Introduction

The Windows Communication Foundation, which was code-named Indigo, is a technology that allows pieces of software to communicate with one another. There are many other such technologies, including the Component Object Model (COM) and Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), Microsoft Message Queuing (MSMQ), and WebSphere MQ. Each of those works well in a particular scenario, not so well in others, and is of no use at all in some cases. The Windows Communication Foundation is meant to work well in any circumstance in which a Microsoft .NET assembly must exchange data with any other software entity. In fact, the Windows Communication Foundation is meant to always be the very best option. Its performance is at least on par with that of any other alternative and is usually better; it offers at least as many features and probably several more. It is certainly always the easiest solution to program.

Concretely, the Windows Communication Foundation consists of a small number of .NET libraries with several new sets of classes that it adds to the Microsoft .NET Framework class library, for use with version 2.0 and later of the .NET Common Language Runtime. It also adds some facilities for hosting Windows Communication Foundation solutions to the 5.1 and later versions of Internet Information Services (IIS), the web server built into Windows operating systems.

The Windows Communication Foundation is distributed free of charge as part of a set that includes several other technologies, including the Windows Presentation Foundation, which was code-named Avalon, Windows CardSpace, which was code-named InfoCard, and the
Windows Workflow Foundation. Prior to its release, that group of technologies was called WinFX, but it was renamed the .NET Framework 3.0 in June 2006. Despite that name, the .NET Framework 3.0 and 3.5 is still primarily just a collection of classes added to the .NET Framework 2.0 for use with the 2.0 version of the .NET Common Language Runtime, along with some enhancements to the Windows operating system, as shown in Figure I.1.

FIGURE I.1 The .NET Framework 3.0.

You can install the .NET Framework 3.0 and 3.5 on Windows XP Service Pack 2, Windows Server 2003, and Windows Server 2003 R2. The runtime components are preinstalled on Windows Vista. On Windows Server 2008 you can add the .NET Framework 3.0 via the Application Server Foundation role service. Only a very small number of features of the .NET Framework 3.0 are available exclusively on Windows Vista and later operating systems.

The .NET Framework 3.5 builds incrementally on top of .NET Framework 3.0. Features relevant to this book include web protocol support for building Windows Communication Foundation services, including AJAX, JSON, REST, POX, RSS and ATOM, workflow-enabled services and full tooling support in Visual Studio 2008. During development, .the .NET Framework 3.5 was factored into “red” bits and “green” bits. The red bits were features from .NET Framework 3.0 and the goal was to provide Service Pack levels of compatibility. All the code that worked in 3.0 will work in 3.5. The green bits provide new, additional functionality. Again, the addition of an assembly containing new functionality should have no effect on existing code. The bottom line is that all the code in this book will work with .NET Framework 3.5 and all the code in this book (except the new features introduced in .NET Framework 3.5) should work in .NET Framework 3.0.

This book does not serve as an encyclopedic reference to the Windows Communication Foundation. Instead, it provides the understanding and knowledge required for most practical applications of the technology.
The book explains the Windows Communication Foundation while showing how to use it. So, typically, each chapter provides the precise steps for building a solution that demonstrates a particular aspect of the technology, along with a thorough explanation of each step. Readers who can program in C#, and who like to learn by doing, will be able to follow the steps. Those who prefer to just read will get a detailed account of the features of the Windows Communication Foundation and see how to use them.

To follow the steps in the chapters, you should have installed any version of Visual Studio 2005 or 2008 that includes the C# compiler. Free copies are available at http://msdn.microsoft.com/vstudio/express/. You should also have IIS, ASP.NET, and MSMQ installed.

The .NET Framework 3.0 or 3.5 is required, as you might expect. You can download them from http://www.microsoft.com/downloads/. The instructions in the chapters assume that all the runtime and developer components of the .NET Framework 3.0 or 3.5 have been installed. It is the runtime components that are preinstalled on Windows Vista and that can be added via the Server Manager on Windows Server 2008. The developer components consist of a Software Development Kit (SDK) and two enhancements to Visual Studio 2005. The SDK provides documentation, some management tools, and a large number of very useful samples. The enhancements to Visual Studio 2005 augment the support provided by IntelliSense for editing configuration files, and provide a visual designer for Windows Workflow Foundation workflows. These features are included in Visual Studio 2008.

To fully utilize Windows CardSpace, which is also covered in this book, you should install Internet Explorer 7. Internet Explorer 7 is also available from http://www.microsoft.com/downloads.

Starting points for the solutions built in each of the chapters are available for download from the book’s companion page on the publisher’s website, as well as from http://www.cryptmaker.com/WindowsCommunicationFoundationUnleashed. To ensure that Visual Studio does not complain about the sample code being from a location that is not fully trusted, you can, after due consideration, right-click the downloaded archive, choose Properties from the context menu, and click on the button labeled Unblock, shown in Figure I.2, before extracting the files from the archive.

Note that development on the Vista operating system is supported for Visual Studio 2008 and for Visual Studio 2005 with the Visual Studio 2005 Service Pack 1 Update for Windows Vista. This update is also available from http://www.microsoft.com/downloads. Developers working with an earlier version of Visual Studio 2005 on the Vista operating system should anticipate some compatibility issues. To minimize those issues, they can do two things. The first is to disable Vista’s User Account Protection feature. The second is to always start Visual Studio 2005 by right-clicking on the executable or the shortcut, selecting Run As from the context menu that appears, and selecting the account of an administrator from the Run As dialog.
As with the .NET Framework 3.5 when compared to the .NET Framework 3.0, this book is very similar to its predecessor. Changes include the addition of Visual Studio 2008 support and Chapter 3, “Data Representation and Durable Services,” now covers durable services. The chapters on Windows CardSpace show the updated user interface and cover new features. Chapter 18, “Representational State Transfer and Plain XML Services,” on REST and POX, includes details on the new syndication and JSON APIs. Perhaps the most significant change is a complete rewrite of Chapter 6, “Using the Windows Communication Foundation and the Windows Workflow Foundation Together,” covering the much improved integration between Windows Workflow Foundation and the Windows Communication Foundation.


A particular debt of gratitude is owed to John Lambert for reviewing the drafts. No one is better qualified to screen the text of a book on a programming technology than an experienced professional software tester. Any mistakes in the pages that follow are solely the fault of the writers, however.

The authors are especially grateful for the support of their wives. They are Marta MacNeill, Kathryn Mercuri, Sylvie Watling, and Libby Winkler. Matt, the only parent so far, would also like to thank his daughter, Grace.
To properly understand and work effectively with the Windows Communication Foundation, you should be familiar with certain facilities of the 2.0 versions of the .NET Framework and the .NET common language runtime. This chapter introduces them: partial types, generics, nullable value types, the Lightweight Transaction Manager, and role providers. The coverage of these features is not intended to be exhaustive, but merely sufficient to clarify their use in the chapters that follow.

**Partial Types**

Microsoft Visual C# 2005 allows the definition of a type to be composed from multiple partial definitions distributed across any number of source code files for the same module. That option is made available via the modifier `partial`, which can be added to the definition of a class, an interface, or a struct. Therefore, this part of the definition of a class

```csharp
public partial MyClass
{
    private string myField = null;

    public string MyProperty
    {
        get
        {
            return this.myField;
        }
    }
}
```
and this other part

```csharp
public partial MyClass
{
    public MyClass()
    {
    }

    public void MyMethod()
    {
        this.myField = "Modified by my method.";
    }
}
```
can together constitute the definition of the type MyClass. This example illustrates just one use for partial types, which is to organize the behavior of a class and its data into separate source code files.

## Generics

"Generics are classes, structures, interfaces, and methods that have placeholders for one or more of the types they store or use" (Microsoft 2006). Here is an example of a generic class introduced in the System.Collections.Generic namespace of the .NET Framework 2.0 Class Library:

```csharp
public class List<T>

Among the methods of that class is this one:

```csharp
public Add(T item)
```

Here, T is the placeholder for the type that an instance of the generic class System.Collections.Generic.List<T> will store. In defining an instance of the generic class, you specify the actual type that the instance will store:

```csharp
List<string> myListOfStrings = new List<string>();
```

Then you can use the Add() method of the generic class instance like so:

```csharp
myListOfStrings.Add("Hello, World");
```

Generics enable the designer of the List<T> class to define a collection of instances of the same unspecified type—in other words, to provide the template for a type-safe collection. A user of List<T> can employ it to contain instances of a type of the user's choosing, without the designer of List<T> having to know which type the user might choose. Note as well that whereas a type derived from a base type is meant to derive some of the functionality it requires from the base, with the remainder still having to be programmed, List<string> comes fully equipped from List<T>.
The class, `System.Collections.Generic.List<T>`, is referred to as a *generic type definition*. The placeholder, `T`, is referred to as a *generic type parameter*. Declaring

```csharp
List<string> myListOfStrings;
```

yields `System.Collections.Generic.List<string>` as a *constructed type*, and `string` as a *generic type argument*.

Generics can have any number of generic type parameters. For example, `System.Collections.Generic.Dictionary<TKey,TValue>` has two.

The designer of a generic may use constraints to restrict the types that can be used as generic type arguments. This generic type definition

```csharp
public class MyGenericType<T> where T: IComparable, new()
```

constrains the generic type arguments to types with a public, parameter-less constructor that implements the `IComparable` interface. This less restrictive generic type definition

```csharp
public class MyGenericType<T> where T: class
```

merely constrains generic type arguments to reference types. Note that `T: class` includes both classes and interfaces.

Both generic and nongeneric types can have generic methods. Here is an example of a nongeneric type with a generic method:

```csharp
using System;
public class Printer
{
    public void Print<T>(T argument)
    {
        Console.WriteLine(argument.ToString());
    }
}
static void Main(string[] arguments)
{
    Printer printer = new Printer();
    printer.Print<string>("Hello, World");
    Console.WriteLine("Done");
    Console.ReadKey();
}
```

In programming a generic, it is often necessary to determine the type of generic argument that has been substituted for a generic type parameter. This revision to the preceding example shows how you can make that determination:
A generic interface may be implemented by a generic type or a nongeneric type. Also, both generic and nongeneric types may inherit from generic base types.

