

Introducing the .NET Framework 3.5

 David Chappell, Chappell & Associates

September 2007

© Copyright Microsoft Corporation 2007. All rights reserved.

Contents

[Describing the .NET Framework 3.5 3](#_Toc178747779)

[The Challenges of Building Modern Applications 3](#_Toc178747780)

[Addressing the Challenges: Spotlighting the .NET Framework 3.5 4](#_Toc178747781)

[ASP.NET AJAX: Support for Responsive Browser Applications 4](#_Toc178747782)

[Language-Integrated Query: Consistent Access to Diverse Data 5](#_Toc178747783)

[Windows Communication Foundation: Support for Service-Oriented Applications 5](#_Toc178747784)

[Windows Workflow Foundation: Support for Workflow-Based Applications 6](#_Toc178747785)

[Windows Presentation Foundation: A Unified Approach to Diverse User Interfaces 7](#_Toc178747786)

[Windows CardSpace: Consistent User Control of Digital Identities 9](#_Toc178747787)

[Summarizing the Evolution: The .NET Framework 3.5 and its Predecessors 10](#_Toc178747788)

[Applying the .NET Framework 3.5: A Scenario 12](#_Toc178747789)

[Understanding the .NET Framework 3.5: The Technologies 13](#_Toc178747790)

[ASP.NET AJAX 13](#_Toc178747791)

[Language-Integrated Query 14](#_Toc178747792)

[Windows Communication Foundation 15](#_Toc178747793)

[Services and Clients 15](#_Toc178747794)

[Support for REST and RSS/ATOM 17](#_Toc178747795)

[Communication Options 18](#_Toc178747796)

[Security, Reliability, and Transactions 18](#_Toc178747797)

[Tool Support 19](#_Toc178747798)

[Windows Communication Foundation and Other Microsoft Technologies 19](#_Toc178747799)

[Windows Workflow Foundation 20](#_Toc178747800)

[Workflows 21](#_Toc178747801)

[The Base Activity Library 21](#_Toc178747802)

[Tools for Windows Workflow Foundation: The Workflow Designer 22](#_Toc178747803)

[The Runtime Engine and Runtime Services 23](#_Toc178747804)

[Workflow-Enabled Services 24](#_Toc178747805)

[Windows Workflow Foundation and Other Microsoft Technologies 24](#_Toc178747806)

[Windows Presentation Foundation 25](#_Toc178747807)

[The Capabilities of Windows Presentation Foundation 25](#_Toc178747808)

[Applying Windows Presentation Foundation 26](#_Toc178747809)

[Tools for Windows Presentation Foundation 27](#_Toc178747810)

[Windows Presentation Foundation and Other Microsoft Technologies 27](#_Toc178747811)

[Windows CardSpace 28](#_Toc178747812)

[Windows CardSpace and the Identity Metasystem 28](#_Toc178747813)

[Fighting Phishing 31](#_Toc178747814)

[Windows CardSpace and Other Microsoft Technologies 31](#_Toc178747815)

[Conclusion 32](#_Toc178747816)

[About the Author 32](#_Toc178747817)

# Describing the .NET Framework 3.5

The goal in application development is always the same: create the best possible software in the least amount of time. Yet the bar is continually raised, as demands from customers increase. To meet these demands, the platforms developers build on and the tools they use must get better and better—they must evolve.

The .NET Framework provides a clear example of this. First released in 2002, version 2.0 of the Framework appeared three years later. The .NET Framework 3.0, released in 2006, was a major update that added a number of new technologies, while the latest version, the .NET Framework 3.5, includes more useful additions. Alongside this evolution, Microsoft’s flagship tool for creating .NET applications has also moved forward. The latest release, Visual Studio 2008, offers a range of improvements for creating .NET applications. Every step in this path has been aimed at providing a better and more productive environment for the people who create Windows software.

This overview provides a big-picture view of the .NET Framework 3.5. While a basic knowledge of the .NET Framework is assumed, this description focuses on the technologies added in the .NET Framework 3.0 and 3.5. The goal is to make clear what this widely used foundation for Windows applications provides today.

## The Challenges of Building Modern Applications

Creating a modern application is not a simple task—the requirements are substantial. Traditional concerns such as creating effective business logic and allowing access via a Web browser are still important, but they’re no longer enough. Modern applications present a range of new challenges, including the following:

1. Users increasingly expect Web browser interfaces to act like installed Windows applications. Loading a new page whenever something has changed is no longer sufficient—it’s just too slow. What’s needed is better *support for responsive browser applications*.
2. Data remains central to most applications. How that data can be represented, however, has expanded considerably. Relational data is still important, as is mapping between objects and relations. Yet the amount of data represented using XML continues to increase, a trend that’s not likely to change. And even though it’s not always viewed in this way, a running program’s objects also contain data. A technology that allowed *consistent access to diverse data* would help developers create applications in less time and with fewer errors.
3. Applications commonly communicate with other applications, both inside and outside the organization. Modern applications also must often fit into a service-oriented architecture (SOA), exposing some of their functionality as interoperable services accessible by other software. Achieving these goals requires *support for service-oriented applications*.
4. Organizations are increasingly taking a process-oriented view of what they do. Since most applications automate some part of a business process, it can be useful to make the steps in this process explicit in the code. An effective way to do this is by using workflow technology, an approach that requires *support for workflow-based applications*.
5. The requirements for a modern user interface have grown significantly. Providing real business value can commonly require working with various kinds of documents, using two- and three-dimensional graphics, displaying video, and more. Meeting these needs requires a *unified approach to diverse user interfaces*.
6. The people who use an application commonly need a way to convey information about who they are. Many different technologies for defining and using a digital identity are in use, and problems such as phishing are common. Given this, a modern application and the people who use it can benefit from *consistent user control of digital identities*.

Given that today’s applications commonly need to tackle some or all of these challenges, the platform those applications are built on should also address all of them. The goal of the .NET Framework 3.5 is to do this for Windows software.

## Addressing the Challenges: Spotlighting the .NET Framework 3.5

The .NET Framework 3.5 contains a number of technologies that can help developers solve the problems just described. Some were part of the .NET Framework 3.0, while others are new with the 3.5 release. This section introduces the most important of these technologies.

### ASP.NET AJAX: Support for Responsive Browser Applications

Web browsers are the most popular user interface for new applications. Still, they’ve traditionally suffered from a significant limitation: Each new request from a user requires a round-trip to the Web server, commonly resulting in loading a new page. A more intelligent—and faster—approach would be to access data in the background whenever possible, then update only those parts of the page that change. Users would see much more responsive applications, since they would spend less time waiting for new pages to be loaded.

This is exactly what’s done by the AJAX approach to building Web applications. Rather than load a new page for each user request, the browser asynchronously requests data in advance. The code that makes this request is typically written in JavaScript, and the data is often (although not always) formatted in XML. These three characteristics—Asynchronous JavaScript and XML—are the source of name *AJAX*.

Even though the core technologies that underlie AJAX first appeared in the 1999 release of Internet Explorer 5, this approach took several years to become popular. Today, however, AJAX is becoming the dominant style for new Web browser applications. Accordingly, the .NET Framework 3.5 now incorporates a technology called ASP.NET AJAX. An extension to the original ASP.NET, the goal is to make it easier for developers to create AJAX applications.

### Language-Integrated Query: Consistent Access to Diverse Data

Most applications work with data in some way. That data can be represented in a variety of ways, including as tables in a relational database, in XML documents, and in objects held in memory. Accessing each of these kinds of data has traditionally required using a different approach. It’s common to use SQL to work with relational data, for example, XQuery for XML data, and custom code for in-memory data. Yet since the intent is the same—accessing information—why not provide a common approach for all of these cases?

