 54
- RouteNext field, 412, 428
- Rule Set Editor, 383
- rule sets
 - chaining, 388–389
 - creating, 383
 - Halt, 389
 - overview, 387–390
 - rules file, 389–390
 - Update, 389
- rules, 387–388
 - defining, 384–386
 - priority, 391–392
- RuleSet class, 383, 385, 386
- ruleset editor, 384
- RuleSetReference property, 391
- Run as administrator option, 121, 122
- RuntimeTransactionHandle class, 260

S

- Save operation, 278
- Schedule Rehearsal activity, 335
- Schema Explorer, XML, 421, 422
- Scope property, 50, 338
- Search property, 161
- searching collections, 353–355
 - ExistsInCollection activity, 354
 - overriding Equals() method, 353–354
 - RemoveFromCollection activity, 355
- Select links, 410, 411
- Select Rule Set dialog, 386, 387, 391
- SelectionChanged event handler, 198
- Send activity, 95–122, 103, 141, 174, 308, 310, 428
 - creating project, 95–102
 - application configuration, 101–102
 - defining messages between applications, 96–101
 - defining workflows, 102–114
 - ProcessRequest class, 107–114
 - SendRequest class, 102–107
 - implementing application, 114–118
 - WorkflowInvoker class, 116–118
 - WorkflowServiceHost class, 114–115
 - running application, 118–122
 - configuring library branch, 118–120
 - expected results, 120–122
 - SendRequest class, 104
- SendReply activity, 110–114, 152, 168, 310, 420, 425, 426
- SendRequest class, 102–107
 - CreateRequest activity, 104–106
 - ReceiveReply activity, 107
 - Send activity, 104
- SendResponse activity, 155–158
- Sequence activity, 5, 8, 13, 25, 63, 84, 273, 322, 325, 327, 328, 329, 333, 341, 395, 400
 - in coded workflows, 28–29
 - difference from Flowchart activity, 34
- Sequence class, 25
- Sequence workflow, 152
- Sequence1.xaml file, 17
- Sequential option, 389
- sequential workflow, 3, 22
 - adding procedural elements, 7–15
 - Assign activity, 11
 - Delay activity, 13
 - If activity, 10–11
 - Sequence activity, 13
 - using variables, 8–9
 - While activity, 12
 - simple workflow, 4–7
 - designing, 5–6
 - IDE, 4–5
 - Program.cs file, 6–7
 - running application, 7
- Sequential Workflow Console Application, creating, 361, 362
- Sequential Workflow Console Application template, VS2010, 3
- Service class, 115
- service contract, defining for workflow service, 152–155
- service layer, in workflow project, 420–437