```csharp
public interface IMyGenericInterface<T>
{
    void MyMethod(T input);
}

public class MyGenericImplementation<T>: IMyGenericInterface<T>
{
    public void MyMethod(T input)
    {
    }
}

public class MyGenericDescendant<T> : MyGenericImplementation<T>
{
}

public class MyNonGenericImplementation : IMyGenericInterface<string>
{
    public void MyMethod(string input)
    {
    }
}
```
Nullable Value Types

According to the Common Language Infrastructure specification, there are two ways of representing data in .NET: by a value type or by a reference type (Ecma 2006, 18). Although instances of value types are usually allocated on a thread’s stack, instances of reference types are allocated from the managed heap, and their values are the addresses of the allocated memory (Richter 2002, 134–5).

Whereas the default value of a reference type variable is null, indicating that it has yet to be assigned the address of any allocated memory, a value type variable always has a value of the type in question and can never have the value null. Therefore, although you can determine whether a reference type has been initialized by checking whether its value is null, you cannot do the same for a value type.

However, there are two common circumstances in which you would like to know whether a value has been assigned to an instance of a value type. The first is when the instance represents a value in a database. In such a case, you would like to be able to examine the instance to ascertain whether a value is indeed present in the database. The other circumstance, which is more pertinent to the subject matter of this book, is when the instance represents a data item received from some remote source. Again, you would like to determine from the instance whether a value for that data item was received.

The .NET Framework 2.0 incorporates a generic type definition that provides for cases like these in which you want to assign null to an instance of a value type, and test whether the value of the instance is null. That generic type definition is System.Nullable<T>, which constrains the generic type arguments that may be substituted for T to value types. Instances of types constructed from System.Nullable<T> can be assigned a value of null; indeed, their values are null by default. Thus, types constructed from System.Nullable<T> are referred to as nullable value types.

System.Nullable<T> has a property, Value, by which the value assigned to an instance of a type constructed from it can be obtained if the value of the instance is not null. Therefore, you can write

```csharp
System.Nullable<int> myNullableInteger = null;
myNullableInteger = 1;
if (myNullableInteger != null)
{
    Console.WriteLine(myNullableInteger.Value);
}
```
The C# programming language provides an abbreviated syntax for declaring types constructed from `System.Nullable<T>`. That syntax allows you to abbreviate

```csharp
System.Nullable<int> myNullableInteger;
```
to

```csharp
int? myNullableInteger;
```
The compiler will prevent you from attempting to assign the value of a nullable value type to an ordinary value type in this way:

```csharp
int? myNullableInteger = null;
int myInteger = myNullableInteger;
```
It prevents you from doing so because the nullable value type could have the value `null`, which it actually would have in this case, and that value cannot be assigned to an ordinary value type. Although the compiler would permit this code,

```csharp
int? myNullableInteger = null;
int myInteger = myNullableInteger.Value;
```
the second statement would cause an exception to be thrown because any attempt to access the `System.Nullable<T>.Value` property is an invalid operation if the type constructed from `System.Nullable<T>` has not been assigned a valid value of `T`, which has not happened in this case.

One proper way to assign the value of a nullable value type to an ordinary value type is to use the `System.Nullable<T>.HasValue` property to ascertain whether a valid value of `T` has been assigned to the nullable value type:

```csharp
int? myNullableInteger = null;
if (myNullableInteger.HasValue)
{
    int myInteger = myNullableInteger.Value;
}
```
Another option is to use this syntax:

```csharp
int? myNullableInteger = null;
int myInteger = myNullableInteger ?? -1;
```
by which the ordinary integer `myInteger` is assigned the value of the nullable integer `myNullableInteger` if the latter has been assigned a valid integer value; otherwise, `myInteger` is assigned the value of `-1`.

## The Lightweight Transaction Manager

In computing, a transaction is a *discrete* activity—an activity that is completed in its entirety or not at all. A resource manager ensures that if a transaction is initiated on some
resource, the resource is restored to its original state if the transaction is not fully completed. A distributed transaction is one that spans multiple resources and therefore involves more than a single resource manager. A manager for distributed transactions has been incorporated into Windows operating systems for many years. It is the Microsoft Distributed Transaction Coordinator.

.NET Framework versions 1.0 and 1.1 provided two ways of programming transactions. One way was provided by ADO.NET. That technology's abstract System.Data.Common.DbConnection class defined a BeginTransaction() method by which you could explicitly initiate a transaction controlled by the particular resource manager made accessible by the concrete implementation of DbConnection. The other way of programming a transaction was provided by Enterprise Services. It provided the System.EnterpriseServices.Transaction attribute that could be added to any subclass of System.EnterpriseServices.ServicedComponent to implicitly enlist any code executing in any of the class's methods into a transaction managed by the Microsoft Distributed Transaction Coordinator.

ADO.NET provided a way of programming transactions explicitly, whereas Enterprise Services allowed you to do it declaratively. However, in choosing between the explicit style of programming transactions offered by ADO.NET and the declarative style offered by Enterprise Services, you were also forced to choose how a transaction would be handled. With ADO.NET, transactions were handled by a single resource manager, whereas with Enterprise Services, a transaction incurred the overhead of involving the Microsoft Distributed Transaction Coordinator, regardless of whether the transaction was actually distributed.

.NET 2.0 introduced the Lightweight Transaction Manager, System.Transactions.TransactionManager. As its name implies, the Lightweight Transaction Manager has minimal overhead: “...[p]erformance benchmarking done by Microsoft with SQL Server 2005, comparing the use of a [Lightweight Transaction Manager transaction] to using a native transaction directly found no statistical differences between using the two methods” (Lowy 2005, 12). If only a single resource manager is enlisted in the transaction, the Lightweight Transaction Manager allows that resource manager to manage the transaction and the Lightweight Transaction Manager merely monitors it. However, if the Lightweight Transaction Manager detects that a second resource manager has become involved in the transaction, the Lightweight Transaction Manager has the original resource manager relinquish control of the transaction and transfers that control to the Distributed Transaction Coordinator. Transferring control of a transaction in progress to the Distributed Transaction Coordinator is called promotion of the transaction.

The System.Transactions namespace allows you to program transactions using the Lightweight Transaction Manager either explicitly or implicitly. The explicit style uses the System.Transactions.CommitableTransaction class:

CommitableTransaction transaction = new CommitableTransaction();
using(SqlConnection myConnection = new SqlConnection(myConnectionString))
The alternative, implicit style of programming, which is preferable because it is more flexible, uses the System.Transactions.TransactionScope class:

```csharp
using(TransactionScope scope = new TransactionScope)
{
    //Do transactional work:
    //...
    //Since no errors have occurred, commit the transaction:
    scope.Complete();
}
```

This style of programming a transaction is implicit because code that executes within the using block of the System.Transactions.TransactionScope instance is implicitly enrolled in a transaction. The Complete() method of a System.Transactions.TransactionScope instance can be called exactly once, and if it is called, the transaction will commit.

The System.Transactions namespace also provides a means for programming your own resource managers. However, knowing the purpose of the Lightweight Transaction Manager and the implicit style of transaction programming provided with the System.Transactions.TransactionScope class will suffice for the purpose of learning about the Windows Communication Foundation.

### Role Providers

Role providers are classes that derive from the abstract class System.Web.Security.RoleProvider. That class has the interface shown in Listing 1.1. It defines ten simple methods for managing roles, including ascertaining whether a given user has been assigned a particular role. Role providers, in implementing those abstract methods, will read and write a particular store of role information. For example, one of the concrete implementations of System.Web.Security.RoleProvider included in the .NET Framework 2.0 is System.Web.Security.AuthorizationStoreRoleProvider, which uses an Authorization Manager Authorization Store as its repository of role information. Another concrete implementation, System.Web.Security.SqlRoleProvider, uses a SQL Server database as its store. However, because the System.Web.Security.RoleProvider has
such a simple set of methods for managing roles, if none of the role providers included in
the .NET Framework 2.0 is suitable, you can readily provide your own implementation to
use whatever store of role information you prefer. Role providers hide the details of how
role data is stored behind a simple, standard interface for querying and updating that
information. Although System.Web.Security.RoleProvider is included in the System.Web
namespaces of ASP.NET, role providers can be used in any .NET 2.0 application.

LISTING 1.1 System.Web.Security.RoleProvider

```csharp
public abstract class RoleProvider : ProviderBase
{
    protected RoleProvider();

    public abstract string ApplicationName { get; set; }

    public abstract void AddUsersToRoles(
        string[] usernames, string[] roleNames);
    public abstract void CreateRole(
        string roleName);
    public abstract bool DeleteRole(
        string roleName, bool throwOnPopulatedRole);
    public abstract string[] FindUsersInRole(
        string roleName, string usernameToMatch);
    public abstract string[] GetAllRoles();
    public abstract string[] GetRolesForUser(
        string username);
    public abstract string[] GetUsersInRole(
        string roleName);
    public abstract bool IsUserInRole(
        string username, string roleName);
    public abstract void RemoveUsersFromRoles(
        string[] usernames, string[] roleNames);
    public abstract bool RoleExists(string roleName);
}
```

The static class, System.Web.Security.Roles, provides yet another layer of encapsulation
for role management. Consider this code snippet:

