Line 6 creates a new document in the current database.

Line 7 populates the document fields using element data stored in temporary variables.

Line 8 saves the new document.

Line 9 calls the SAXStartElement method when an element opening tag is read.

Line 10 determines the element name and sets the corresponding Boolean variable to True to signal the element is currently being processed.

Welcome to the future

LotusScript programmers are now privy to the world of XML. Working with XML is much more than parsing via SAX, so there are many other features available in Domino 6. I'll cover those issues in future articles. ■

Lotus Domino 5

APPLICATION INTEGRATION

Domino and .NET: Leverage these Complimentary Technologies

By John Duggan

Microsoft .NET is the foundation for the next generation of Microsoft's products. It's also Microsoft's next generation development platform, complete with a rich framework.

Many Domino implementations operate with Microsoft products, so .NET isn't something you can dismiss lightly. We're still in the early days of .NET, but it's likely that .NET will affect future Domino and Notes implementations.

This article includes an overview of .NET, how .NET components can access Domino, and how Domino can access .NET components.

Subscriber Download

- The source code used in this article.
- http://Advisor.com/Article/DUGGJ07

. NET overview

In a nutshell, you can think of .NET as being divided into three parts:

.NET products—Applications based on the .NET platform; e.g., Visual Studio .NET and the .NET servers.

.NET services—XML Web services

.NET Framework—Programming model of the .NETplatform for building nad implementing XML Web services and applications.

This article discusses how you can integrate Domino with .NET, thus I only refer to the .NET framework.

.NET Framework

The .NET Framework is the infrastructure for the overall .NET platform. It consists of four parts (figure 1).

The first part is a runtime environment, called Common Language Runtime (CLR). It handles memory management, error trapping, processes/threads, and security. This environment only runs applications that have been compiled to Intermediate Language (IL) as shown in figure 2. This is similar to the Java virtual machine (JVM), but the JVM is interpreted and can only be used by applications written in Java. (Although you can use other programming languages to generate Java bytecode.)

The second part of the .NET Framework is a set of unified framework classes. .NET removes the need for multiple classes, because software developers can share the same framework. For example, previously Visual C developers used MFC/ATL, Java developers used Swing/AWT, Visual Basic developers use VB libraries, and so on.

The third part of the framework is Windows forms, which involves developing standard Win32 applications.

Finally, there's ASP.NET, which involves developing Web applications and Web services.

.NET is language-independent, and provides a tight integration with a wide range of languages, such as Managed C++, JavaScript, Eiffel, Component Pascal, APL, COBOL, Oberon, Perl, Python, Scheme, Smalltalk, Standard ML, Haskell, Mercury, etc.

.NET has been in development for four years, and to provide a comprehensive overview of this large framework is beyond the scope of this article. To narrow the focus, I'll concentrate on the CLR and framework classes. You can find in-depth information about .NET at http://www.gotdotnet.com/.

Where to get the .NET framework?

Currently, .NET is an add-on for Windows. The Windows .NET server will be the first OS to include the framework. The framework will most likely be part of future service packs.

The .NET Framework runtime (or redistributable) is freely available from http://www.microsoft.com/net/. It's approximately 20MB and works on Windows 98, NT4, ME, 2000, and XP. As an aside, for Linux developers, the platform is being converted to Linux as part of an open source project called Mono, located at http://www.go-mono.com/. There is also a .NET compact framework for Windows CE.

John Duggan is an independent consultant and has been involved with IT development/management technologies since 1990. john_duggan@hotmail.com.

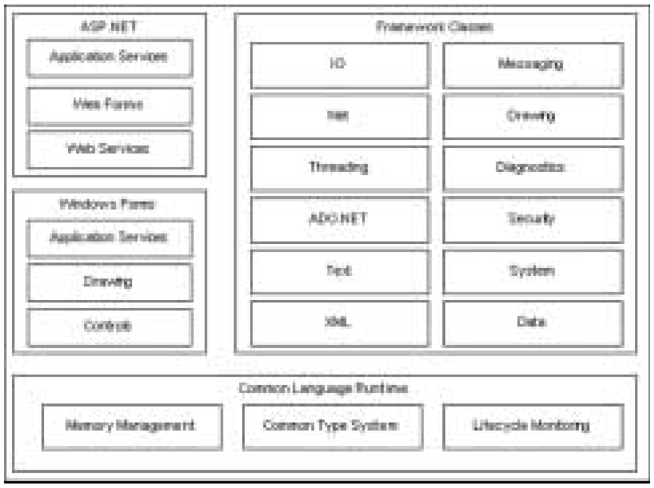


Figure 1: **.NET Framework**—The framework is the core of the .NET initiative.

The Framework SDK is a free development environment you can use to develop .NET applications. It provides a Visual Basic, Visual C#, JScript, and a C++ compiler. It also includes a graphical debugger and a collection of other useful tools. It doesn't provide a graphical IDE, such as Visual Studio .NET.