This is the goal of Language-Integrated Query (LINQ), a technology that’s new with the .NET Framework 3.5. Rather than requiring specialized languages and a separate approach for each kind of data, LINQ adds a set of extensions to C# and VB that allow common access to diverse information. This technology covers a large area, including object/relational mapping and more. The goal is to make life simpler and more efficient for anybody who creates or maintains .NET Framework applications that work with data.

### Windows Communication Foundation: Support for Service-Oriented Applications

From the beginning, the .NET Framework has provided plenty of approaches to communication. Those choices include the following:

1. ASP.NET Web Services (commonly known as ASMX), providing interoperable SOAP-based communication.
2. .NET Remoting, focusing on communication between .NET applications.
3. Enterprise Services, offering support for scalable, transactional applications.
4. System.Messaging, supporting queued messaging via Microsoft Message Queuing (MSMQ).
5. Web Services Enhancements (WSE), an extension to ASP.NET Web Services that provides support for more recent specifications such as WS-Security.

All of these technologies have had a role to play. Yet why have several different solutions to address what is essentially the same problem? Why not instead create a single foundation for application communication?

This is exactly what’s done by Windows Communication Foundation (WCF). Rather than requiring developers to use a different technology with a different application programming interface for each kind of communication, WCF provides a common approach using a common API. Originally released as part of the .NET Framework 3.0, WCF is now Microsoft’s recommended approach for communication. Most applications that might have used one of the technologies just listed should instead use WCF for communication.

As the figure below shows, the basic model of WCF is simple: A client accesses some service, invoking operations as required. WCF doesn’t mandate any particular host, and so developers are free to use this communication technology inside any host process.

WCF provides strong support for interoperable communication via SOAP, an essential part of modern computing. This includes support for several of the WS-\* specifications, including WS-Security, WS-ReliableMessaging, and WS-AtomicTransaction. WCF doesn’t require SOAP, however, and so other approaches can also be used, including an optimized binary protocol, queued messaging using MSMQ, and a simpler REST-based approach built directly on HTTP.

Communication between applications, whether within an organization or across organizations, is a fundamental part of modern software. The .NET Framework 3.5 addresses this challenge via the service-oriented approach of WCF.

### Windows Workflow Foundation: Support for Workflow-Based Applications

A workflow is a simple idea: it’s just a series of steps performed in some order. One might even argue that every application implements a workflow, since every application executes some process. Yet the traditional approach to creating an application using C# or Visual Basic or some other programming language is to make the steps in this process implicit in the code. This certainly works, but it also embeds the process itself deeply into a program’s logic, making that process more difficult to create and to change.

Using workflow technology to implement process logic can be an effective way to address this problem. Rather than intertwining the logic in ordinary code, each step in the process is explicitly defined, then executed by a *workflow engine*. The result is a clean implementation of the process itself. With Windows Workflow Foundation (WF), Microsoft provides a common workflow technology for Windows, giving any workflow-based application a common foundation to build on. Since its release in the .NET Framework 3.0, WF has been used both in software provided by Microsoft, such as Windows SharePoint Services, and in applications created by other organizations.

But how can a single technology meet the diverse set of requirements presented by different workflow applications? The answer adopted by WF is to take a very general view of workflow. As the figure below shows, a WF workflow is just a group of *activities* that are executed by the WF engine in some order. Each activity is actually a class, and it can contain any work that the workflow’s creator deems necessary. Activities can potentially be reused across different workflows, making it easier to create automated solutions to new problems.

By providing a common workflow technology for Windows, WF makes this useful paradigm for building software generally available to developers. As a process-oriented view of software continues to gain in popularity, the use of workflow will likely grow as well.

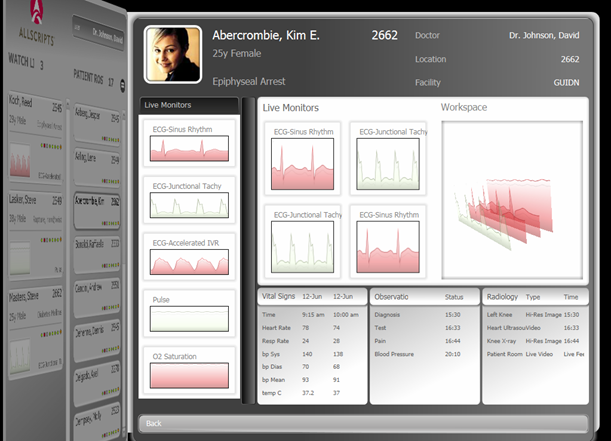
### Windows Presentation Foundation: A Unified Approach to Diverse User Interfaces

The user interface is an important part of nearly every application. Yet what users expect from those interfaces has advanced significantly. Traditional menu-driven GUIs are still required, of course, but applications may also need to display video, run animations, use two- and three-dimensional graphics, and work with various kinds of documents. And all of this must be possible whether the application is accessed from a standalone desktop client or via a Web browser.

Traditionally, all of these aspects of the user interface have been provided in different ways on Windows. For example, a developer can use Windows Forms, part of the .NET Framework, to build a Windows GUI. Creating a Web browser interface requires using HTML and perhaps Java applets or JavaScript code. Displaying video might rely on Windows Media Player or something else, while document formats might be defined by Microsoft Word or PDF or in another way. The challenge for developers is clear: building a coherent user interface for different kinds of clients using diverse technologies isn’t simple.

A primary goal of Windows Presentation Foundation (WPF), originally released with the .NET Framework 3.0, is to address this challenge. By offering a consistent technical underpinning for all of these user interface aspects, WPF makes life simpler for developers. By taking a more modern approach, including support for video, animation, two- and three-dimensional graphics, and various kinds of documents, WPF can let users work with information in new ways. And by providing a common foundation for desktop clients and browser clients, WPF makes it easier to build applications that address both.

The example interface shown below, containing images, live graphics, three-dimensional views, and more, illustrates some of what WPF provides. Rather than requiring developers with skills in diverse technologies, user interfaces like this one can now be created in a more consistent way.



One challenge that has long faced the creators of user interfaces stems from the different roles required for building effective interfaces. Software developers are needed to create the logic behind the interface, but they’re rarely the best people to define the interface’s look and feel. Designers, specialists in human/machine interaction, are typically a much better choice for this role. Yet older technologies such as Windows Forms are focused entirely on the developer. There’s no truly effective way for developers and designers to collaborate. To address this problem, WPF relies on the eXtensible Application Markup Language (XAML). An XML-based language, XAML allows specifying a user interface declaratively rather than in code. This makes it much easier for tools to generate and work with an interface specification based on the visual representation created by a designer. In fact, Microsoft provides Expression Blend to do exactly this. Designers can use this tool (and others provided by third parties) to create the look of an interface, then have a XAML definition of that interface generated for them. The developer reads this definition into Visual Studio, then creates the logic the interface requires.

When a developer creates a standalone WPF application, one that runs directly on Windows, she has access to everything WPF provides. To create a client that runs inside a Web browser, however, a developer can build a *XAML browser application*, commonly referred to as an *XBAP.* Built on the same foundation as a standalone WPF application, an XBAP allows presenting the same style of user interface within a downloadable browser application. The same code can potentially be used for both kinds of applications, which means that developers no longer need different skill sets for desktop and browser clients. As is typical for this kind of rich Internet application, an XBAP downloaded from the Internet runs in a secure sandbox, which limits what the application can do. Still, a large subset of the user interface functionality available to a standalone WPF application can also be used in an XBAP.