```csharp
if (!Roles.IsUserInRole(userName, "Administrator"))
{
    [...]
}
```

Here, the static System.Web.Security.Roles class is used to inquire whether a given user
has been assigned to the Administrator role. What is interesting about this snippet is that
the inquiry is made without an instance of a particular role provider having to be created
first. The static `System.Web.Security.Roles` class hides the interaction with the role provider. The role provider it uses is whichever one is specified as being the default in the configuration of the application. Listing 1.2 is a sample configuration that identifies the role provider named `MyRoleProvider`, which is an instance of the `System.Web.Security.AuthorizationStoreRoleProvider` class, as the default role provider.

**LISTING 1.2 Role Provider Configuration**

```xml
<configuration>
  <connectionStrings>
    <add name="AuthorizationServices"
         connectionString="msxml://~\App_Data\SampleStore.xml" />
  </connectionStrings>
  <system.web>
    <roleManager defaultProvider="MyRoleProvider"
                 enabled="true"
                 cacheRolesInCookie="true"
                 cookieName=".ASPROLES"
                 cookieTimeout="30"
                 cookiePath="/"
                 cookieRequireSSL="false"
                 cookieSlidingExpiration="true"
                 cookieProtection="All" >
      <providers>
        <clear />
        <add
          name="MyRoleProvider"
          connectionStringName="AuthorizationServices"
          applicationName="SampleApplication"
          cacheRefreshInterval="60"
          scopeName="" />
      </providers>
    </roleManager>
  </system.web>
</configuration>
```

**Summary**

This chapter introduced some programming tools that were new in .NET 2.0 and that are prerequisites for understanding and working effectively with the Windows Communication Foundation:

- The new `partial` keyword in C# allows the definitions of types to be composed from any number of parts distributed across the source code files of a single module.
Generics are templates from which any number of fully preprogrammed classes can be created.

Nullable value types are value types that can be assigned a value of `null` and checked for `null` values.

The Lightweight Transaction Manager ensures that transactions are managed as efficiently as possible. An elegant new syntax has been provided for using it.

Role providers implement a simple, standard interface for managing the roles to which users are assigned that is independent of how the role information is stored.

References


Index

A

Abort() method, 701
abstract base classes, 22-23
Access Control Entries, 374
access control lists, 374
ACE (Access Control Entries), 374
ACID transactions, activities (Windows Workflow Foundation), 170
ACL (access control lists) versus claim-based authorization, 374-375
Action parameter (OperationContract attributes), 393, 547
Activate() method, streamed transfer mode, 563-565
activities (Windows Workflow Foundation), 149
ACID transactions, 170
binding, 156-157
CAG activity, 202-203
compensation, 171-172
custom activities
  adding properties, 153-154
  basic activities, 152-158
  calling services via, 210-211, 214
  composite activities, 158-160
  DependencyProperty property, 155-157
  root activities, 184
  XAML properties, 154
Delay activity, state machine workflows, 183
design behavior, 167-170
error handling, 149
EventDrivenActivity, state machine workflows, 183
HandleExternalEvent, state machine workflows, 183
IfElse, sequential workflows, 175
initializing, 149
InvokeWebService activity, consuming services, 214
lifecycle of, 149
out of the box activities, 151-152
Parallel activity, 149, 176-183, 248
PlaceOrder activity, 149
promoting properties, 157-158
Receive activity, 233
exposing workflows as services, 226-229, 245-247
multiple client conversations, 245-247
Replicator activity, 248
Send activity
BeforeSend event handler, 248
consuming services, 214-219
endpoint creation via, 217-219
exposing workflows as services, 240, 243, 248
multiple client conversations, 248
service address updates via, 219
SetState activity, state machine workflows, 183
tracing, 640-642, 696-698
validation, 168, 170
workflow communication
host configuration, 163-166
interface, 160-163
AddItem messages, defining service contracts, 582
addresses, 28-30
DerivativesCalculator example, 44-51
dynamic updates via Send activity, 219
endpoints, 85
Windows Communication Foundation, 679-680
ADOM.NET transactions, initiating, 15
Ajax (Asynchronous JavaScript and XML), 615-619, 703-705
Announce messages, defining service contracts, 582
anonymous identity, 292
anticipating exceptions, 701-702
API layers (frameworks), designing public layers, 685
Approval activity, 159
architectures (service-oriented), defining, 25
.asmx extension, 68
ASP.NET 2.0, enabling in CardSpace (Windows), 333
ASP.NET Compatibility Mode, 695
assertions (policies), 670
ATOM, syndicating in NET Framework 3.5, 609-615
attaching custom behaviors to operations/ endpoints, 457-459, 480-481
auditing security events, 698-699
authentication
CardSpace (Windows), 313-319
COM+, 418
FFIEC (Federal Financial Institutions Examination Council), 287
Identity
Identity Metasystem, 291-298, 319
Laws of Identity, 289-291
regulatory compliance, 288
role of, 285-287
user-centric identity management, 290
Identity Metasystem, Information Cards, 298-306
Kerberos authentication method, 313
Password (Microsoft), 289
passwords, 287
PKI (Public Key Infrastructure), 289
SSO (single sign on), 288, 319
authorization

claim-based versus role-based, 373-374
COM+, 418
context (claim sets), 373, 395
defining, 252
domains, 373
FFIEC (Federal Financial Institutions Examination Council), 287
identity
      Identity Metasystem, 291-298, 319
      Laws of Identity, 289-291
      regulatory compliance, 288
      role of, 285-287
      user-centric identity management, 290
Identity Metasystem, Information Cards, 298-306
Password (Microsoft), 289
passwords, 287
PKI (Public Key Infrastructure), 289
policies
      authorization domains, 373
      claim sets, 373, 395
      resource access, Identity (Windows), 377-383
security, 272, 275, 281
      configuring services to use role providers, 273-274
      service configuration, 276-280
ServiceAuthorizationManager implementation, 279-280
SSO (single sign on), 288, 319
Authorization Manager
      Windows Server 2003, authorizing resource access, 379-383
      XSI, claim-based authorization, 384-391
Axis, WSDL2Java, 65, 67

B

backwards-compatible changes (versioning), 662
base addresses, 50-51
BasicHttpBinding, 48-50, 446, 554, 557, 681
BeforeSend event handler, 248
behavioral contracts, 689
behaviors
      custom behaviors
        attaching to operations/endpoints, 457-459, 480-481
defining, 453-458
dispatchers, 452
informing Windows Communication Foundation of, 457-466, 481-482
instance context provider, 476-477
instance provider, 477-478
message formatter, 471-473
message inspector, 474-476
operation invokers, 478-479
operation selector, 467-468
parameter inspector, 469-471
usage examples, 482-483
WSDL export extension, 479-482
Bind dialog (Windows Workflow Foundation), 157
binding, 28-30
      activities (Windows Workflow Foundation), 156-157
      DerivativesCalculator example, 44-51
      Peer Channel, 570-573
binding (WSDL), 28
binding elements, 48
custom binding elements
      applying through configuration, 508-511

How can we make this index more useful? Email us at indexes@samspublishing.com
inbound communication support, 502-507
outbound communication support, 495-501
understanding starting points, 493-494
inbound communication, 487-488
outbound communication, 486
**BindingContext property, 491**

**bindings**
changing, versioning, 669
endpoints, 85
HTTP bindings, MessageEncoding property, 706
REST POX service endpoints, 602-603

**Windows Communication Foundation**
activating/deactivating binding features, 683
custom message inspectors, 684
custom operation selectors, 684
custom peer resolver services, 683
HTTP bindings, 682
identifying services with contended resources, 681
predefined bindings, 681-682
queueing service requests, 681

**brokered notifications, 537**

**browsers**
architectures of, 600
Information Cards, adding to, 353-363

**buffered transfer mode, 553**

**BufferedDataSubscriptionManager, 565**

**BufferedDataSubscriptionManager class, 564**

**BufferedSubscriptionManager class, 562-565**

**BuildChannelFactory<T>, 500**

C

**C++**
abstract base classes, 22-23
capsulation, 21-22
interfaces, defining, 22-23
messages, defining, 22-23

**CAG (ConditionedActivityGroup) activity, 202-203**

**CalculateDerivative() operation, 86**

**callback contracts, 538-544**

**CallExternalMethod activity, workflow communication, 160, 165-166**

**CardId, 317**

**CardSpace (Windows), 298-299, 306, 319, 321-322**
architecture of, 306-309
ASPNET 2.0, enabling, 333
authentication methods, 313-319
cards
adding/installing, 300-303
editing, 304-305
metadata, 305
Choose a Card dialog, 311
exceptions, catching, 348-350
GetToken() method, 308
HTTP, 325-326, 336-337, 370
IIS (Internet Information Services) setup, 333-335

**Information Cards**
adding to browser applications, 353-363
adding to WCF applications, 337-348
building STS, 367-369
creating Managed Cards, 364-367
processing security tokens, 350-351
Introduction dialog, 308-310
IssuedToken policy assertion, 307
Metadata Resolver, 351-353
passwords, 312
preview dialog, 300
protocols, 311
security tokens, 305, 313
site information page, 343
usage example, 331-332
Windows clients, building, 330-331
X.509 certificates, 333
host file updates, 334-335
importing into certificate store, 334
private key access, 336
cardspace.db (CardSpace), 307
centralized lifecycle management, 670-672
