Issue	November 2001
Section	
Main file name	Nu0001at4.doc
Listing file name	
Sidebar file name	
Table file name	
Screen capture file names	Nu0001f1.bmp
Infographic/illustration file names	
Photos or book scans	
Special instructions for Art dept.	
Editor	
Status	
Spellchecked (set Language to English U.S.)	
PM review	
Character count	
Package length	
ToC blurb	.NET radically transforms existing programming models. Learn how this overhaul can change how your organization works.

Overline:

Byline:
By Juval Lowy

Head:

.NET Overhauls App Frameworks

Deck:

This new technology jump starts e-commerce.

Microsoft's .NET doesn't simply tweak a technology here and enhance another one there. It's not about patching something together with bailing wire or duct tape—this bad boy starts from the ground up and rebuilds completely most of Microsoft's existing technologies. .NET provides a new set of application-development frameworks, including Web Services, ASP.NET, Windows Forms, ADO.NET, and Enterprise Services..

In this article you'll learn .NET's application frameworks core concepts, along with its benefits and advantages. Whenever possible, I'll contrast .NET with existing application frameworks. I'll discuss the objectives of the frameworks, so you'll know what the application frameworks are all about and be able to make educated decisions on adopting them in your organization.

Here are some of the changes: As the framework for developing Windows user-interface applications, .NET Windows Forms replace Microsoft Foundation Classes (MFC) and Visual Basic 6.0. ASP.NET replaces ASP for creating dynamic Web pages, and ADO.NET replaces ADO/OLEDB for data access. A new set of languages, particularly C# (pronounced C-Sharp) replace Visual C++ (and C++ itself). VB6 has been completely deprecated. You won't need to worry about Windows programming (Win32 API) because.NET exposes its own rich set of application-programming classes and interfaces. .NET has built-in support for developing components, so COM/DCOM and ATL also disappear.

The changes from an existing technology to the .NET equivalent, such as from ASP to ASP.NET, are substantial. The new technology is radically different, not just in syntax but also in the programming model itself. But .NET has more to it than improving existing programming models and technologies. The most exciting new capability is its support for Web Services. .NET Web Services support can potentially transform e-commence into the next information-technology revolution. In addition to the technology itself, the development environment's user interface (called IDE, for Integrated Development Environment) differs drastically from all previous versions of Visual Studio's environment.

The IDE

The Visual Studio IDE looks overwhelming at first. However, you can customize the IDE, and you can select from several available IDE templates, such as Visual C++ 6.0, to ease the transition. Many changes from the VS6 IDE provide you with more rapid development and better tools. The new IDE offers improved ergonomics, including toolbars that appear only when you require them and are minimized when not in use. The IDE is better integrated. All .NET languages, such as C# and VB.NET, now use the same development environment, and the IDE can display almost any development-related information such as system processes, threads, message queues, performance counters, system services, Web Services, control panel applets, database tables, and query results.

On of the most impressive and important improvements in the IDE is the debugger. It includes all the old windows, such as call stack and threads, in addition to new windows, and you can tab and dock all of them together. The class view is also improved, and it now shows namespaces, classes, base classes, methods and properties, interfaces, and available override-ables, which are virtual methods from the base class. A new object browser—similar to the one available in VB6—can display information about your project or any other assembly, often saving you the trouble of using Help. You edit most things using the enhanced properties window. VS6 only allowed you to store one item in the clipboard; you can now store as many clips as you like in the Clipboard Ring (see Figure 1).

Collapse Code

But the IDE has many new features, too. The code editor can collapse blocks of code and hide them from view, similar to collapsing folders in the file Explorer, allowing you to focus only on the method you want. The code editor can also present line numbers. The IDE checks what you type constantly, and if it finds a compilation problem, it underlines the code the way Word's spellchecker does when you type misspelled text. You'll see compilation warning and errors as code underlining, and you can get the warning or error as a tool tip on that underlining.

A special help window keeps track of your activities and suggests relevant help links dynamically. A task list window lists remaining tasks such as complication errors or any other entries the developer puts in. You also have built-in support for documentation. If you enter inline comments in your code using special XML tags, VS.NET can present you with auto-generated, well-formatted Web page documentation of your project.