Both WPF standalone applications and XBAPs can take advantage of WPF’s modern graphics support, including the ability to use hardware acceleration, support for vector graphics, and more. By making it easier to create 3D graphics, WPF makes available a range of data visualization options that aren’t possible with Windows Forms or other earlier technologies. WPF also provides the foundation for the XML Paper Specification (XPS), which defines a standard format for viewing, distributing, and printing fixed-format documents.

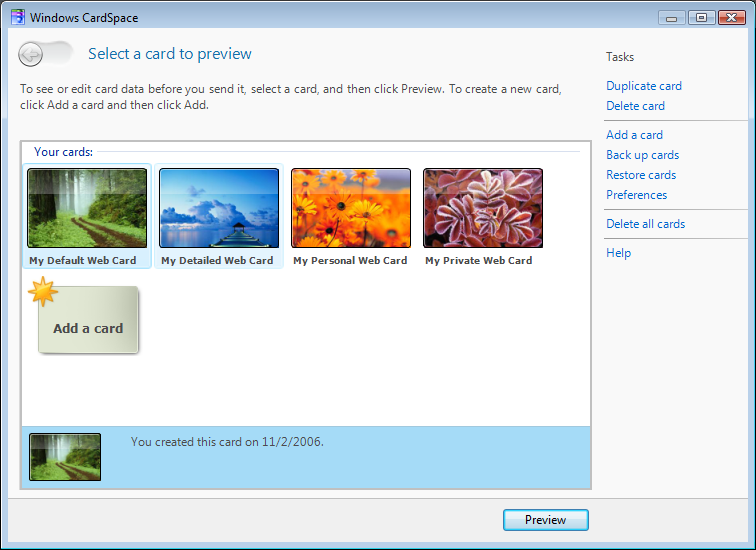
User interfaces are a complex and important part of modern applications. Through WPF, the .NET Framework 3.5 presents a more complete and consistent solution to the challenges these interfaces present. The goal is to let people who create user interfaces—both developers and designers—do their jobs more effectively.

### Windows CardSpace: Consistent User Control of Digital Identities

Think about how people represent themselves today on the Internet. In the majority of cases, a person’s digital identity is expressed as a simple username. Combined with a password, this identity is used to access email accounts, Internet merchants, and even on-line banks and other financial institutions. Yet despite their popularity, usernames and passwords have several drawbacks. Here are the two most important:

1. People have a hard time remembering all of the usernames and passwords they’ve chosen for different sites. Many people use the same values for different sites, easing the memory problem but increasing the security risk.
2. Usernames, passwords, and other personal information can be stolen by phishers. By sending deceptive emails, phishers entice their victim to log into a Web site that looks just like, say, the site of the victim’s bank. The site is actually controlled by the phisher, however, and so once the victim enters his username and password, the phisher can use this information to masquerade as the user at the real site.

Reducing the severity of these problems requires a new approach to managing digital identities. Windows CardSpace, originally released with the .NET Framework 3.0, is an important part of that approach. To help people keep track of their digital identities, CardSpace represents each identity as a distinct *information card*. If a Web site accepts CardSpace logins, a user attempting to log into that site will see a CardSpace selection screen like the one below. By choosing a card, the user also chooses a digital identity that will be used to access this site. Different cards can contain different information, allowing a user to control exactly what each site learns about her.



The identities represented by these cards are created by one or more *identity providers*. Any organization can offer an identity provider, and CardSpace itself also includes a self-issued identity provider that runs on client machines. With this provider, users can create their own identities that don’t rely on passwords for authentication. Web sites can accept these self-issued CardSpace identities rather than relying on the usual password-based approach, reducing the problems that passwords bring.

Windows CardSpace is actually part of a larger *identity metasystem*. Based entirely on open, public protocols, this metasystem defines a way to use different digital identity technologies consistently across diverse platforms (including operating systems other than Windows) and diverse applications (including Web browsers other than Internet Explorer). By providing a common way to select identities and more for Windows, CardSpace fills a key role in the metasystem. And by addressing the fundamental problem of identity, CardSpace also plays an important part in the .NET Framework 3.5.

## Summarizing the Evolution: The .NET Framework 3.5 and its Predecessors

The .NET Framework 3.5 is the latest step in the evolution of Microsoft’s flagship development platform, with each step building on what came before. This most recent release is a superset of the .NET Framework 3.0, and it brings no breaking changes. Similarly, the .NET Framework 3.0 was a superset of the 2.0 release, and it also contained no breaking changes. To help make the stages in this evolution clear, the figure below shows what’s been added in the 30 and 3.5 releases.

Everything in the .NET Framework depends, as it always has, on the Common Language Runtime (CLR). A large group of classes, known as the .NET Framework class library, is built on top of the CLR. This library has expanded with each release, as the figure shows. The .NET Framework 2.0 provided the fundamentals of a modern development environment, including the base class library, ASP.NET, ADO.NET, and much more. The .NET Framework 3.0 didn’t change any of this, but it did add four important new technologies: WCF, WF, WPF, and Windows CardSpace.

The changes in the .NET Framework 3.5 affect several parts of the 3.0 release. ASP.NET gets AJAX support, while LINQ is available for use with ADO.NET and in other ways. Various additions were made to the base class library, such as addition of a type supporting sets—unordered collections of unique elements—and improved encryption support. WCF, WF, and WPF each get enhancements as well, as described later in this overview. The set of operating systems on which the Framework runs was also updated: The .NET Framework 3.5 runs only on Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP.

Because each release adds to its predecessor, the need to retest applications built for earlier releases is minimized. Also, because all three of these versions can run simultaneously, it’s still possible to run applications on older versions of the Framework if desired. Visual Studio 2008 even allows creating projects that target a particular version of the Framework. An application built in this way will use only the binaries for that version, and the developer will see only those aspects of Visual Studio and the Framework itself that work in this older world. A developer who chooses to build a new application targeting the .NET Framework 2.0, for example, can make his world look just as if only this older version of the Framework were available.

# Applying the .NET Framework 3.5: A Scenario

One way to understand how a group of technologies work together is to look at an example of how they can be used. Imagine, for example, an application that lets customers and agents submit insurance applications. If it were implemented using the .NET Framework 3.5, this application might look something like the figure below.

The application’s business logic, shown in the upper left of the diagram, is implemented using a WF workflow. Handling an application for insurance is a multi-step process, including evaluating the application against this organization’s underwriting rules, perhaps checking the applicant’s credit, and maybe even getting a manager’s approval. The workflow implements each of these steps as an activity, relying on other software as needed. To access stored data, the activities in this workflow use a LINQ option for issuing SQL queries.

This insurance company provides a call center, allowing its customers to apply for insurance over the phone. The client software used by the people staffing that call center, shown in the upper right of the diagram, is implemented as a standalone WPF application. This client communicates with the application’s business logic via WCF, using a binary protocol optimized for WCF-WCF communication. As the figure shows, the call center workers rely on Windows CardSpace to select the identity they will use when logging into this application.

Customers can also apply for insurance via the Web. To allow this, the application uses ASP.NET AJAX to communicate with Web browsers, providing a responsive user interface for customers. As shown in the lower left corner of the diagram, a customer accessing this application via a Web browser can use CardSpace to select the identity he wishes to present.

Insurance agents who access this application over the Internet may well need a more functional interface than what customers see. Accordingly, they can rely on an XBAP rather than an AJAX interface. As shown in the lower center of the diagram, this gives those agents a large share of the user interface functionality provided by the WPF desktop application used in the call center. Both are built on the same foundation, and so the application’s developers can reuse the same code in the two types of clients. And as with the other kinds of clients, agents can use CardSpace to select the identity they wish to present to the application.