For this article, I use the Framework SDK available at http://www.microsoft.com/net/. Currently, it only works on NT4, Windows 2000, and XP. The .NET service pack 1 is available at http://msdn.microsoft.com/netframework/downloads/sp1/default.asp.

Writing a .NET component to access Domino

I'll show you how to write a simple .NET program that shows the titles of all the documents in a specified view in a Notes database. First, you must install the framework. All the source code used in this article is available for download from http://Advisor.com/Article/DUGGJ07.

Let's look at the source code first, then discuss the steps for running the application. I've written the code in Visual Basic, so it's similar to LotusScript:

```
Imports System
Imports domobj

Namespace Notesvb

Module Notes
 Sub Main()
  Dim s As New NotesSession
  Dim db As NotesDatabase
  Dim v As NotesView
  Dim vn As NotesViewNavigator
  Dim e As NotesViewEntry
  Dim doc As NotesDocument
  Dim subj as NotesItem
  Dim msg as String

  Try
   ' Initialize Notes
   Call s.Initialize
   ' Open the dotnet.nsf database
   db = s.GetDatabase("", "dotnet.nsf")
   ' open the view that contains all the documents
   v = db.GetView("All")
```

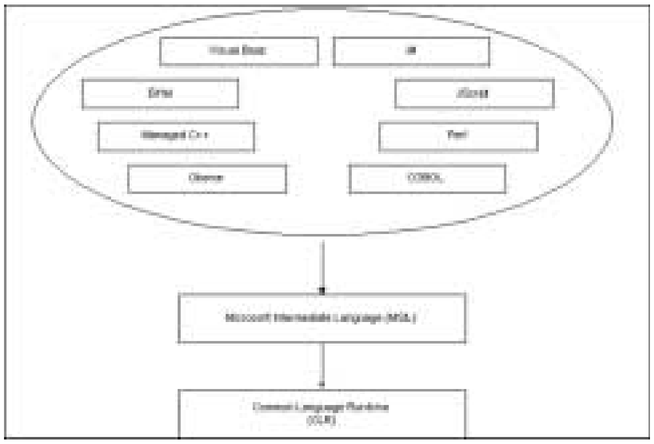


Figure 2: **Intermediate Language**—You can use one of many languages to write for .NET.

```
   vn = v.CreateViewNav()

   e = vn.GetFirstDocument()

   ' repeat for each document within the view
   ' show the contents of the subject field.
   While Not (e Is Nothing)
    doc = e.Document
    subj = doc.GetFirstItem("Title")
    Console.WriteLine("Subject {0} ",subj.text)
    e = vn.GetNextDocument(e)
   End While
  Catch ex as Exception
    Console.WriteLine(ex.toString)
   Finally
     Console.WriteLine("done!")
    End Try
  End Sub
 End Module
End Namespace
```

The important line is the "Import domobj" statement, which lets you use the Domino objects. There are no native .NET classes for Domino, so where does it come from?

Lotus Notes 5.0.2b and higher provides support for COM objects. Fortunately, Microsoft has provided a means for letting you access COM objects within the .NET environment.

You can use the utility Type Library Importer (tblimp.exe) included with the .NET Framework to create a .NET proxy component for a COM component. The Notes type library for its COM component is located in the domobj.tlb file in the Notes program directory. You can use this command to create a .NET proxy component:

```
tlbimp domobj.tlb /out:domobj.dll
```

The generated .DLL must reside within your application's directory. In other words, the program you are about to write requires the domobj.dll to be located in the same directory as your program. Ideally, you don't want to have lots of copies of the same .DLL on the hard disk. It's preferable to share the .DLL among all .NET applications.

The solution is to use the Global Assembly Cache (GAC), which is a shared repository of assemblies. Each assembly has a unique name identifying itself. This is to ensure the application uses the correct .DLL, even though there may be multiple copies of the same .DLL.

First, generate a unique name using the Shared Name tool (sn.exe):

```
sn -k domino.key
```

Convert the type library using the generated name:

```
tlbimp domobj.tlb /out:domobj.dll /keyfile:domobj.key
```

Finally, install it in the global assembly using the assembly registration tool (gacutil.exe):

```
gacutil /i domobj.dll
```

If you want to uninstall it, issue this command:

```
gacutil /u domobj.dll
```

You can check whether it's installed correctly by going to the Windows\Assembly directory under Windows Explorer (figure 3). Or, go to Control Panel > Administrative Tools > Microsoft .NET Framework Configuration to check.

You can now compile your Visual Basic program by typing:

```
vbc /r:c:\lotus\notes\domobj.dll notes.vb
```

The path to the .DLL is only required for compilation. When the program executes, it uses the GAC to locate the .DLL.

The result of the compilation is an executable, where you can type in the name of the program:

```
notes.exe
```

Windows detects whether it's a .NET file or a standard Win32 and runs the code under the appropriate environment.