certificate store (CardSpace), importing X.509 certificates to, 334
certificates, transport security
installing, 253-255
server provided certificates, identifying, 255-256
specifying for
HTTP.SYS usage in SSL exchanges, 256
IIS usage in SSL exchanges, 255-256
SSL configuration, removing from
HTTP.SYS, 283
IIS, 282
trustworthiness, determining, 253-255
uninstalling, 281-282
Channel Layer, 27-28, 85, 252
channel listeners, custom transports, 525-527
ChannelFactory<T> generic, MSMQ PGM, 548
ChannelListenerBaseIReplyChannel, 503
ChannelManagerService, endpoint creation via Send activity, 217-218
channels
custom channels
applying through configuration, 508-511
binding elements, 486-488
declaring shapes, 489
inbound communication support, 502-507
matching contracts to, 490-492
outbound communication support, 495-501
session support, 490
shapes of, 488-489
state machines, 492-493
understanding starting points, 493-494
Peer Channel, 567
ChannelToken property, consuming services, 216
Choose a Card dialog (CardSpace), 311
claim sets (security)
authorization context, 373, 395
description of, 372
evaluation context, 373, 395
STS, 375
WS-Trust, 375
claims (security)
claim-based authorization, 411
ACL versus, 374-375
adoption strategies, 375-376
resource access, authorizing, 384-391
role-based authorization versus, 373-374
description of, 372
federated security versus, 412
Identity Metasystem, 292
normalization, 376
rights of, 372
SAML, 375
types of, 372
values of, 372
Claims section (ini files), Managed Cards, 366
class file format, 23
class libraries, building service types, 693
classes, representation in XML, 95

How can we make this index more useful? Email us at indexes@samspublishing.com
client applications, configuring via Service Configuration Editor, 631-632
client message formatters, 454
Client project, streamed transfer mode, 553
Client.csproj console application, 341
clients
  building, 95
data contracts, defining, 106
DerivativesCalculator example, 55-60
coding alternatives, 60, 63
Java clients, 65-67
Windows Communication Foundation
  AJAX, 703-705
  anticipating exceptions, 701-702
  asynchronous pattern for operation contracts, 699-701
  lifetime management, 703
  Open() method, 702
  scoping service hosts/clients, 703
ClientViewOfData class, defining data contracts, 106
Close() method, 692-693, 701
COM, 23
COM+
  authentication, 418
  authorization, 418
  component methods, modifying automatically, 418
Hosted mode, 419
integrating with
  COM+ Hosted mode, 419
  exposing component interfaces as web services, 417-418
  supported interfaces, 418
  Web Hosted In-Process mode, 419
  Web Hosted mode, 419
Service Model Configuration tool (ComSvcConfig.exe), 419, 424
  flags of, 420-421
  modes of, 420
web services
  automatic mapping, 417
  exposing component interfaces as, 417-418, 422-426
  referencing Windows Forms clients, 426-427
  updating, 418
Windows Communication Foundation services, calling via
  building clients, 431-432
  building services, 428-430
  building VBScript files, 433
  testing solutions, 433
Common Intermediate Language, 23
Common Language Infrastructure Specification, 23
Common Language Runtime, 24
communications
  implementing, Peer Channel example, 584-585
  security, basic tasks of, 251-252
communication state machines, 492-493
CommunicationObject property, 499
compensation activities (Windows Workflow Foundation), 171-172
component-oriented programming, COM and, 23
composability (web services), defining, 445-446
composite activities (Windows Workflow Foundation), 158-160
CompositeDuplexBindingElement, 538
conditions, rules as (Windows Workflow Foundation), 200-202
configuration system, 625-627
configuring
  client applications via Service Configuration Editor, 631-632
  role providers, 18
  trade-recording services via Service Configuration Editor, 627-630
**confirmation messages, receiving**, 132

**consuming services**
- custom activities, calling services via, 210-211, 214
  - GetOptionChain method, 216
  - InvokeWebService activity, 214
  - Send activity, 214-219

**context tokens, exposing workflows as services**, 234-237

**contracts**, 28
- callback contracts, 538-544
- contract-first development, 91
- defining for DerivativesCalculator example, 36-42
- endpoints, 85, 104
- explicit communication semantics, 490
- implementing, 29
- implicit communication requirements, 490
- interfaces, designating as, 29
- matching custom channels to, 490-492
- names, altering, 39
- operation contracts, asynchronous pattern for, 699-701
- Peer Channel, 574
  - data contracts, 579-581
  - duplex contracts, 575
  - service contracts, 582-585
- REST POX service endpoints, 603
- Windows Communication Foundation
  - behavioral contracts, 689
  - defining data contracts, 688
  - designing public API layers, 685
  - designing service contracts, 686-687
  - designing via scenarios, 684
  - duplex service contracts, 690-691
  - fault contracts, 689
  - identifying/documenting service model items, 684
  - message contracts, 689
  - Principle of Scenario-Driven Design, 685
  - service contracts, 690-691
  - structural contracts, 687-689
- converting PPID to readable version, 317-318
- Correct member (Peer Channel example), 581
- counterfeit websites, 286
- counters (performance), 647-648, 653-658, 696

**custom activities**
- calling services via, 210-211, 214

**custom activities (Windows Workflow Foundation)**
- activities
  - basic activities, 152-158
  - composite activities, 158-160
- binding, 156-157
- properties
  - adding, 153-154
  - DependencyProperty property, 155, 157
  - promoting, 157-158
  - XAML, 154
- services, calling via, 210-211, 214

**custom behaviors**
- declaring, 453-458
- dispatchers, 452
- endpoints, attaching to, 457-459, 480-481
- instance context providers, service applications, 476-477
- instance providers, service applications, 477-478
- message formatters
  - client applications, 471-472
  - service applications, 472-473
- message inspectors
  - client applications, 474
  - service applications, 475-476
- operation invokers, service applications, 478-479

How can we make this index more useful? Email us at indexes@samspublishing.com
operation selectors
client applications, 467-468
service applications, 468
operations, attaching to, 457-459, 480-481
parameter inspectors
client applications, 469-470
service applications, 470-471
usage examples, 482-483
Windows Communication Foundation, informing of, 457-466, 481-482
WSDL export extension, 479
attaching to operations/endpoints, 480-481
declaring, 480
informing Windows Communication Foundation of, 481-482
custom binding elements
configuration, applying through, 508-511
inbound communication support, 502-507
outbound communication support, 495-501
starting points, understanding, 493-494
custom channels
binding elements
inbound communication, 487-488
outbound communication, 486
configuration, applying through, 508-511
contracts, matching to, 490-492
inbound communication support, 502-507
outbound communication support, 495-501
session support, 490
shapes of, 488-489
state machines, 492-493
starting points, understanding, 493-494
custom composite activities (Windows Workflow Foundation), 160
custom message inspectors, 684
custom operation selectors, 684
custom peer resolver services, 683

custom root activities (Windows Workflow Foundation), 184
custom services, workflow hosting, 196-198
custom streams, streamed transfer mode
implementing via, 561-565
transmitting via, 557-561
custom transports, 513
binding elements, 516-532
channel listeners, 525-527
message encoders, 530-531
TCP protocol support, 520-532
CustomStream class, 557-561

D
data contracts, 87
defining, 106, 688
DerivativesCalculation data contracts, 88-89
exception handling, 110-114
inheritance as versioning, 110
parameters, changing in, 664-668
Peer Channel, 579-581
versioning, 110
data representation
clients, building, 95
dispatchers, 86
exception handling, 110-114
Infosets, 86
Message classes, 86
Message objects, 86
services
building, 92-94
defining, 85
durable services, 114-121
endpoints, 85, 103
stateful services, 115
XML serializers
  System.Runtime.SerializationDataContractSerializer class, 87-110
  System.Xml.Serialization.XmlSerializer class, 87-90

DataContract attribute, durable services, 117
DataContractSerializer class, 87-91
  clients, building, 95
  services, building, 92-94
  usage example, 96-110

DataMember attribute, durable services, 117

DataMember attribute, durable services, 117

DataPoint class, 539

dead-letter queues, 132
debugging applications, 707-708
decision trees, versioning, 662
declaring
  custom behaviors, 453-458, 480
  custom channel shapes, 489

DefaultLoaderService, 194
DefaultWorkflowSchedulerService, 193

Delay activity (Windows Workflow Foundation)
  state machine workflows, 183
  workflow communication, 166
delegating maintenance, Windows Communication Foundation integration, 678-679
deleting operations, versioning, 668

DependencyProperty property, custom activities, 155-157
deploying services, DerivativesCalculator example, 51-55

DerivativesCalculation class, 34
  addresses, 44-51
  binding, 44-51
  clients
    coding alternatives, 60, 63
    using with, 55-60
  contracts, defining, 36-42
  DivideByZero() method, 111
  exception handling, 111
  Java clients, 65-67
  Program.cs file, 57-58
  services
    deploying, 51-55
    hosting, 42-44, 67-72

DerivativesCalculator.csproj class library, 337-338

DerivativesCalculatorService.csproj class library, 338-339

DerivativesCalculatorServiceType.cs, 350-351

DerivedData class,DataContractSerializer class
  usage example, 109

DerivativesCalculation class, 87-88

Details section (ini files), Managed Cards, 366
digital identity (Identity Metasystem), 292-293
dispatch message formatters, 455