Finally, it’s likely that this application will need to access and be accessed by other applications. If approving a customer requires a credit check, for example, this will most likely be done through a call to an external service. Or perhaps the application accepts requests directly from other software, exposing services that these external applications can invoke. For cases like these, shown in the lower right of the figure, the application relies on WCF to communicate using standard Web services. Whatever technology these applications are built on, WCF’s support for SOAP makes interacting with them straightforward.

This scenario illustrates how some of the .NET Framework 3.5’s components might be used in a typical application. Quite a few options have been omitted, and so don’t view this simple example as a complete illustration of what this technology family provides. Instead, the goal is to give a clear idea of how the various parts of the .NET Framework can be used together to address real business problems.

# Understanding the .NET Framework 3.5: The Technologies

To get a better feel for what the .NET Framework 3.5 provides, it’s useful to dig a little deeper into its components. This section provides a short tutorial on each of the technologies spotlighted earlier.

## ASP.NET AJAX

ASP.NET is one of the today’s most popular technologies for creating Web-based applications. AJAX is becoming a dominant approach for designing those applications. These two realities imply that ASP.NET AJAX will be one of the most widely used parts of the .NET Framework 3.5. The figure below illustrates the two main components of this technology.

As the figure shows, an ASP.NET application can use the ASP.NET 2.0 AJAX Extensions. These provide a set of server-side controls for interacting with code in a Web browser. The most important of these controls is UpdatePanel, which allows updating only part of a user-visible page rather than replacing the whole thing. The ASP.NET 2.0 AJAX Extensions also provide Web services that browser code can invoke to access information provided by the application.

For clients, ASP.NET AJAX provides the Microsoft AJAX Library. This library implements a framework for creating client scripts, and it runs in Internet Explorer, Firefox, Opera, and Safari. As in all AJAX implementations, this client code can use an object provided by the Web browser called XMLHttpRequest. By calling a method in this object, a script can request data via HTTP from an application built using ASP.NET or another technology, such as PHP. This request can be made asynchronously, allowing data to be retrieved without forcing the user to block while waiting for a response. The data it returns can then be accessed as XML using the Document Object Model (DOM). Along with XML, ASP.NET AJAX also allows exchanging data represented using the JavaScript Object Notation (JSON).

The AJAX style of Web application was a good idea when the notion first surfaced in 1999. While we’ve had to wait several years for this approach to find its way into the mainstream, AJAX applications are now becoming the norm. Microsoft’s decision to include ASP.NET AJAX in the .NET Framework 3.5 is certain to help speed adoption of this useful approach.

## Language-Integrated Query

Creating a common approach for accessing diverse data isn’t an easy task. Making that approach comprehensible to developers—not bogging them down in more complexity—is even harder. To address this, LINQ relies on a common and quite familiar syntax for working with data.

Here’s a simple LINQ query, expressed in C#, that reads data:

var exampleQuery =

from s in Students

where s.Gender == "M"

select s.Name

This query returns the names of all male students as a list of strings stored in the variable exampleQuery. To print out that list, a program might do this:

foreach (string name in exampleQuery) {

Console.WriteLine(name);

}

The syntax of the query is reminiscent of SQL, today’s standard language for accessing relational data. This makes sense, since SQL is a widely used language, one that many developers know. Yet it’s important to understand that even though it looks somewhat like SQL, the LINQ query shown above isn’t an embedded SQL statement. Instead, it’s pure C#, part of the language itself. This means that the query can use other program variables, be accessed in a debugger, and more. The name of this technology is “Language-Integrated Query” for a reason: the statements for querying diverse kinds of data are integrated directly into the programming language.

And despite its similarity to SQL, this example query isn’t limited to accessing only relational data. In fact, the .NET Framework 3.5 includes several different LINQ variations, all of which use the same basic syntax for queries. Those variations include the following:

1. LINQ to ADO.NET: Provides object/relational (O/R) mapping. This aspect of LINQ includes two options. The first, LINQ to SQL, translates a query like the one above into a SQL query, then issues it against tables in a SQL Server database. The second option, LINQ to DataSet, executes a query like the one shown above on the contents of a DataSet returned by ADO.NET
2. LINQ to Objects: Allows querying collections of in-memory data structures, such as object hierarchies. The query above could also be issued against this kind of data—despite its similarity to SQL, it’s not limited to accessing only data in tables.
3. LINQ to XML: Allows querying XML data. This approach uses a slightly different syntax, reflecting the unique needs of mapping between the world of XML and a programming language, but the basic structure of a LINQ query remains the same.

Like SQL, LINQ also defines other operators for queries. They include things such as OrderBy, which determines how results are ordered; GroupBy, which organizes selected data into groups; and arithmetic operators such as Sum. And once again, these can be used generally across the LINQ varieties—they’re not just for the LINQ to SQL option.

LINQ’s creators aimed at several targets, including providing O/R mapping for .NET applications, allowing a common syntax for working with different kinds of data, integrating that syntax directly into the programming language, and more. As with everything else described in this introduction, the goal is to make life better for developers working with Visual Studio 2008 and the .NET Framework 3.5.

## Windows Communication Foundation

The change to service-oriented communication marks a shift in how applications interact. Explicitly designed to support service-oriented applications, WCF reflects this shift. This section describes the most important aspects of WCF, including services and clients, communication options, and support for security, reliable communication, and transactions.

### Services and Clients

The fundamental idea of WCF is simple: a service exposes an interface that can be accessed by a client. That interface can be defined using the Web Services Description Language (WSDL), then turned into code, or it can be defined directly in a language such as C# or Visual Basic. For a simple interface exposing an insurance application service, the latter approach might look something like this:

[ServiceContract]

interface IInsurance

{

[OperationContract]

int Submit(int policyType, string ApplicantName);

[OperationContract]

bool CheckStatus(int applicationNumber);

[OperationContract]

bool Cancel(int applicationNumber);

}

The definition of this C# interface is marked with the **ServiceContract** attribute. This attribute indicates that WCF can expose methods in this interface as remotely-callable operations. Which of the interface’s methods are exposed depends on which are marked with the **OperationContract** attribute. In this simple example, every method is marked with this attribute, and so all of them will be exposed to remote callers. This isn’t required, however—it’s legal to apply **OperationContract** to only some of an interface’s methods. Whichever choice is made, some class in the application must implement this interface, providing actual code for the methods the interface defines. Once this is done, WCF automatically makes the methods marked with **OperationContract** accessible to clients of this service.

The figure below gives a slightly more detailed view of how a service is actually exposed to its clients. Rather than access an interface directly, a client instead connects to a specific *endpoint*. A service can expose multiple endpoints, potentially allowing different clients to access it in different ways.

As the figure shows, each endpoint specifies three things:

1. An *address* indicating where this endpoint can be found. As shown in the figure, the address is expressed as a URL.
2. A *binding* defining how the endpoint’s operations can be invoked. Each binding defines several things, including what protocol should be used to invoke the operations, what kind of security should be used, and more. WCF includes a set of pre-defined bindings, such as the BasicHttpBinding shown here, for the most common cases, and it’s also possible to define custom bindings. Because a single service can expose multiple endpoints, each with a different binding, it can simultaneously allow access to different kinds of clients over different protocols and with different security options.
3. A *contract* describing the operations that can be invoked via this endpoint. The contract can be identified using just the name of the interface that defines these operations, which here is IInsurance.

The basics of WCF are simple. As with most communication technologies, the details can get complex—there are many options—but creating ordinary WCF applications isn’t hard.