Web Services Support

.NET also offers Web Services, which now allow a middle-tier component at one site to invoke methods on a middle-tier component at another site as easily as if the remote component were local. Today, Web sites have a hard time interoperating among themselves. Even though most Web sites are the same architecturally, and a given system is more or less homogeneous in technology (platform, language, component technology), components developed in one technology can't invoke methods on components developed in another technology.

Firewalls also disallow binary calls. Developers today handcraft interoperability solutions between businesses. These solutions are proprietary, they provide singular relief, and they're expensive and time-consuming to implement. Web Services address these problems elegantly and simply.

The underlying technology facilitating Web Services is a serialization of the calls into XML packages using protocols such as Simple Object Access Protocol (SOAP) or HTTP GET/POST. XML-based calls are plain text, so you can make them across firewalls. This fact makes them an ideal transport mechanism for Web Services calls. .NET support for Web Services completely encapsulates the underlying Web Services complexity from both the service implementor and its consumer. Using .NET Web Services, you can create an application that combines Web Services from multiple Web sites, so you can view an entire Web site as one component in your application.

.NET hides the required details successfully from the client and the server developer. All a Web Service developer must do is use the WebMethod attribute on the public methods exposed as Web Services. The MyWebService Web Service provides the MyMessage service, which returns the string "Hello" to the caller. To qualify as a Web Service, use the WebMethod attribute on the exposed Web Services and optionally derive from the WebService base class:

```
using System.Web.Services;

public class MyWebService : WebService
{
      public MyWebService(){}

      [WebMethod]
      public string MyMessage()
      {
            return "Hello";
      }
}
```

On the consumer side, all the developer has to do is point Visual Studio.NET to the remote Web Service and have it generate a wrapper class that looks and feels like a local class, but actually forwards the call to the remote service.

Develop Rich UI Apps

Although .NET's marketing effort concentrates heavily on demonstrating how to build Web applications in .NET, Microsoft isn't abandoning the desktop market. .NET includes a comprehensive framework for developing rich UI client applications for Windows called Windows Form*s*. Windows Forms projects are somewhere between MFC applications and VB6. These projects provide VB6's productivity-oriented environment and ease of development—such as the properties window—but the code layout is much more exposed, as in an MFC application where you see controls binding to handling methods.

Using Windows Forms, you can build almost every kind of Windows application, including ActiveX controls, SDI, (Single Document Interface, like Notepad) and MDI (Multiple Document Interface) applications, and custom controls. But the default is a dialog-based application. If you use VS.NET, you can build Windows Forms using only C# or VB.NET without managed C++. A Windows Forms project lets you drag and drop controls such as buttons and edit boxes to a form layout, and it generates the binding code for you. Interestingly enough, unlike a classic Windows application where you map user events such as a button click to messages handled in your code, a Windows Forms

control handles events the .NET way—you map the event to a .NET delegate bound to a method in your class.

As in MFC, .NET provides many base classes in the System.Windows Forms namespace—including Button, DataGrid, and CheckBox—that you can use as is or derive and extend for your own need. Finally, Windows Forms provide a new development service called Visual Inheritance. You can derive your form from an already existing component in a binary assembly. The user interface layout associated with the base component, such as buttons and controls on the form, is displayed in the visual editor in a special way. You can't change these controls, but it's possible to see where they are and figure out how best to add the controls. This feature is essential in applications involving hundreds of forms.

ASP .NET

Another new .Net application framework is ASP.NET, which you use to develop dynamic HTML pages. ASP.NET a complete overhaul of classic ASP. Classic ASP, a blend of static HTML and script code, executing either on the server or on the browser side, has many limitations and deficiencies. The resulting programming model is messy because it doesn't separate clearly user interface code from business logic, and it has many design limitations. ASP projects often result in unmaintainable spaghetti code that doesn't scale, both in performance and management. Interdev (ASP's development tool) also leaves much to be desired compared with the Visual Studio tools. Interdev has limited debugging capabilities and requires developers to master HTML for even simple rendering.

ASP.NET's goal is to allow you to develop Web applications as easily as you develop desktop applications. You simply drop controls on a form, called a Web Form, and bind the control properties to object members and event handlers. The controls (also called server-side controls) execute on the server and are smart enough to know how to render themselves in HTML and even accommodate different browsers—all without the developer writing a single line of code.

ASP.NET collects user input on the browser side and posts it back to the server, where it's stored as the form class data members and properties. The resulting programming model is manageable, extensible, and object-oriented, much like Windows Forms. ASP .NET developers no longer need to know HTML because the controls do the rendering.