- activities, 423
- AssignOperator method, 427–428
- correlation, 428–430
- database design, 422–423
- service contract, 420–421
- SubmitRequest method, 425–426
- using WorkflowServiceHost, 430–437
- workflow design, 424–428
- Service1.wsdl file, 421
- Service1.xamlx file, 152
- ServiceContract attribute, 155
- ServiceContract interface, 100–101
- ServiceContract property, 155
- ServiceHost class, 114, 134–135
- SetupHost() method, 300, 301
- SetupInstance class, 210
- SetupInstance() method, 211, 212, 218, 233, 238, 246, 264, 294, 301, 433
- SetValue() method, 82
- ShoppingList project, 345–360
 - ClearCollection activity, 355–360
 - creating, 345–349
 - AddToCollection activity, 348
 - defining, 346
 - initial workflow, 347–348
 - invoking workflow, 348
 - running application, 349
 - printing, 349–351
 - searching, 353–355
 - ExistsInCollection activity, 354
 - overriding Equals() method, 353–354
 - RemoveFromCollection activity, 355
 - sorting, 351–353
- Solution Explorer, VS 2010 IDE, 4
- Sort() method, List class, 352
- SortCollection class, 351, 352
- sorting collections, 351–353
- SQL persistence, 177–207
 - creating application, 177–184
 - defining window form, 178–180
 - renaming window, 178
 - TextWriter class, 181–184
 - designing workflow, 191–195
 - CreateLead activity, 191–192
 - defining workflow activities, 193–195
 - WaitForInput activity, 193
 - implementing application, 195–200
 - application configuration (app.config) file, 196
 - assigning leads, 198–200
 - creating leads, 197–198
 - loading existing leads, 200
 - persisting arguments and variables, 207
 - running application, 204–205
 - setting up database, 185–191
 - installing schema, 185–188
 - LINQ to SQL classes, 188–191
- SqlPersistenceProvider class, 281
- SqlTrackingParticipant class, 241–254
 - configuring, 245–246
 - implementing, 243–245
 - running application, 247–254
 - setting up database, 241–242
- SqlWorkflowInstanceStore extension, 434
- SqlWorkflowInstanceStore extension, 301
- SqlWorkflowInstanceStoreBehavior class, 301
- Starting activity, 332, 333
- StartupUri attribute, 291
- static Create<T>() method,
 - ExpressionServices class, 27
- static reference, ApplicationInterface class, 128–129
- static WorkflowInvoker class, 7
- Status property, 260
- String input arguments, 170
- Submit link, 408
- submit page, 409
- SubmitChanges() method, 192, 281
- SubQueue table, 414, 423, 425–426
- SupportsQC option, 414
- Switch activity, 51–52, 54, 425
- System namespace, 370
- System.Activities assembly, 337
- System.Activities.Expressions namespace, 53
- System.Workflow.ComponentModel assembly, 365, 393

■ T

- Target property, 339
- TargetObject property, 88
- TargetType drop-down list, 88
- TargetType property, 88
- Text property, 6, 323, 325, 326, 327, 328, 329, 332, 333, 334, 362, 373, 395
- TextWriter class, 127–131, 181–184
 - ListBoxTextWriter class, 129–131
 - providing static reference, 128–129
- TextWriter property, 6
- Then action, 384, 385, 387, 388, 391
- Then element, of If activity, 11
- Then property, 27
- Then section, 11, 73, 341
- Throw activity, 73, 334, 335, 336, 339, 341
- throw statement, 314
- ThrowActivity, 370
- TimeSpan class, 13, 29, 233
- timeToUnload property, 435
- Title argument, 172
- To property, 11, 28
- token variables, 337–338
- Toolbox, VS 2010 IDE, 4
- ToString() method, 29
- TotalAmount argument, 54
- Track() method, 232–233, 236, 245, 418, 419
- tracking events, 229
 - EtwTrackingParticipant class, 238–241
 - running application, 239–241
 - setting up extension, 238
 - TrackingProfile class, 239
 - ListBoxTrackingParticipant class, 230–238
 - configuring, 233–234
 - CustomTrackingRecord class, 236–238
 - overriding Track() method, 232–233
 - running application, 238
 - TrackingProfile class, 234–236
 - setting up solution, 229–230
 - setting up database, 230
 - tracking participants, 230
 - SqlTrackingParticipant class, 241–247
 - configuring, 245–246
 - implementing, 243–245
 - running application, 247
 - setting up database, 241–242
- tracking extension, 435–437
- tracking participants
 - EtwTrackingParticipant class, 238–241
 - running application, 239–241
 - setting up extension, 238
 - TrackingProfile class, 239
 - ListBoxTrackingParticipant class, 230–238
 - configuring, 233–234
 - CustomTrackingRecord class, 236–238
 - overriding Track() method, 232–233
 - running application, 238
 - TrackingProfile class, 234–236
 - overview, 230
 - SqlTrackingParticipant class, 241
 - configuring, 245–246
 - implementing, 243–245
 - running application, 247
 - setting up database, 241–242
- tracking workflow
 - in UserTasks project, 418–420
 - in workflow project, 413
- Tracking.dtt query, 413
- TrackingParticipant class, 230, 232
- TrackingProfile class, 234–236
 - ActivityStateQuery class, 235
 - BookmarkResumptionQuery class, 235
 - CustomTrackingQuery class, 236
 - Event Tracing for Windows, 239
 - WorkflowInstanceQuery class, 235
- TrackingQuery class, 234
- TrackingRecord class, 232, 245
- TransactionConfig class, 379, 394, 400
- TransactionConfig object, 379, 402
- transactionData argument, 394, 400
- TransactionList class, 379, 400, 402
- TransactionList object, 402
- transactions, 255–276
 - application changes, 262–264