### Support for REST and RSS/ATOM

In its original release in the .NET Framework 3.0, WCF’s Web services support was focused on SOAP. Another approach to Web services can be better in some cases, however. Known as Representational State Transfer (REST), it’s built directly on the technology of the Web itself. Rather than defining a new protocol, as did the creators of SOAP, REST relies on the built-in operations in HTTP: GET, POST, and others. And rather than identify the information to be accessed with parameters defined in XML, as SOAP typically does, REST assumes that everything is identified with a URL.

To support this approach, WCF in the .NET Framework 3.5 adds a new binding, WebHttpBinding, that sends information directly over HTTP. It also adds two new attributes, as this example shows:

[ServiceContract]

interface IAccount

{

[OperationContract]

[WebGet]

int GetBalance(string account);

[OperationContract]

[WebInvoke]

int UpdateBalance(string account, int amount);

}

This simple interface exposes a RESTful service for reading and updating a bank balance. To indicate that a call to GetBalance should be conveyed directly via an HTTP GET, this operation is marked with the WebGet attribute. Similarly, because the UpdateBalance operation is marked with the WebInvoke attribute, it will be conveyed via an HTTP POST. (It’s also possible to specify that an operation marked with this attribute should use some other HTTP operation, such as PUT or DELETE.) Rather than using SOAP and the WS-\* protocols, this simpler RESTful style adheres to the traditional fundamentals of Web-based communication. Because each approach makes sense in some situations, WCF in the .NET Framework 3.5 supports both.

The ability to send the XML formats defined by RSS and ATOM has also become more important since WCF first appeared. Accordingly, the .NET Framework 3.5 version of WCF provides built-in support for these two formats. Information structured using either one can be sent over any WCF binding, although the new WebHttpBinding is likely to be the most common choice. Using WCF-supplied formatters, a .NET Framework 3.5 application can create or consume an RSS or ATOM feed, making it straightforward to work with this increasingly important kind of content.

### Communication Options

Different kinds of applications built by different kinds of developers need different ways to communicate. The simplest approach for most developers is remote procedure call (RPC), which lets a client invoke remote operations much like local ones. Given the interface shown earlier, for example, a client could invoke any operation in the usual synchronous way, waiting patiently until a response comes back. This option is easy for developers, and it’s the right choice in some circumstances.

WCF also provides several other options, however. They include the following:

1. Calls that have no response. Marked with the attribute **OneWay**, this kind of communication can be useful for sending events or other one-way interactions.
2. Asynchronous message-based communication, using direct sends and receives over MSMQ.
3. Explicit manipulation of SOAP messages, including the ability to insert elements directly in the SOAP header.

WCF also allows a developer to control various local aspects of how a service behaves. Using the **ServiceBehavior** attribute, for example, she can set whether the service is single or multi-threaded, whether a new instance of the service is created for each call, and other options.

### Security, Reliability, and Transactions

Basic communication—the ability to move data between systems—is useful, but it’s rarely enough. Most applications need more. For example, the great majority of distributed applications need some kind of security. Providing security can be complex, given the range of different approaches and diversity of technologies in use today. To let developers create secure distributed applications without forcing them to understand all of the details, WCF relies primarily on bindings for security. For example, the BasicHttpBinding shown earlier can be configured to use HTTPS rather than plain HTTP, and other bindings provide more security options. WsHttpBinding, for instance, supports WS-Security, allowing interoperable SOAP-based authentication, data integrity, and data confidentiality. A developer can also create a custom binding that provides the exact security services his application needs.

Making sure that communication is reliable is also essential for many applications. The traditional Web services approach, sending SOAP over HTTP, is sufficient in some cases, and it’s what’s done when the BasicHttpBinding is used. There are plenty of situations, however, where this widely-used choice isn’t enough. Messages that go through one or more SOAP intermediaries, for instance, can’t rely on this simple approach for end-to-end reliability. For these cases, WCF implements WS-ReliableMessaging. By choosing a binding that supports this option, such as WsHttpBinding, a developer can automatically get interoperable reliable message transfer.

Distributed transactions can also be important in some applications. WCF builds on System.Transactions, originally released in the .NET Framework 2.0, to allow creating transactional software. A method can use the **OperationBehavior** attribute to indicate that it requires a transaction and to define how that transaction behaves. To allow distributed transactions that are interoperable across vendor boundaries, WCF relies on the WS-AtomicTransaction specification. Using the technology defined in this multi-vendor agreement, WCF applications can participate in transactions that span diverse technologies.

### Tool Support

Good tools make every developer’s life better. Accordingly, the version of WCF included in the .NET Framework 3.5 has support in Visual Studio 2008 for things such as the following:

1. Project types to help developers get started creating WCF applications. These include projects for building:
2. A Website or Web application that hosts a WCF service;
3. A library implementation of a WCF service;
4. A WCF application that exposes syndication feeds using RSS or ATOM;
5. A WCF service designed for use by an AJAX client. This service is automatically configured to use WebHttpBinding with the JavaScript Object Notation (JSON), one of the encoding choices this binding provides.
6. A WCF Autohost that can automatically host a library-based WCF service.
7. The Service Configuration Editor, a tool that makes creating and modifying WCF configuration files easier.

### Windows Communication Foundation and Other Microsoft Technologies

As mentioned earlier, WCF supersedes several earlier Microsoft technologies for creating distributed applications. Most applications that would have been built using ASP.NET Web Services, .NET Remoting, Enterprise Services, or WSE will instead be built on WCF. WCF applications can interoperate with ASP.NET Web Services applications—both support standard SOAP—as well as applications built on Enterprise Services, MSMQ, and version 3.0 of WSE. And even though new .NET Framework 3.5 applications won’t commonly use them, all of the technologies that WCF supersedes are still part of this version of the Framework, and they’ll all be supported as usual. Applications built using earlier versions of these technologies will continue to run normally; installing and using the .NET Framework 3.5 won’t break existing code.

## Windows Workflow Foundation

A process-oriented design, driven by a workflow, can be the right approach for a significant fraction of Windows software. The purpose of WF is to let developers create and execute these workflow-based applications. The diagram below shows the components WF provides to do this.

As described earlier, every workflow is built from some number of activities. Workflows and activities are just classes, so both can be created directly in code. WF also provides the Workflow Designer, a Visual Studio-hosted graphical tool for constructing workflows. However a workflow is created, its activities can be drawn from the Base Activity Library (BAL) provided with WF or from any other source.

Once a workflow has been defined, it’s eventually executed by the WF runtime engine. This engine relies on a group of runtime services for persisting the workflow’s state, tracking the workflow’s execution, and more. All of these things—the runtime services, the runtime engine, and the workflow itself—are contained within some host process. This process can be any Windows process, ranging from a simple console or WPF application running on a desktop to a scalable server process.

Understanding WF requires knowing at least a little about all of its components. The following sections take a brief look at each one.

### Workflows

Stripped to its essentials, a workflow is nothing more than a group of activities. WF provides built-in support for two styles of workflow:

1. Sequential workflows, which execute activities in a defined order. Like a traditional flow chart, a sequential workflow can contain branches, loops, and other control structures. By default, however, activities execute in sequence, one after another.
2. State machine workflows, which implement a traditional finite state machine. Like any state machine, which activity executes at a particular time is determined by the combination of the current state and whatever event has been received.

The sequential option is useful for well-defined workflows, such as those used in purely software-based processes. They’re relatively simple to create and understand, and they initially feel more natural to most developers. State machine workflows are a better choice when the path of execution is less predictable. A good example of this is a workflow that involves interactions with people, any of whom can cancel the workflow at any point. Addressing this situation with a sequential workflow is possible, but every step could be a branch: Do this if the workflow isn’t cancelled, do something else if it is cancelled. Modeling this kind of behavior using a state machine is significantly simpler, since a request to cancel the workflow is just another event that can be received and handled at any point.