dispatchers, 86, 452

DispatchOperation property, 455

Distributed Transaction Coordinator, 15
distributed transactions, 15, 136

DivideByZero() method, exception handling, 111

DNS (domain name servers), CardSpace host file updates, 334-335

DSL (domain-specific languages), 27-28
duplex communication, 683
duplex contracts, 575, 690-691
duplex messaging, exposing workflows as services, 237-243

DuplexChannel property, 489
durable services, 114
 DataContract attribute, 117
 DataMember attribute, 117
  DurableService attribute, 117
  implementing, 115-121

How can we make this index more useful? Email us at indexes@samspublishing.com
email spoofing, 286
EnableSsl property, 706
encapsulation, 21-22
encoders (message), 514
binding elements, implementing, 516-519
custom transports, 530-531
ending/initiating
sessions, 130
transactions, 15
EndpointBehaviors property, 457
EndpointName property, consuming services, 216
endpoints, 44
addresses, 85
bindings, 85
contracts, 85, 104
custom behaviors, attaching to, 457-459, 480-481
defining, 85
message security configuration, 266-269
NET application configuration files, adding to, 626-627
Peer Channel, 569, 574, 585-587
REST POX services
address of, 602
bindings, 602-603
contracts, 603
retiring, versioning, 669-670
Send activity, creating via, 217-219
services
adding to, 103
changing endpoint addresses, 670
Enterprise Services, initiating transactions, 15
error handling, activities (Windows Workflow Foundation), 149
evaluation context (claim sets), 373, 395
EventDrivenActivity activity (Windows Workflow Foundation), state machine workflows, 183
events (security), auditing, 633, 636
exceptions
anticipating, 701-702
CardSpace (Windows), catching in, 348-350
data contracts, 110-114
exchange patterns, designing, 581
Execute() method, custom activities, 152-153
exemplars, Windows Communication Foundation integration, 677
explicit communication semantics, 490
export extension (WSDL), 479
declaring, 480
endpoints, attaching to, 480-481
operations, attaching to, 480-481
Windows Communication Foundation, informing of, 481-482
Extension property, 693-695

Fabrikam certificates, 336, 344
fault contracts, 689
Faulty operation, exception handling, 112-113
federations (security), 320, 376
claim-based security versus, 412
security token services, 321-322
FFIEC (Federal Financial Institutions Examination Council), authentication, 287
Fielding, Dr. Roy, web browser architectures, 600
formatname:MULTICAST, 549
Formatter property, 454-455
Forms (Windows), client references to COM+ web services, 426-427
forward chaining, 204-205
frameworks
   API layers, designing public layers, 685
   layered designs, guidelines for, 451
   Principle of Scenario-Driven Design, 685
framing messages, 514

G - H

generics
   argument type, determining, 11
   base type inheritance, 12
   List<T> class, 10
   methods, 11
   GetCallbackChannel<T>() generic method, call- 
   back contracts, 539
   GetKnownDataPoints() operation, callback con- 
   tracts, 540
   GetOptionChain method, consuming services, 
   216
   GetPicture() operation
      custom streams, 557-561
      streamed transfer mode, 553, 557
   GetToken() method, CardSpace, 308
   Graph tab (Trace Viewer), 644

HandleExternalEvent activity (Windows
   Workflow Foundation)
   state machine workflows, 183
   workflow communication, 160, 165-166
health model services, 696-697
high assurance, 343
high value assurance certificates, 343
Host.csproj console application, 339-340
hosting
   services, 30
      DerivativesCalculator example, 42-44
      IIS, 67-69, 72
Windows Workflow Foundation workflows
   inside WCF services
      message routing, 221-223
      runtime hosting options, 223-226
   workflows, 185-198
Hosting modes (COM+), 419
HTTP (Hypertext Transfer Protocol)
   bindings, 48, 682, 706
   CardSpace (Windows)
      configuring in, 336-337
      Cardspace over, 325-326, 370
   .NET Framework 3.5 support, 324-326
   HTTPSYS, removing SSL configuration, 283

identity
   FFIEC, 287
   Laws of Identity, 289-291
   Passport (Microsoft), 289
   passwords, 287
   PKI, 289
   PPID, 317, 325
   regulatory compliance, 288
   resource access, authorizing, 377-383
   role of, 285-287
   SSO, 288, 319
   theft, 286
   user-centric identity management, 290
Identity Metasystem, 291
   anonymous, 292
   architecture of, 294
   CardSpace (Windows),
      298-299, 306, 319-322
      adding Information Cards to browser
      applications, 353-363

How can we make this index more useful? Email us at indexes@samspublishing.com
adding Information Cards to WCF applications, 337-348
adding/installing cards, 300-303
architecture of, 306-309
ASP.NET 2.0, 333
authentication methods, 313-319
building Windows clients, 330-331
card metadata, 305
catching exceptions, 348-350
Choose a Card dialog, 311
editing cards, 304-305
GetToken() method, 308
host file updates, 334-335
HTTP configuration, 336-337
IIS setup, 333-335
importing into certificate store, 334
Introduction dialog, 308-310
IssuedToken policy assertion, 307
Metadata Resolver, 351-353
passwords, 312
preview dialog, 300
private key access, 336
processing security tokens, 350-351
protocols, 311
security tokens, 305, 313
site information page, 343
usage example, 331-332
X.509 certificates, 333-336
claims, 292
digital identity, 292-293
Identity Providers, 294-295
building, 329
Information Cards, 298
Identity Selector, 297-298
Information Cards, 298-300, 306
adding/installing, 300-303
editing, 304-305
Managed Cards, 301
metadata, 305
Personal Cards, 300-303
Provider Cards, 301
revoking, 311
security tokens, 305
Self-Issued Cards, 300-301, 305
Liberty identity protocols, 296
Relying Parties, 294, 297, 330
requirements for, 295
roles of, 293
security tokens, 293
Triangle of Trust, 319
Windows clients, building, 330-331
WS-Security, 296
WS-Trust, 296
Identity Providers (Identity Metasystem), 294-295
building, 329
Information Cards, 298
Identity Selector
CardSpace (Windows), 298-299, 319-322
adding/installing cards, 300-303
architecture of, 306-309
authentication methods, 313-319
card metadata, 305
Choose a Card dialog, 311
editing cards, 304-305
GetToken() method, 308
Introduction dialog, 308-310
IssuedToken policy, 307
passwords, 312
preview dialog, 300
protocols, 311
security tokens, 305, 313
Identity Metasystem, 297-298
IdP (Identity Providers), OpenID, CardSpace
usage example, 332
IfElse activity (Windows Workflow Foundation), sequential workflows, 175

IIS (Internet Information Services)
  CardSpace (Windows), 333-335
  services, hosting, 67-69, 72, 692
impersonation, 269-272
implicit communication requirements, 490
importing X.509 certificates to CardSpace certificate stores, 334
InfoCard, 298
infocard.exe (CardSpace), 306
infocardapi.dll (CardSpace), 307
Information Cards, 298-300, 306
  adding/installing, 300-303
  browser applications, adding to, 353-363
  building STS (Security Token Services), 367-369
  editing, 304-305
  logins, Windows Live ID support for, 332
  Managed Cards, 301, 364-367
  metadata, 305
  Metadata Resolver, 351-353
  Personal Cards, 300-301, 303
  Provider Cards, 301
  revoking, 311
  security tokens, 305
Self-Issued Cards, 300-301, 305
WCF applications, adding to, 345-346
  changing client credential types, 342
  Client.csproj console application, 341
  DerivativesCalculator.csproj class library, 337-338
  DerivativesCalculatorService.csproj class library, 338-339
  Host.csproj console application, 339-340
  logotypes, 343-344
  security tokens, 343, 350-351
  wsFederationBinding, 347-348
InfoSet (XML Information Set), data representation, 86
ini files (Managed Cards), 366
initiating/ending
  sessions, 130
  transactions, 15
InnerProxy property, ClientBaseT, 63
InputChannel property, 489
instance context provider custom behavior, service applications, 476-477
instance context sessions, 129
instance provider custom behavior, service applications, 477-478
InstanceContextProvider property, 454
InstanceData SQL database, 119
InstanceProvider property, 455
instrumentation
  activity tracing, 640-642
  configuration system, 625-627
  message logging, 637-640
  performance counters, 647-648, 653-658
  security auditing facility, 633, 636
  Service Configuration Editor, configuring client applications, 631-632
  trade-recording services, 627-630
  tools overview, 624-625
  trace listener, 644-647
  Trace Viewer, 643-644
  WMI Provider, 649
    accessing data via Windows PowerShell, 652
    accessing data via WMI CIM Studio, 650
    adding custom performance counters, 653-658
interfaces
  C++, defining with, 22-23
  contracts, designating interfaces as, 29
  .NET, writing in, 29
interoperability, web services, 445-447
Introduction dialog (CardSpace), 308-310
InvokeWebService activity, consuming services, 214
IOrderSubscriber interface, 547
IOrderSubscriber service contracts (MSMQ PGM), 551
IPictureServer
  custom streams, 557-561
  streamed transfer mode, 553, 557
IPublisher, callback contracts, 538-544
IPv6 NAT Traversal, Teredo, 568
ISecurityTokenService service contracts, STS addition process, 393
issued tokens, 343, 350
IssuedToken policy assertions (CardSpace), 307
Issuer section (ini files), Managed Cards, 366
ISubscriber
  callback contracts, 538-544
  stream transfer mode, 562-563
Item member (Peer Channel example), 581

J - K - L

Java clients, DerivativesCalculator example, 65-67
Java Virtual Machine Specification, class file format, 23
JSON (JavaScript Object Notation), 615-619

Kerberos authentication method, 313

Laws of Identity, 289-291
layered framework designs, guidelines for, 451
Liberty identity protocols, 296

Lightweight Transaction Manager, 15-16
Listen activity, workflow communication, 166
List<T> class generics, 10
logins, Windows Live ID support for information cards, 332
logotypes, adding Information Cards to WCF applications, 343-344