- adding workflow event handlers, 264
 - removing database updates, 263–264
 - updating list of leads, 262–263
- Assignment table, 256–262
 - adding LINQ to SQL class, 256–258
- AssignLead class, 258–260
- CreateAssignment class, 260–262
- running application, 276
- setting up solution, 255–256
- workflow changes, 272–273
- TransactionScope activity, 308, 311
- TransactionScopeActivity class, 273
- Trigger activities, 173
- Trigger property, 173
- True branch, FlowDecision activity, 36
- TrueLabel property, 36
- Try section, 70, 72, 77, 322
- TryCatch activity, 70, 77, 78, 322–323, 338, 341, 342, 344, 371, 373, 374
- TypeArgument property, 400

■ U

- UnAssignQueueInstance activity,
 - UserTasks project, 415
- UnderEvaluation property,
 - OperatorConfig class, 379
- Unloaded event handler, 135
- Update command, 389
- UpdateControls() method, 262, 263
- UpdateCounter.cs file, 397
- UpdateCounters activity, 398
- UpdateCounters.cs file, 397
- UpdateLead() method, 262, 263, 273
- UpdateRequest activity, 423, 428
- UserTasks project, 413–420
 - activities, 415
 - CompleteInstance activity, 415–416
 - database design, 414
 - QCPolicy activity, 416–418
 - tracking workflow events, 418–420

- UserTasks.dbml file, UserTasks project, 414

■ V

- Validate() method, 302
- Value property, 11, 28, 55
- Values property, 400
- Variable class, 28
 - getting data from, 27
- Variable type, 156, 163
- Variable type drop-down list, 337, 338
- Variable type field, 83
- Variable type property, 156
- variables, 5, 8–9
 - persisting, 207
- Variables button, 8
- Variables collection, 337, 338
- Variables control, 156
- Variables link, 337
- Variables list, 338
- Variables window, VS 2010 IDE, 8–10
- versions of WF, 21–22
- View message link, 157, 158

■ W

- WaitForInput activity, 139, 141, 193, 195, 218, 247, 272, 273, 311
- WaitForInput class, 135
- WCF (Windows Communication Foundation), 95, 151
- WCF Test Client, 161
- WCF Workflow Service Application
 - template, 151
- WCF Workflow Service template, 163
- Web Service Definition Language (.wsdl)
 - file, 421
- web services, 151–174
 - client workflow, 168–172
 - defining, 170–171
 - implementing host application, 171, 172
 - running application, 172