In addition, there is a clear separation of the user interface (the form layout in the visual designer) from the business logic, the class behind the form.

You write the class associated with the Web Form in a .NET language such as C#, and this code compiles at run time to native code, providing a significant performance boost compared with the interpreted classic ASP script.

ASP.NET developers are first-class citizens and can use the rich set of base classes available with the .NET platform. Developers can call and step into other components in other assemblies and take advantage of the VS.NET Integrated Development Environment, including the debugger. In addition to what I've already described, ASP.NET provides many other new features: automatic validation controls that execute

on the client side to validate input and save round trips to the server, and extensive tracing. ASP.NET also offers control values caching, automating data binding of controls, such as a grid, to a data source. You'll find numerous new controls, such as the Calendar, and you can define easily your own custom server controls.

ADO.NET

Another new .NET application framework is ADO.NET, a set of classes you use to access data sources such as databases. ADO.NET is based on a new object model. Unlike classic ADO, it's oriented and optimized for disconnected remote access to the data sources over the Internet. ADO.NET decouples the data consumer from the data source and the platform it resides on by introducing a level of indirection using two classes, DataSet and DataAdapter.

DataSet is the basic data-container object, containing structured information on a set of tables. You associate each DataSet object with a particular subclass of DataAdapter. That subclass is tailored to interact with a particular data source, such as SqlDataAdapter. This design pattern provides the data source indirection because the DataSet interacts only with the generic interface of the DataAdapter base class.

To change a source type, the data consumer simply associates the DataSet with a different DataAdapter. The DataSet caters to doing disconnected work over the Internet because it can cache whole portions of a database and perform the synchronization once changes are committed to the database on the server. While in transit, the data is represented in XML and can pass through firewalls over a regular HTTP port. .NET Web Services methods can return DataSets as parameters, enabling remote data access for Web Services consumers. Another benefit of the DataSet class is you can access tables as a property of the DataSet, so you don't have to know—or at least handcraft—SQL queries.

That said, you can still use ADO.NET for whatever you used classic ADO for—namely, same-machine or intranet data access and manipulation. VS.NET provides visual designer support for setting up a connection with a data source; all you do is drag and drop a data source (a data base or a table) to your project from the Server Explorer, and VS.NET creates the template code for you to access and bind to that data source.

Enterprise Services Key to .NET

.NET relies on COM+ to provide it with component services such as instance management, transactions, activity-based synchronization, granular role-based security, disconnected asynchronous queued components, and loosely coupled events. In fact, Microsoft renamed COM+ in .NET as Enterprise Services, which better reflect its pivotal role in .NET.

A .NET component using COM+ services is called a serviced component, and it must derive from the .NET base class ServicedComponent. You can configure a serviced component to use COM+ services in two ways. The first is similar to how you do it in COM: You derive the component from ServicedComponent, add the component to the Component Services Explorer, and configure it there. The second way is to apply special attributes to the component, configuring it at the source-code level. When you add the component to the Component Services Explorer, it's configured automatically according to the values of those attributes.

.NET allows you to apply the serviced component attributes with great flexibility. When you don't configure a service with attributes, it's configured according to the default settings when you add that component to the Component Services Explorer. You can apply as many attributes as you like, although you can apply some COM+ services only through the Component Services Explorer. These services are mostly deployment-specific configurations, such as persistent subscriptions to COM+ Events and allocation of users to roles.

In general, almost everything you can do with the Component Services Explorer you can do with attributes. The recommended usage pattern for serviced components is to enter as many design-level attributes into the code as possible—such as transaction support and object pooling—and use the Component Services Explorer to configure deployment-specific details. For example, to configure a serviced component to use object pooling, use the ObjectPooling attribute. You can provide optional parameters for pool parameters:

```
[ObjectPooling(MinPoolSize = 3,MaxPoolSize = 10)]
public class MyComponent :ServicedComponent
{…}
```