logs (message), 637-640

M

Main() method (MSMQ PGM), 547-551
maintenance delegation, Windows Communication Foundation integration, 678-679
Managed Cards, 301
  creating, 364-367
  ini files, 366
management
  activity tracing, 640-642
  configuration system, 625-627
  message logging, 637-640
  Microsoft Management Model designer, 658
  performance counters, 647-648, 653-658
  security auditing facility, 633, 636
  Service Configuration Editor, configuring client applications, 631-632
  trade-recording services, 627-630
  tools overview, 624-625
  trace listener, 644-647
  Trace Viewer, 643-644
  WMI Provider, 649
    accessing data via Windows PowerShell, 652
    accessing data via WMI CIM Studio, 650
    adding custom performance counters, 653-658
Management Model designer (Microsoft), 658
ManualWorkflowSchedulerService, 193-194
mapping, COM+ web services, 417
maxMessageSize property, streamed transfer mode, 556
Message classes, 86
message formatter custom behavior, 471-473
message inspector custom behavior, 474-476, 684
Message objects, 86
Message Transmission Optimization Mechanism, 48
MessageEncoding property, 706
MessageFormatter property, 457
MessageInspectors property, 454-455
messages
C++, defining with, 22-23
contracts, 689
defined, 22-25
designing, Peer Channel example, 581
duplex messaging, exposing workflows as services, 237-243
encoders, 514
   binding elements, 516-519
custom transports, 530-531
exchange patterns, 489
logging, 73, 637-640, 696-698
routin, Windows Workflow Foundation workflows inside WCF services, 221-223
security, 252, 263
   client configuration, 264-266
   service endpoint configuration, 266-269
MetadataResolver class, 63, 351-353
Microsoft Management Model designer, 658
Microsoft Message Queuing, 130
Microsoft Password, 289
Microsoft Windows Vista DEBUG Build Environment prompt, 56
model-driven software development, 26, 28
Mono, 24
MSMQ (Microsoft Message Queuing), 130
   base address schemes, 51
   integrating with, 434
   PGM, publish/subscribe systems, 544-552
   queued delivery, 131
      dead-letter queues, 132
      poison queues, 133-134
Windows Communication Foundation service integration with
   creating clients, 438-442
   creating requests, 434-435
   creating services, 435-438
   testing, 442
MsmqIntegrationBinding, 49, 549, 552
MsmqMessage<PurchaseOrder>, 548
MsmqMessage<PurchaseOrder>, MsmqMessage<T>, 547
MsmqMessage<T> types, 552
MSMQPragmaticMulticastingSolution, 551
MTOM (Message Transmission Optimization Mechanism), 48, 706
MyChannelListener.cs, 502
MyCustomBindingElement.cs, 495-496, 501-502, 507
MyCustomReplyChannel.cs, 504
MyCustomRequestChannel.cs, 498
N
Name parameter, 39
Named Pipes, base address schemes, 51
Namespace parameter, 39
NAT-T, Teredo, 568
.NET Framework 3.5, 24, 29
   ATOM syndication in, 609-615
      Class Library, service-oriented programming, 26

How can we make this index more useful? Email us at indexes@samspublishing.com
HTTP support in, 324-326
new features of, 322-324
RSS syndication in, 609-615
Site Information page, 323
NetMsmqBinding, 49, 681-682
MSMQ PGM, 552
queued delivery, 131
session management, 129
NetNamedPipesBinding, 49, 682
Reliable Sessions, 126
session management, 129
transaction execution, 135
NetPeerTcpBinding, 49, 683
NetTcpBinding, 49, 517, 682-683
CompositeDuplexBindingElement, 539
Reliable Sessions, 126
session management, 129
transaction execution, 135
networks, P2P (peer-to-peer), 569
NetworkStream class, custom transports, 518
New Client Element Wizard, configuring client applications via, 632
New Project dialog (Visual Studio 2008), 75-76
New Service Element Wizard, configuring trade-recording services, 628-629
NextValueHandler() method, callback contracts, 542
non-backwards-compatible changes (versioning), 662
normalization (claims), 376
NotificationStreamWriter class, 562-565
Notify() operation
callback contracts, 538, 542
MSMQ PGM, 548, 551
streamed transfer mode, 565
nullable value types, 13-14

O
OleTx protocol, 134
one-way methods, transaction execution, 135
one-way operations (callback contracts), 538
Open() method, 551, 693, 702
OpenID, 332
operation contracts, asynchronous pattern for, 699-701
operation invokers custom behavior, service applications, 478-479
operation selector custom behavior, 467-468, 684
OperationBehaviors property, 457
OperationContext class, callback contracts, 541
OperationContract attribute, 39, 393, 547
OperationInvoker property, 455
operations
custom behaviors, attaching to, 457-459, 480-481
defining, 104-105
ServiceViewOfData class, 105
OperationSelector property, 454-455
Order project (MSMQ PGM), 545
OrderPullPoint configuration (MSMQ PGM), 548-549
OrderSubscriber class (MSMQ PGM), 550-551
out of the box activities (Windows Workflow Foundation), 151-152
OutputChannel property, 489

P
P2P (peer-to-peer) networks, 569
Parallel activity, 149
multiple client conversations, 248
workflows
exposing as services, 248
sequential workflows, 176-183
parameter inspector custom behavior, 469-471
ParameterInspectors property, 454-455
parameters, changing data contracts, 664-668
Partial Loading dialog (Trace Viewer), 644
partial types, 9-10
Password (Microsoft), 289
passwords, 287
   CardSpace, 312
   message security
      client configuration, 265-266
   service endpoint configuration, 267-269
Peer Channel
   binding, 570-573
   contracts, 574-575
   custom peer name resolution, 590-595
   data contracts, 579-581
   endpoints, 569, 574, 585-587
   example
      communication, 584-585
      designing messages/message exchange patterns, 581
      overview, 575-579
      testing, 595-597
   messages, directing to specific peers, 587-590
People Near Me, 598
   reasons for, 567-568
   service contracts
      defining, 582-584
      implementing, 584-585
Windows Peer-to-Peer Networking features, 568
Peer Name Resolution Protocol (PNRP), 568
peer-to-peer applications, structured data in, 567
People Near Me, 598
performance counters, 647-648, 653-658, 696
persistence services, workflow hosting, 187-189
Personal Cards (Information Cards), 300-303
PGM (pragmatic multicasting) protocol, 544-552
phishing attacks, 286
Phishing-Resistant Authentication Specification (OpenID), 332
PictureBox control, 555
PictureClient class, streamed transfer mode, 555
PictureServer class, streamed transfer mode, 553-554
PKI (Public Key Infrastructure), 289
PlaceOrder activities, 149
PNRP (Peer Name Resolution Protocol), 568
poison queues, 133-134
policies
   assertions, 670
   centralized lifecycle management, 670
   rules as (Windows Workflow Foundation)
      external policy execution, 205
      forward chaining, 204-205
portType (WSDL), 28
PowerShell
   scripts, services, 696
   WMI Provider data, accessing, 652
PPID (Private Personal Identifiers), 317-318, 325
pragmatic multicasting protocol, 544
predefined bindings, 681-682
Principle of Scenario-Driven Design, 685
Program.cs file, DerivativesCalculator example, 57-58
promoting
   properties in activities (Windows Workflow Foundation), 157-158
proof keys, 313
properties
   activities
      adding to custom activities, 153-154
promoting in (Windows Workflow Foundation), 157-158
DependencyProperty, 155-157
XAML properties, 154
Provider Cards (Information Cards), 301
public API layers (frameworks), designing, 685
public key infrastructure, 289
publish/subscribe systems
brokered notifications, 537
callback contracts, 538-544
PGM, 544-552
pull-style notifications, 537, 544
push-style notifications, 537
streamed transfer mode, 552-556
implementing via custom streams, 561-565
transmitting custom streams, 557-561
Publisher class (MSMQ PGM), 547-548
Publisher project (MSMQ PGM), 545
PublisherService class, 539, 563-565
PublisherService project, 539
PublisherService service type, callback contracts, 542
PublishingAgent class, streamed transfer mode, 563-565
pull-style notifications, 537, 544
PurchaseOrder class (MSMQ PGM), 545
pure virtual class, abstract base classes, 23
push-style notifications, 537
receiving messages
confirmations, 132
multiple instances, 131
via queue, 132
usage example, 137-143
queueing service requests, 681
QuizItem struct (Peer Channel example), 581
QuizResponse struct (Peer Channel example), 581
RandomDataPoint class, 539
Read() method, custom streams, 559-565
Really Simple Syndication, 604
Receive activity, 233
multiple client conversations, 245-247
workflows, exposing as services, 226, 245-247
declaratively creating contacts, 227-229
selecting existing contracts, 227
reference types, 13
Reliable Sessions, 125
custom bindings, 126-127
supported bindings, 126
toggling on/off, 126
usage example, 127-128
Relying Parties (Identity Metasystem), 294, 297, 330
Replicator activity, 248
ReplyAction parameter (OperationContract attributes), 393
ReplyChannel property, 489-491, 502-505
Repentational State Transfer, 599
RequestChannel property, 489-491, 497-500
resolver services, 683
Resource Access client application
resource access, authorizing, 378-383
STS addition process
queued delivery, 130
dead-letter queues, 132
NetMsmqBinding, 131
poison queues, 133-134
private messages, 131
certificate installation, 391-392
Fabrikam STS authorization policies, 395-396
Fabrikam STS configurations, 393-395
ISecurityTokenService service contracts, 393
prebuilt STS, 392
Resource Access Service reconfiguration, 405-408
Woodgrove STS, 397-405
Responder member (Peer Channel example), 581
ResponseText member (Peer Channel example), 581
REST POX services, 599
endpoints
addresses, 602
bindings, 602-603
contracts, 603
implementation of, 604
RSS servers, building, 604-608
SOAP services versus, 601
XML, 600
retiring endpoints, versioning, 669-670
RetrievePicture() method, streamed transfer mode, 554
RetrievingExceptionDetailInFaults, 708
revoke Information Cards, 311