Support for state machine workflows is one example of how WF attempts to provide support for human as well as system workflow. Another example of this is WF’s support for changing a running workflow. People can be capricious, and it’s not uncommon for someone involved in a workflow to wish to add a step, delete a step, or make some other change in the process while it’s underway. To accommodate this in a controlled way, WF allows the developer who creates a workflow to specify whether and how that workflow can be modified while it’s executing.

### The Base Activity Library

Developers are free to create custom activities. In fact, Microsoft’s goal is to foster the growth of a WF ecosystem full of reusable activities. Still, starting with a common set of fundamental activities makes life simpler for everyone. Providing this common set is the role of the Base Activity Library (BAL).

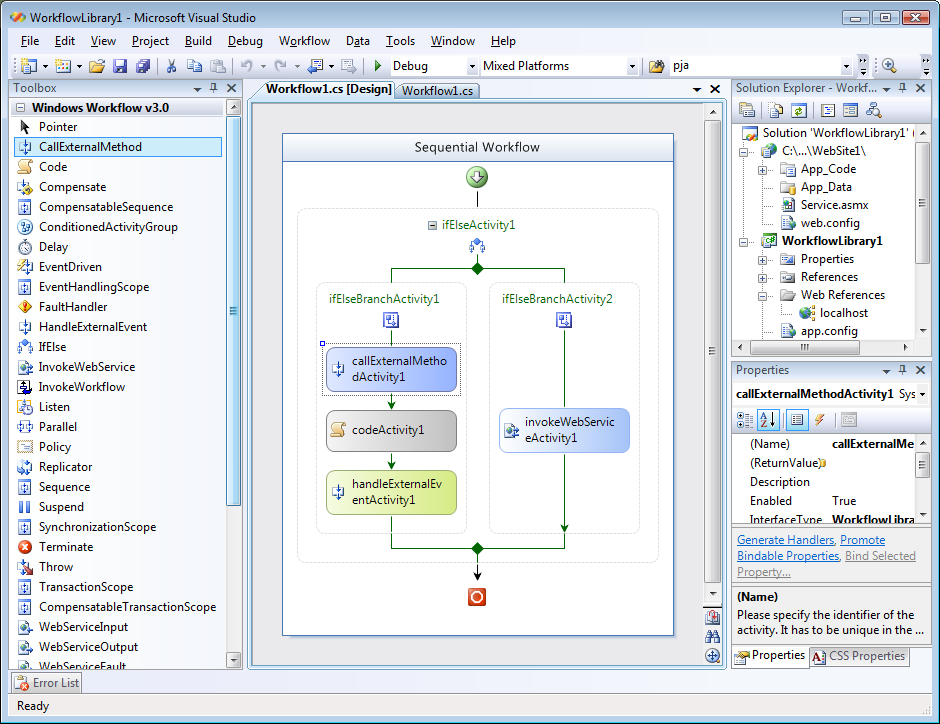
A workflow isn’t required to use anything from the BAL. Still, many developers will find that the BAL makes their lives simpler, especially at first. Among the activities contained in the BAL are the following:

1. IfElse: executes the activities contained in two or more possible paths based on whether a condition is met.
2. While: repeatedly executes one or more activities as long as a condition is true.
3. Sequence: executes a group of activities one at a time in a defined order.
4. Parallel: executes two or more groups of activities in parallel.
5. Code: executes a defined chunk of code.
6. Listen: waits for a specific event, then executes one or more activities when that event is received.
7. InvokeWebService: calls a Web service using ASP.NET Web Services.
8. State: represents a state in a workflow’s state machine.
9. EventDriven: defines a transition containing one or more activities that should be executed when a specific event is received while in a particular state.
10. Policy: allows executing business rules using a WF-supplied rules engine.

Rather than define a particular language for specifying workflows, WF instead takes the more general approach of using activities. The BAL provides one “language”, but anybody using WF is free to define his own as well.

### Tools for Windows Workflow Foundation: The Workflow Designer

One of the advantages of creating applications using workflows is the ability to define the workflow graphically. WF’s Workflow Designer allows this, as shown below. By default, the activities in the BAL appear in the Toolbox, letting a developer drag and drop them onto the tool’s design surface to create a workflow.



Some developers prefer to write code—they don’t like graphical designers. WF allows this approach, too (and sometimes requires it: activities are generally built directly in code). It’s also possible to combine the two approaches, creating a workflow using both the Workflow Designer and direct coding. The goal is to let developers use the approach that’s most productive for them. And to allow broad er tool support, workflows can also be expressed in XAML, the same language used by WPF. In fact, workflows created using the Workflow Designer default to a XAML definition.

### The Runtime Engine and Runtime Services

As described earlier, the WF runtime engine has the job of executing the activities in a workflow. As part of doing this, it relies on a group of runtime services. WF includes standard implementations of these services, but ambitious developers can replace them if desired. These services support a few different things, but two stand out as most interesting:

1. Persistence: A workflow that’s blocked waiting for some event can use this service to have its in-memory state automatically persisted to disk. When the event occurs, the service will automatically reload the workflow’s state and restart execution. This is especially useful for workflows that involve people, since hours, days, or more might elapse while waiting for a response.
2. Tracking: The activities in a workflow cleanly demarcate the execution of the process they implement. WF’s tracking service allows a developer to cause information about the workflow’s execution to be automatically written to a database. For example, a developer might wish to track when a workflow starts and ends, when each of its activities starts and ends, and other information.

### Workflow-Enabled Services

WF and WCF make an obvious combination. Workflows will commonly need to invoke services, and implementing a service with a workflow can often be a good idea. The initial release of WF in the .NET Framework 3.0 didn’t make combining these two technologies especially easy, but the situation has improved significantly in the .NET Framework 3.5. In this version, it’s straightforward to use WF and WCF together to create workflow-enabled services.

This combination depends on two new WF activities:

1. Send: Sends a request using WCF, then waits for a response. A developer specifies the operation that should be invoked and the endpoint at which that operation can be found.
2. Receive: Receives an incoming request via WCF, then sends a response. The developer specifies just the name of the operation that accepts this incoming request.

These two activities can be dragged and dropped into a WF workflow like any others, then configured as needed. The intent is to allow combining WF and WCF in useful ways.

### Windows Workflow Foundation and Other Microsoft Technologies

Introducing new approaches inevitably influences what already exists. With WF, this influence has been felt most strongly by Windows SharePoint Services, the Microsoft Office 2007 system, and BizTalk Server.

To let developers more easily create workflow applications for document collaboration and other kinds of information sharing, Windows SharePoint Services, version 3 hosts the WF runtime. Office SharePoint Server 2007, part of the Office 2007 system, builds on the WF support in Windows SharePoint Services. Among other things, adding this server allows displaying InfoPath forms directly in Office 2007 client applications and using a set of pre-defined workflows for common scenarios such as approving a document.

Anyone familiar with BizTalk Server has certainly noticed by now the similarity between this product’s orchestration capabilities and what WF provides. In fact, the next major release following BizTalk Server 2006 R2 is scheduled to replace the product’s existing orchestration function with WF, providing tools to help migrate existing orchestrations to WF workflows.

Because it’s the standard workflow technology for Windows, WF will also show up in other Microsoft products and technologies. WF has also found a home in a number of applications created by independent software vendors (ISVs) and enterprises. While not every Windows application should be built as a workflow, WF can make a developer’s life much easier for those that are.