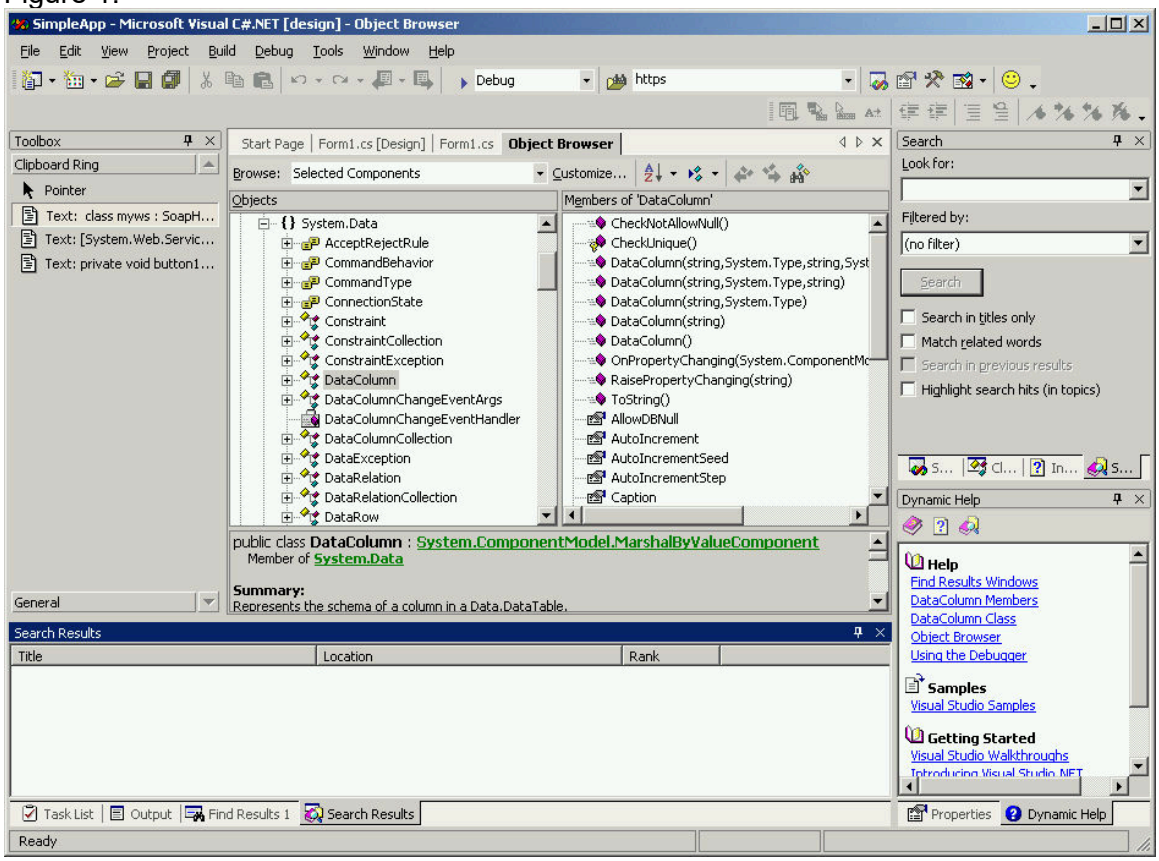
From a configuration management point of view, the .NET integration with COM+ is better than COM under VS6 because .NET allows you to capture your design decisions in your code, instead of storing them separately in the Component Services Explorer.

About the Author:

Juval Lowy is a software architect and the principal of IDesign, a consulting company focused on .NET design and .NET migration. Juval also conducts training classes and conference talks on component-oriented design and development processes. He wrote "COM and .NET Component Services—Mastering COM+" (O'Reilly). Reach him at www.componentware.net.

Captions:

Figure 1.



Get it all Here. The Visual Studio .NET IDE contains everything you need—and then some. Some of the support windows any Visual Studio project can display include the Object Browser, Clipboard ring, integrated Dynamic Help topics and search capabilities. Even though the IDE presents an overwhelming amount of information in one place, it's completely manageable through an intuitive tabbed, retracting window management scheme.

Listing 1:
Provide a Web Service. The MyWebService Web Service provides the MyMessage service, which returns the string “Hello” to the caller. To qualify as a Web Service, use the WebMethod attribute on the exposed Web Services, and optionally derive from the WebService base class.

```
using System.Web.Services;

public class MyWebService : WebService
{
      public MyWebService(){}

      [WebMethod]
      public string MyMessage()
      {
            return "Hello";
      }
}
```

Pullquotes:

The new technology is radically different, not just in syntax but also in the programming model itself.

NET Web Services support can potentially transform e-commence into the next information-technology revolution.

Using Windows Forms, you can build almost every kind of Windows application.