rights (claims), 372
role providers, 16-18, 273-274
roles (security)
definition of, 373
role-based authorization
claim-based authorization versus, 373-374
resource access, authoring, 377-383
Root Agency, transport security, 253-255
routing messages, Windows Workflow Foundation workflows inside WCF services, 221-223
RSS (Really Simple Syndication)
NET Framework 3.5, syndicating in, 609-615
servers, building, 604-608
rules engine (Windows Workflow Foundation), 199
CAG activity, 202-203
rules as
conditions, 200-202
policies, 204-205
runtime services, workflow hosting, 186
custom services, 196-198
persistence services, 187-189
scheduler services, 192-195
tracking services, 189-192
S
SAML (Security Assertion Markup Language), claim-based authorization, 375
scheduler services, workflow hosting, 192
DefaultLoaderService, 194
DefaultWorkflowSchedulerService, 193
ManualWorkflowSchedulerService, 193-194
SharedConnectionWorkflowCommitWorkBatchService, 195
WorkflowQueuingService, 195
scoping service hosts/clients, 703
security
authorization, 272, 275, 281
configuring services to use role providers, 273-274
defining, 252
service configuration, 276-280
ServiceAuthorizationManager implementation, 279-280
basic tasks of, 251-252

How can we make this index more useful? Email us at indexes@samspublishing.com
CardSpace (Windows), 321-322
authentication methods, 313-319
Choose a Card dialog, 311
passwords, 312
security tokens, 313
Channel Layer, 252
claim sets, 372
claims
claim-based authorization, 373-376, 411
description of, 372
federated security versus, 412
normalization, 376
rights of, 372
role-based authorization, authorization
resource access, 384-391
SAML, 375
types of, 372
values of, 372
e-mail spoofing, 286
events, auditing, 633, 636, 698-699
federations, 376, 412
FFIEC, 287
identifiers, 374
identity
Identity Metasystem, 291-298, 319
Laws of Identity, 289-291
regulatory compliance, 288
role of, 285-287
user-centric identity management, 290
Identity Metasystem, Information Cards, 298-306
impersonation, 269-272
information card logins, Windows Live ID
support for, 332
messages, 252, 263
client configuration, 264-266
service endpoint configuration, 266-269
overview, 251-252
Password (Microsoft), 289
passwords, 287
phishing attacks, 286
PKI, 289
roles
definition of, 373
role-based authorization, 377-383
Service Model, 252
SID, 374
social-engineering schemes, 286
SSO, 288, 319
STS, 375
certificate installation, 391-392
client reconfiguration, 409-411
Fabrikam STS authorization policies, 395-396
Fabrikam STS configurations, 393-395
IIdentityTokenService service contracts, 393
prebuilt STS, 392
Resource Access Service reconfiguration, 405-408
Woodgrove STS, 397-405
transport, 252
client configuration for SSL over HTTP, 258
client configuration for SSL over TCP, 259-260
configuring server identities, 256
determining certificate trustworthiness, 253-255
identifying server provided certificates, 255-256
installing certificates, 253-255
removing SSL configuration from HTTPSYS, 283
removing SSL configuration from IIS, 282
restoring server identities, 283
service configuration for SSL over HTTP, 260
service configuration for SSL over TCP, 261-262
specifying certificates for HTTPSYS usage in SSL exchanges, 256
specifying certificates for IIS usage in SSL exchanges, 255-256
SSL protocol, 253
uninstalling certificates, 281-282
usage example, 257-262
websites (counterfeit), 286
Security Assertion Markup Language, 375
security auditing facility, 633, 636
security token service, 375
security tokens
CardSpace, 313, 350-351
federation, 321-322
Identity Metasystem, 293
Information Cards, adding to WCF applications, 343
service, 375
Self-Issued Cards (Information Cards), 300-301, 305
Send activity
BeforeSend event handler, 248
endpoint creation via, 217-219
multiple client conversations, 248
services
address updates via, 219
consuming, 214-219
exposing workflows as, 240, 243, 248
SendResponse message, defining service contracts, 582
sequential workflows, Windows Workflow Foundation
IfElse activity, 175
Parallel activity, 176-183
Serializable attribute (PurchaseOrder class),

545
serializers (XML)
System.Runtime.SerializationDataContractSerializer class, 87, 89-90
building clients, 95
building services, 92-94
usage example, 96-110
System.Xml.Serialization.XmlSerializer class, 87-90
server identities, restoring, 283
Service Configuration Editor
client applications, configuring, 631-632
trade-recording services, configuring, 627-630
Service directive, IIS hosting, 68
Service Host (WCF), 77-78
Service Library template (WCF), 77
Service Metadata Tool, 56
Service Model, 27
addresses, 28-30, 44-51
binding, 28-30, 44-51
Channel layer, 85
clients, building, 95
contracts, 28
defining, 36-42
implementing, 29
interfaces, designating as, 29
DerivativesCalculator example
clients, coding alternatives, 60, 63
clients, using with, 55-60
deploying services, 34-36, 51-55
Java clients, 65-67
identifying/documenting items, 684
overview, 28-29, 33
security, 252
services
building, 92-94
defining, 85

How can we make this index more useful? Email us at indexes@samspublishing.com
endpoints, 85, 103
hosting, 30, 42-44, 67-69, 72

Service reference settings dialog (Visual Studio 2008), 81-82

service section (WSDL), 28

Service Trace Viewer, 643

ServiceBehavior attribute, 29

ServiceContract attribute, 547
  callback contracts, 538
  contract names, altering, 39
  service types and, 40

ServiceHost class, 30, 551

services, 42
  addresses, dynamic updates via Send activity, 219
  authorization, configuring, 276-280
  building, 92-94
  clients, writing, 86
  configuring, 140
  consuming
    calling services via custom activities, 210-211, 214
    GetOptionChain method, 216
    InvokeWebService activity, 214
    Send activity, 214-219
  contracts
    defining, 582-584
    designing, 686-687
    duplex contracts, 690-691
    implementing, 584-585
    inheritance, 662
    custom activities, calling via, 210-211, 214
  data contracts, defining, 106
  defining, 85
  durable services, 114
    DataContract attribute, 117
   DataMember attribute, 117
    DurableService attribute, 117
    implementing, 115-121
IIS hosting, 692
message logging, 696-698
performance counters, 696
PowerShell scripts, 696
session state information, 693
synchronizing shared data access, 691-692
WMI provider, 696
Windows Workflow Foundation workflows, hosting as, 221-226
workflow services, creating, 233
workflows, exposing as, 220, 226
content tokens, 234-237
duplex messaging, 237-243
multiple client conversations, 245-248
Receive activity, 226-229
WorkflowServiceHost class, 230-232
XML
   describing documents in, 28
   REST services, 600
ServiceViewOfData class
   data contracts, defining, 106
  DataContractSerializer class usage example, 105
session management
   initiating/ending sessions, 130
   instance context sessions, 129
   supported bindings, 129
   usage example, 137-143
Session parameters (ServiceContract attribute), callback contracts, 538
SetState activity (Windows Workflow Foundation), state machine workflows, 183
shared data, synchronizing access, 691-692
SharedConnectionWorkflowCommitWorkBatchService, 195
SID (security identifiers), 374
Singleton services, Windows Workflow Foundation workflows inside WCF services, 224-226
Site Information page
   CardSpace, 343
   .NET Framework 3.5, 323
SOAP (Simple Object Access Protocol)
   binding elements and, 48
   messages, streamed transfer mode, 556-557
   MTOM, 706
   REST POX services versus, 601
toolkit, service-oriented programming, 26
social-engineering schemes, 286
software factory templates, 26-28
SomeError class, exception handling, 112
spoofed email, 286
SQL (Structured Query Language), InstanceData database, 119
SqlTrackingService, workflow hosting, 191-192
SSL (Secure Sockets Layer) protocol
   HTTP.SYS, removing SSL configuration from, 283
   transport security, 253
SSO (single sign on), 288, 319
standard bindings, 48-50
state machine workflows, Windows Workflow Foundation, 183-184
state machines, 492-493
stateful services, 115
Stream object, 553
streaming transfer mode, 706-707
   custom streams
      implementing via, 561-565
      transmitting, 557-561
   publish/subscribe systems, 552-556
      implementing via custom streams, 561-565
      transmitting custom streams, 557-561
BindingElementExtensionElement, 509-511
System.ServiceModel.Description.
IEndpointBehavior, 457
System.ServiceModel.Description.
IOperationBehavior, 457
System.ServiceModel.Dispatcher, 452-453
ChannelDispatcher, 454
ClientOperation, 453
System.ServiceModel.Dispatcher.ClientRuntime,
453-454
System.ServiceModel.Dispatcher.Dispatch
Operation, 454-455
System.ServiceModel.Dispatcher.Dispatch
Runtime, 454
System.ServiceModel.Dispatcher.IClient
MessageFormatter interface, 453
System.ServiceModel.Dispatcher.IClient
MessageInspector, 454
System.ServiceModel.Dispatcher.IDispatch
MessageFormatter, 455
System.ServiceModel.Dispatcher.IDispatch
MessageInspector, 455
System.ServiceModel.Dispatcher.IDispatch
OperationSelector, 455
System.ServiceModel.Dispatcher.IInstance
ContextProvider, 454
System.ServiceModel.Dispatcher.IInstance
Provider, 453-455
System.ServiceModel.Dispatcher.IOperation
Invoker, 455
System.ServiceModel.Dispatcher.IParameter
Inspector, 454-455
System.ServiceModel.Dispatcher.IReplyChannel
interface, 504-505
System.ServiceModel.OperationContext object,
135
System.ServiceModel.ServiceAuthorization
Manager, 695
System.ServiceModel.ServiceDebugBehavior,
ReturnExceptionDetailInFaults, 708
System.ServiceModel.ServiceHost
Close() method, 692-693
Open() method, 693
System.ServiceModelDispatcher.IClientMessage
Formatter, 457
System.TimeoutException, 701
System.Transactions namespace, 15-16, 136
System.Web.Security.RoleProvider class, 16-17
System.Xml.Serialization.XmlSerializer class,