## Windows Presentation Foundation

Both service-oriented communication and workflow-based logic are important in modern applications. Yet users often care most about what they see: the user interface. The goal of WPF is to address the challenges of creating user interfaces for modern applications. As described next, WPF provides a range of capabilities to do this.

### The Capabilities of Windows Presentation Foundation

A developer is free to create a WPF application’s interface entirely in C#, Visual Basic, or some other CLR-based language. As described earlier, however, WPF also allows specifying an interface using the XML-based XAML. Elements and attributes in XAML map directly to the classes and properties that WPF provides. For example, here’s a simple button defined in XAML:

<Button Background="Red“>

No

</Button>

This example creates a red button that contains the text “No”. The exact same result could also be produced with the following code:

Button btn = new Button();  
btn.Background = Brushes.Red;  
btn.Content = “No";

However it’s defined, a WPF application can rely on *panels* for basic layout. Each panel typically contains *controls*, and those provided by WPF include Button, TextBox, ComboBox, Menu, and many more. How those controls are positioned depends on what type of panel is chosen. A Grid, for instance, allows positioning controls on a specified grid, while a Canvas lets the developer place controls anywhere within its boundaries. And as usual in a GUI, events generated by the user are caught and handled by the various controls and other classes in the application. It’s also possible to apply styles and templates to groups of controls, which makes it easier for applications to have a uniform look.

WPF supports much more than these basic user interface functions, including the following:

1. Documents: A WPF application can display XPS documents using XAML’s FixedDocument tag. An application can also display *flow documents* using the FlowDocument tag. A flow document can behave like a traditional on-screen document, allowing the user to scroll through its contents. By setting various attributes on this tag, however, a developer can make the document more adaptive to its surroundings. A document can display a page at a time, for example, freeing its reader from the need to scroll back and forth. The goal is to allow on-screen documents to be as readable as possible.
2. Graphics: WPF includes support for creating two-dimensional and three-dimensional vector graphics. For 2D work, WPF provides standard abstractions such as shapes, brushes, and pens, while 3D graphics allowing defining a model that can be assigned lighting and camera position information. Unlike earlier technologies such as Windows Forms, which relies on GDI+ for graphics, WPF graphics aren’t cabined off using a separate set of concepts that developers must understand. Instead, the XAML elements used for graphics can be combined naturally with those used for anything else in a user interface.
3. Images: Using XAML’s Image tag, a WPF application can display images in various formats, including JPEG, GIF, and others.
4. Media: A WPF application can use the MediaElement tag to display video and audio in various formats, including WMV, AVI, and MPEG.
5. Animation: WPF provides built-in support for animating most parts of a user interface. A circle can grow and shrink, for example, or a button can smoothly change size. Applications can also define storyboards containing timelines, allowing coordinated sequences of animations to occur.
6. Data Binding: Because many WPF applications display data, it’s useful to have automatic support for mapping data to user interface elements. WPF provides this kind of data binding for information contained in objects and other sources.

### Applying Windows Presentation Foundation

WPF provides a large set of user interface functionality, allowing developers and designers to create more appealing user interfaces. Yet no matter how lovely a client application looks, some organizations might resist using it because of deployment issues. If rolling out a new version of the client requires physically touching every desktop machine that this application is installed on, the cost of upgrades can be significant. One common way to avoid this problem today is to create browser-based clients, perhaps using AJAX, rather than native Windows clients. Yet even with AJAX, native Windows clients can often present better, more responsive user interfaces than browsers. To address the challenge of deploying these clients, standalone WPF applications can be deployed using ClickOnce. A technology that first appeared in the .NET Framework 2.0, ClickOnce lets Internet Explorer users select an application via the Web, then have it automatically installed on their local machine. Once installed, the application can also be automatically updated when a new version is made available. The goal is to combine the simplicity and inexpensive deployment of a Web client with the power and functionality of a standalone WPF application.

Especially when they’re deployed using ClickOnce, standalone WPF applications are a good choice in many situations. Alternatively, a user can download an XBAP directly into her browser. This application can then present the user with a WPF-based user interface. To protect the user from malicious developers, all XBAPs downloaded from the Internet run in a partially trusted sandbox. Based on the code access security provided by the .NET Framework, this sandbox limits what XBAPs can do. Yet despite the restrictions imposed by the sandbox, an XBAP can still use a large fraction of WPF functionality, including 2D and 3D graphics, animations, on-screen documents, images, video, and more.

As described earlier, WPF lets applications display adaptive documents using XAML’s FlowDocument element. Documents whose appearance changes based on how they’re displayed aren’t always the best solution, though. Fixed-format documents, which always look the same on a screen and a printer, are sometimes a better choice. WPF’s XPS documents address this problem. Defined using a subset of XAML, XPS documents can be read on any system that provides an XPS reader. They also provide a new print format for Windows, allowing complex graphics to be printed with more fidelity.

### Tools for Windows Presentation Foundation

It’s possible to create any WPF user interface directly in code and/or XAML using a basic text editor. Most people would prefer to work with better tools, however, and so Visual Studio 2008 provides the WPF designer. Using this tool, a developer can graphically create the user interface she wishes to see, then let the tool generate the code for this interface.

Yet developers often aren’t the best people to define user interfaces. Designers are usually much better at this kind of work, since they specialize in communicating with people. Most designers don’t write code, though, and so the WPF designer isn’t an effective tool for this group. To let designers work effectively in the WPF world, Microsoft has instead created Expression Blend. The figure below shows how these tools relate to one another.

As the figure illustrates, a designer can use Expression Blend to define the look and feel of an interface, specify animations, and more, then let the tool generate a XAML version of what he’s created. A developer can import this XAML into Visual Studio and add code for things such as handling events. Because both Visual Studio and Expression Blend use the same build system, it’s possible for developers and designers to work on a project iteratively, each using the tool they’re comfortable with. The goal is to help people from the divergent disciplines of design and software engineering work together effectively.

### Windows Presentation Foundation and Other Microsoft Technologies

Like other .NET Framework 3.5 components, WPF has a relationship with other Microsoft technologies. The most important of these are the following:

1. Windows Forms: The .NET Framework’s original approach for creating application GUIs, Windows Forms is used in many current applications. Even with the advent of WPF, Windows Forms is still the right choice for creating some new software, such as line-of-business (LOB) applications. To let these technologies be used together, WPF applications can host Windows Forms controls, while WPF controls can be hosted in Windows Forms applications. For example, a Windows Forms application might host a WPF control that provides three-dimensional data visualization, or a WPF application might use the DataGridView control provided in Windows Forms. While there are some limitations, creating applications that use both technologies is certainly possible.
2. Silverlight: While XBAPs allow creating WPF-based interfaces that run in browsers, they’re limited to Windows systems that have the .NET Framework installed. Rich internet applications (RIAs) typically need to be accessible from any kind of client. To meet this need, Microsoft provides Silverlight. This freely downloadable technology provides a subset of full WPF functionality, including 2D graphics, animation, and video, and it runs in a variety of browsers on Windows, Macintosh, and (through Novell) Linux systems.
3. ASP.NET and ASP.NET AJAX: Windows applications can use WPF (or perhaps Windows Forms) for their user interface, while multi-platform RIAs can use Silverlight. One more option is to provide a standards-based Web interface using only what the browser provides—no download is necessary. This is exactly what’s done by ASP.NET and ASP.NET AJAX. This kind of interface isn’t as good at providing some features of an RIA, such as video, but especially with AJAX, it can still offer a responsive and effective user experience.