- Pick activity, 173–174
- using parameters to pass data, 162–168
 - modified PerformLookup activity, 166
 - second workflow service, 163–165
 - testing, 167–168
- workflow service, 151–162
 - defining service contract, 152–155
 - PerformLookup activity, 158–161
 - ReceiveRequest activity, 155–158
 - SendResponse activity, 155–158
 - testing, 161–162
- web.config file, 406, 431, 434, 435, 436
- Wedding project
 - cancellation handlers, 331–332
 - CompensableActivity, 324
 - configuring TryCatch activity, 322–323
 - customizing compensation and confirmation, 337–342
 - designing compensation handlers, 332–336
 - designing Invitations activity, 328–329
 - designing Reception activity, 327–328
 - designing Wedding activity, 325–326
 - modifying application, 320–321
 - running application, 330
 - using Parallel activity, 323
- WF (Workflow Foundation) 3.5. *See* workflow 3.5
- WF (Workflow Foundation) 4.0, 21–22. *See also* workflow 4.0
- WF 3.5 workflow designer, 363, 364
- WF 4.0 designer, VS 2010 IDE, 5
- While activity, 12, 426, 428
- Window1.xaml file, 170, 173
- Windows Communication Foundation (WCF), 95, 151
- Windows Presentation Foundation (WPF) project, 123–127
 - defining window form, 125–127
 - reusing classes from Chapter 8, 124–125
- WorkAssignment workflow class, 308–311
- workflow 3.5, creating, 362–365
- workflow 4.0
 - creating, 366–367
 - interoperability with workflow 3.5, 361–374
 - creating 4.0 workflow, 361–367
 - executing custom 3.5 activity, 367–374
- workflow application
 - creating
 - configuring arguments, 400
 - custom activity, 393–396
 - implementing console application, 402
 - incrementing activity counters, 396–399
 - main workflow, 400
 - overview, 392–404
 - running, 403–404
- Workflow Console Application template, 168
- Workflow Foundation (WF) 3.5. *See* workflow 3.5
- Workflow Foundation (WF) 4.0, 21–22. *See also* workflow 4.0
- workflow project, 405–437
 - configuring database, 405–406
 - running application, 406–413
 - logging in, 407
 - processing requests, 409–413
 - submitting request, 408
 - tracking workflow, 413
- service layer, 420–437
 - activities, 423
 - correlation, 428–430
 - database design, 422–423
 - service contract, 420–421
 - using WorkflowServiceHost, 430–437
 - workflow design, 424–428
- UserTasks project, 413–420
 - activities, 415
 - CompleteInstance activity, 415–416
 - database design, 414
 - QCPolicy activity, 416–418
 - tracking, 418–420
- workflow service, 151–162
 - defining service contract, 152–155

- PerformLookup activity, 158–161
- ReceiveRequest activity, 155–158
- SendResponse activity, 155–158
- testing, 161–162
- Workflow1 class, 7
- Workflow1.asmx file, 7
- Workflow1.xaml file, 4, 48, 322, 400
- Workflow35 assembly, 372
- WorkflowApplication class, 210, 321
- WorkflowApplicationCompletedEventArgs class, 219
- WorkflowExtensions collection, 433
- WorkflowID, 200
- workflowIdle behavior, 435
- WorkflowInstance class, 233
- WorkflowInstance constructor, 142
- WorkflowInstanceId, 259
- WorkflowInstanceQuery class, 235
- WorkflowInstanceRecord class, 232, 245
- WorkflowInvoker class, 25, 58, 116–118, 143, 321, 349
- WorkflowRuntime class, 22
- WorkflowService, 300–304
- WorkflowServiceHost
 - adding app.config file, 312–313
 - adding LeadResponse project, 290–294
 - ApplicationInterface class, 311
 - defining workflows
 - CompleteAssignment class, 304
 - EnterLead workflow modifications, 304–308
 - Persist activity, 311
 - WorkAssignment workflow class, 308–311
 - Language-Integrated Query (LINQ)
 - conflict, 313–314
 - running applications, 314
 - setting up solution, 289–290
 - in workflow project, 430–437
 - configuring extensions, 434
 - persistence extension, 434–435
 - tracking extension, 435–437
 - writing extensions, 431–434
- WorkflowService, 300–304
- WorkflowServiceHost class, 114–115
- WorkflowServiceImplmentation, 115
- WPF (Windows Presentation Foundation)
 - project, 123–127
 - defining window form, 125–127
 - reusing classes from Chapter 8, 124–125
- WPF Application template, 123, 177
- Write() method, 131
- WriteLine activity, 5, 6, 22, 76, 110, 140, 173, 195, 323, 325, 326, 327, 328, 329, 332, 333, 334, 341, 362, 366, 367, 373, 395, 400
- WriteLine() method, 128
- .wsdl (Web Service Definition Language)
 - file, 421

X, Y

- xaml (eXtensible Application Markup Language), 4
- .xaml file, 17
- XAML tab, 291
- .xamlx file, 162, 424
- x:Null attribute, 53
- XPath query, 430

Z

- zoom control, VS 2010 IDE, 5