87-90

TCP (Transfer Control Protocol)
base address schemes, 51
binding elements and, 48
TcpClient class, custom transports, 517-519
TcpListener class, custom transports, 517-519
templates, WCF project templates, 75-77
Teredo, 568
Test Client (WCF), 79-80
testing Peer Channel, 595-597
TokenDetails section (ini files), Managed Cards,
366
TokenProcessor, CardSpace over HTTP, 370
TokenType section (ini files), Managed Cards,
366
ToString() method (MSMQ PGM), 545
trace listeners, 637, 644-647
Trace Viewer, 643-644
tracing (activity), 640-642
tracking services, workflow hosting
SqlTrackingService, 191-192
tracking profiles, 189-191
tracking stores, querying, 192
trade-recording services, configuring via Service
Configuration Editor, 627-630

How can we make this index more useful? Email us at indexes@samspublishing.com
transactions
ADO.NET, initiating via, 15
committing, 135
distributed transactions, 15, 136
Enterprise Services, initiating via, 15
Lightweight Transaction Manager, 15-16
OleTx protocol, 134
one-way methods, 135
service operators, including in, 136
supported bindings, 135
usage example, 137-143
WS-AT protocol, 134
transport channels, 528-530
transport security, 252
certificates
determining trustworthiness, 253-255
identifying server provided certificates, 255-256
installing, 253-255
removing SSL configuration from HTTP.SYS, 283
removing SSL configuration from IIS, 282
specifying certificates for HTTP.SYS usage in SSL exchanges, 256
specifying certificates for IIS usage in SSL exchanges, 255-256
uninstalling, 281-282
client configuration for SSL over HTTP, 258
TCP, 259-260
server identities
configuring, 256
restoring, 283
service configuration for SSL over HTTP, 260
TCP, 261-262
SSL protocol, 253
usage example, 257-262
transports
customizing, 513
binding elements, 532
channel listeners, 525-527
implementing binding elements, 516-532
message encoders, 530-531
TCP protocol support, 520-532
inbound communication
binding elements, 515-516
framing messages, 514
stack construction, 515-516
message encoders, 514-519
outbound communication, 514-516
Triangle of Trust (Identity Metasystem), 319
Trusted Root Certification Authority Store, installing transport security certificates, 253-255
types (claims), 372
U - V
UML (Unified Modeling Language), 27
updates
COM+ services, 418
service addresses via Send activity, 219
user-centric identity management, 290
usernames, message security
client configuration, 265-266
service endpoint configuration, 267-269
validation, activities (Windows Workflow Foundation), 168-170
value types (nullable), 13-14
values (claims), 372
VBScript files, building Windows Communication Foundation services via COM+, 433

versioning
  backwards-compatible changes, 662
  bindings, changing, 669
  centralized lifecycle management, 670-672
decision trees, 662
endpoints, retiring, 669-670
non-backwards-compatible changes, 662
operations
  adding, 662, 664
  changing, 668
  changing parameter data contracts, 664-667
deleting, 668
services
  changing endpoint addresses, 670
  contract inheritance, 662

Vista (Windows)
dead-letter queues, 132
poison queues, 133-134

Visual Studio .NET, Windows Forms Designer, 27

Visual Studio 2008
  New Project dialog, 75-76
  Service reference settings dialog, 81-82
WCF
  project templates, 75-76
  Service Host, 77-78
  Service Library template, 77
  Test Client, 79-80

W

web browsers
  architectures of, 600
  Information Cards, adding to, 353-363

Web Hosted In-Process mode, 419
Web Hosted mode, 419

web services
  COM+
    automatic mapping, 417
    COM+ Hosted mode, 419
  exposing component interfaces as, 417-418, 422-426
  referencing Windows Forms clients, 426-427
  supported interfaces, 418
  Web Hosted In-Process mode, 419
  Web Hosted mode, 419
  composableity, defining, 445-446
  defining, 445
  interoperability, 445-447

Web Services Eventing, publish/subscribe systems and, 537

Web Services Notification, publish/subscribe systems and, 537

Web Services Trust Language, 375

websites
  counterfeit websites, 286
  Relying Party websites, building, 330

Windows clients (Identity Metasystem), building, 330-331

Windows Communication Foundation
  addresses, 679-680
  bindings
    activating/deactivating binding features, 683
    custom message inspectors, 684
    custom operation selectors, 684
    custom peer resolver services, 684
    HTTP bindings, 682
    identifying services with contended resources, 681
    predefined bindings, 681-682
    queueing service requests, 681

How can we make this index more useful? Email us at indexes@samspublishing.com
clients
AJAX, 703-705
anticipating exceptions, 701-702
asynchronous pattern for operation contracts, 699-701
lifetime management, 703
Open() method, 702
scoping service hosts/clients, 703

contracts
behavioral contracts, 689
defining data contracts, 688
designing public API layers, 685
designing service contracts, 686-687
designing via scenarios, 684
duplex service contracts, 690-691
fault contracts, 689
identifying/documenting service model items, 684
message contracts, 689
Principle of Scenario-Driven Design, 685
service contracts, 690-691
structural contracts, 687-689
debugging applications, 707-708

integrating
delegating maintenance, 678-679
exemplars, 677
incorporating existing applications, 678
rebuilding existing applications, 677
large amounts of data, 705-706
MTOM, 706
project templates, 75-77
Service Host, 77-78

services
activity tracing, 696-698
ASP.NET Compatibility Mode, 695
auditing security events, 698-699
authorization, 695
class libraries, 693
COM+, calling from, 428-433
comprehensive health models, 696-697
ensuring manageability, 695
Extension property, 693-695
IIS hosting, 692
message logging, 696-698
MSMQ integration with, 434-442
performance counters, 696
PowerShell scripts, 696
session state information, 693
synchronizing shared data access, 691-692
System.ServiceModel.Service AuthorizationManager, 695
WMI provider, 696
streaming transfer mode, 706-707
Test Client, 79-80

Windows Forms
client references to COM+ web services, 426-427
Windows Forms Designer, 27

Windows Live ID, information card login support, 332

Windows Peer-to-Peer Networking features, 568

Windows PowerShell, accessing WMI Provider data, 652


Windows Workflow Foundation
activities
ACID transactions, 170
binding, 156-157
CAG (ConditionedActivityGroup), 202-203
compensation, 171-172
custom activities, 152-160
custom root, 184
Delay, 183
design behavior, 167-170
error handling, 149
EventDrivenActivity, 183
HandleExternalEvent, 183
IfElse, 175
initializing, 149
lifecycle of, 149
out of the box activities, 151-152
Parallel, 176-183
Parallel activities, 149
PlaceOrder activities, 149
promoting properties, 157-158
SetState, 183
validation, 168-170
workflow communication host configuration, 163-166
workflow communication interface, 160-163
Bind dialog, 157
components of, 147
defining, 148
rules engine, 199
CAG activity, 202-203
rules as conditions, 200-202
rules as policies, 204-205
workflows, 173
exposing as services, 226-248
hosting, 185-198
hosting inside WCF services, 221-226
sequential workflows, 175-183
services, 233
state machine workflows, 183-184
workflows
activities
communicating via, 160-163
configuring host communication, 163-166
hosting, 185
inside WCF services, 221-226
runtime services, 186-198
services, creating, 233
services, exposing as, 220, 226
content tokens, 234-237
duplex messaging, 237-243
multiple client conversations, 245-248
Receive activity, 226-229
WorkflowServiceHost class, 230-232
Windows Workflow Foundation, 173
custom root activities, 184
sequential workflows, 175-183
state machine workflows, 183-184
WorkflowServiceHost class, exposing workflows as services, 230-232, 241-243
WS-AT (WS-AtomicTransaction) protocol, 134
WS-FederationBinding, 681
WS-ReliableMessaging protocol, 48
WS-Security, 48, 296
WS-Trust (Web Services Trust Language), 296, 375, 393
WSDL (Web Services Description Language), 24
downloading, 32

How can we make this index more useful? Email us at indexes@samspublishing.com
export extension, 479
attaching to operations/endpoints, 480-481
declaring, 480
informing Windows Communication Foundation of, 481-482
key terms, 28
WSDL2Java, 65, 67
WSDualHttpBinding, 49, 683
    CompositeDuplexBindingElement, 539
    Reliable Sessions, 126
    session management, 129
    transaction execution, 135
WSFederationBinding, 49, 446
    Information Cards, adding to WCF applications, 347-348
    Reliable Sessions, 126
    transaction execution, 135
WSHttpBinding, 49, 446, 681
    Reliable Sessions, 126
    session management, 129
    transaction execution, 135

X - Y - Z

X.509 certificates, CardSpace (Windows), 333
    host file updates, 334-335
    certificate stores, importing into, 334
    private key access, 336
XAML properties in custom activities, 154
XML (Extensible Markup Language), 615
    class representation in, 95
    clients, building, 95
    contract-first development, 91
    data contracts, 87-89
    REST services, 600
    serializers

System.Runtime.SerializationDataContractSerializer class, 87-110
System.XML.Serialization.XmlSerializer class, 87-90
services, building, 92-94
WSDL and, 24
XML Information Set (InfoSet), data representation, 86
XmlSerializer class, 87-90
XSI
    Authorization Manager, authorizing resource access, 384-391
    claim-based authorization, 411
    ACL versus, 374-375
    adoption strategies, 375-376
    role-based authorization versus, 373-374
STS addition process
    certificate installation, 391-392
    client reconfiguration, 409-411
    Fabrikam STS authorization policies, 395-396
    Fabrikam STS configurations, 393, 395
    ISecurityTokenService service contracts, 393
    prebuilt STS, 392
    Resource Access Service reconfiguration, 405-408
    Woodgrove STS, 397-405