## Windows CardSpace

Whether it’s via a Web browser or some other kind of client, users routinely access applications across a network. Since these applications commonly require their users to identify themselves in some way, the inescapable corollary is that people are regularly forced to acquire and present identity information to remote software. One very common example of this is accessing Internet applications via a browser.

As described earlier, people most often rely on usernames and passwords to express these digital identities today, with all of the problems this entails. Windows CardSpace, part of the larger identity metasystem, offers an alternative approach to addressing these problems. To get a deeper understanding of how CardSpace does this, the place to start is with the basic concepts of the identity metasystem.

### Windows CardSpace and the Identity Metasystem

When a user accesses an application, whether from a Web browser or an application-specific client or something else, she commonly presents some kind of digital identity. Digital identities come in many varieties, but virtually all of them are represented on the network by a *security token*. While a simple security token can be just a username and password, a more complex token might include an X.509 certificate or an XML document. However they’re represented, security tokens are today’s typical mechanism for representing digital identities on the network.

While it’s tempting to believe that we’ll all one day adopt a common security token format, the reality is that diverse approaches will continue to be used. Just as we today carry multiple identity cards in our wallet—drivers license, credit card, airline frequent flyer card, and more—we’ll always have diverse digital identities represented by diverse kinds of security tokens. No single identity system can provide a universal answer, and so multiple security tokens will always be necessary.

Yet users still need some way to work consistently with their diverse digital identities. Even though no single identity system will suffice, it’s possible to create a *system* of identity systems—an identity metasystem—that allows using a myriad of digital identities in a consistent way. Working with others, Microsoft has led the process of defining this metasystem. Based on open Web services technologies, such as WS-Security and WS-Trust, this metasystem defines how digital identities can be acquired and used, regardless of the type of security token they depend on.

The process of issuing, acquiring, and using digital identities can be thought of as requiring three distinct roles. Those roles are the following:

1. User: More formally known as the *subject*, the user is the entity that has a digital identity.
2. Identity provider: An identity provider supplies a digital identity for a user. For the digital identity you use at work, for example, the identity provider is your employer, and your security token is likely issued by a system such as Active Directory. For the digital identity you use with Amazon, the identity provider is effectively you, since you define your own username and password. Digital identities created by different identity providers can carry different information and provide different levels of assurance that the information they contain is accurate.
3. Relying party: A relying party is an application that in some way relies on a digital identity. A relying party will frequently use an identity (that is, the information contained in this identity’s security token) to authenticate a user, then make an authorization decision, such as allowing this user to access some information. A relying party might also use the identity to get a credit card number, to verify that the same user is accessing it at different times, or for other purposes. Typical examples of relying parties include Internet Web sites, such as banks, online merchants, and auction sites, and any application that accepts requests via Web services.

These three kinds of entities interact in the identity metasystem. The figure below illustrates these interactions, along with where CardSpace fits in.

The process begins when a user accesses a relying party via a CardSpace-aware application. To learn what type of security tokens this relying party will request, the application must get the relying party’s policy (step 1). For a browser accessing a Web site, which is the most common case today, the site’s policy is expressed in HTML and sent back as part of a Web page. For applications accessed via Web services, however, the application instead uses the industry-standard protocol defined by WS-MetadataExchange to ask the relying party for its policy. In this case, the policy is expressed using WS-SecurityPolicy, another industry standard. However the policy information is acquired, it always indicates what kind of security tokens this relying party will accept and what information those tokens must contain.

Once CardSpace knows what kind of security tokens the relying party needs, it displays the identity screen shown earlier. Each digital identity available to this user is represented as an information card on this screen. Cards issued by an external relying party are referred to as *managed* cards, while those issued by CardSpace’s self-issued provider are known as *self-issued* cards. Both kinds of cards can be shown on this screen, and the user can potentially select one of either type. To make this decision easier, the screen indicates which identities meet the requirements of the relying party by graying out cards that don’t qualify. The user then selects one of these as the digital identity he wants to use (step 2).

A card doesn’t contain an actual security token, however. Instead, it holds the information necessary to locate a particular identity provider and request a security token for this user. (In fact, each card is originally created by some identity provider.) CardSpace uses the contents of the card selected by the user to request a security token from the identity provider that issued that card (step 3). This request is made using WS-Trust, another industry-standard protocol, and the user authenticates himself to the identity provider using Kerberos, an X.509 certificate and digital signature, or another mechanism. The token is returned in an encrypted form, which also contains a timestamp to prevent this token from being stolen off the wire and reused in the future.

Once the requested security token is returned, it’s sent to the relying party (step 4). How the relying party uses the information in the token can vary. If the token contains an X.509 certificate, for example, and is accompanied by a digital signature, the relying party will likely use the token to authenticate the user. There’s no requirement that tokens be used for authentication or for any other security-related purpose, however. A token might carry information such as proof of a user’s age, eligibility for a discount at an Internet shopping site, or anything else. Authentication is one important use for security tokens, but it’s not the only option.

It’s important to note that CardSpace need not be aware of the format or technology used for the security token. Rather than trying to create a new single source for digital identities or a standard format for security tokens, the goal of the metasystem is to provide a coherent way to use any digital identity based on any kind of security token. By providing a Windows implementation of key parts of the metasystem, CardSpace plays an important role in making this general approach to digital identity a reality.

### Fighting Phishing

Identity providers are often distinct from the user, such as when an identity is assigned by an employer. Yet there are plenty of situations where the identity provider is actually the user herself. If CardSpace isn’t used, for instance, accessing many Web sites requires giving a username and password, both of which are defined by the user. Once the user has created this identity, she can later supply the username and password, then check her bank balance, buy a book, or do whatever else this site allows.

Yet because they depend on passwords, these simple self-issued identities are targets for attackers, as described earlier. To help reduce these attacks, CardSpace provides an alternative way to create a self-issued identity. This self-issued identity provider runs locally on the user’s Windows system. Rather than relying on a username and password, the security tokens created by the self-issued identity provider are defined using the Security Assertion Markup Language (SAML), an OASIS-defined standard. These tokens rely on public key technology rather than passwords to prove the user’s identity, and if a relying party accepts them, they can play the same role as the traditional username and password. The advantage is that there is no longer a password for phishers to steal. Reducing the use of passwords can make phishing a much less serious problem.

### Windows CardSpace and Other Microsoft Technologies

CardSpace is related to several other Microsoft technologies, including the following:

1. WCF: Because it relies on Web services standards such as WS-Security and WS-Trust, CardSpace uses WCF for communication. In fact, the creator of a WCF application can cause that application to use CardSpace just by specifying a particular binding.
2. Active Directory: Although it’s not currently possible, Active Directory will eventually be able to act as an identity provider in the metasystem. Once this is possible, expect to see Windows CardSpace used inside enterprises as well as on the Internet.
3. Windows Live ID: Microsoft’s Live ID identity system is capable today of acting as a managed identity provider for Windows CardSpace. Note that CardSpace isn’t a replacement for Live ID (formerly known as Passport), since the two address quite different problems.

# Conclusion

The .NET Framework 3.5 is the latest incarnation of the mainstream Windows programming environment. Built on and extending its predecessors, its goal is to support the creation of modern applications. By building its various technologies on a common foundation, Microsoft is striving to make the whole greater than the sum of the parts, letting developers create applications that use the various parts of the .NET Framework 3.5 in a coherent way.

From its first release in 2002, the .NET Framework has matured into the dominant platform for new Windows applications. Used widely today by enterprises, ISVs, and Microsoft itself, the Framework has proved its value. Together with Visual Studio 2008, the .NET Framework 3.5 represents Microsoft’s current state of the art in application development.

# About the Author

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps IT professionals around the world understand, use, and make better decisions about enterprise software.