Using C#

One of the strengths of .NET is the ability to write in any language, and C# is a new language from Microsoft similar to Java. Remembering that .NET has a single set of classes, C# developers can call the same set of Notes classes as these:

```
using System;
using domobj;

class Notes
{
  public static int Main()
  {
    try
    {
      NotesSession session = new NotesSession();
      session.Initialize("");
      Console.WriteLine("Using the ID: _
         {0}",session.CommonUserName);

      NotesDatabase db = _
         session.GetDatabase("","dotnet.nsf",false);
      NotesView view = db.GetView("All");

      NotesViewNavigator vn = view.CreateViewNav(0);
      NotesViewEntry entry = vn.GetFirstDocument();

      while (entry!=null)
      {
       NotesDocument doc = entry.Document;
       NotesItem subj = doc.GetFirstItem("Title");
       Console.WriteLine(" Subject {0} ",subj.Text);
       entry = vn.GetNextDocument(entry);
      }
     }
     catch (Exception ex)
     {
       Console.WriteLine(ex.ToString());
     }
     finally
     {
```

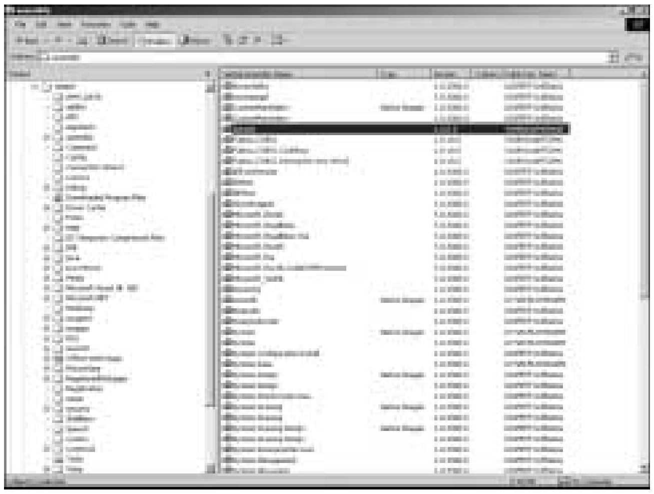


Figure 3: **Is it installed correctly?**—List of assemblies within the Global Cache.

```
       Console.WriteLine("Done!");
    }

    return 0;
 }
}
```

You can compile it using the C# compiler and run it from the command line in exactly the same way you compiled the Visual Basic sample:

```
csc /r:domobj.dll notes.cs
Notes.exe
```

Other languages as well!

Hopefully, you can see that you can use the same set of classes for all languages, which is a powerful feature of .NET. You can even use the Notes classes in JScript, like this:

```
import System;
import domobj;

Console.WriteLine("Hello World.NET");
var session = new NotesSession();
session.Initialize("");

Console.WriteLine("You are using the ID:
{0}",session.CommonUserName);
```

The program returns the name of the current Notes' ID. Compile and run the same as the previous samples:

```
Jsc /r:c:\lotus\notes\domobj.dll notes.js
Notes.exe
```

You can call Domino classes from Managed C++, like this:

```
#using <mscorlib.dll>
#using <domobj.dll>

void main()
{
  domobj::NotesSessionClass *s = new _
     domobj::NotesSessionClass();
  s->Initialize("");
  System::Console::WriteLine(L"The Notes ID you are
using is");
  System::Console::WriteLine(s->get_CommonUserName());
}
```

Managed C++ is different than C++ because it's "managed" by the CLR. Code that doesn't operate under the CLR is known as "unmanaged."

Perhaps most interesting language is J#. It's Microsoft's equivalent to Java, but only compiles to IL. There are some other differences, such as not providing support for running applets, JNI, RMI, sun.io.*, or sun.net.* packages. Microsoft makes it clear that it's different by providing a different file extension in ".jsl". It's convenient for Java developers to program on .NET.

Your J# code for retrieving the current Notes ID would look like this:

```
import domobj.NotesSessionClass;

class Notes
{
  public static void main (String args[])
  {
    domobj.NotesSessionClass s = new _
       domobj.NotesSessionClass();
    s.Initialize("");
    System.out.println("The Notes ID you are using is");
    System.out.println(s.get_CommonUserName());
  }
}
```

If you only have access to the compiled Java byte code, you can convert it to IL using a utility located in the J# toolkit, called J# .NET binary converter (jbimp.exe).

Mixing languages

The real power of .NET is combining objects in multiple languages. It doesn't matter what your skills are, there is a language that will let you access the Domino classes.

As an example, let's create a simple C# program that will call a VB and a COBOL object.

The C# program is straightforward:

```
using System;

public class Notes
{
 public static void Main()
 {
  // Initialize and call the VB object
  VBid vb  = new VBid();
  vb.Show_NotesID();

  // Initialize and call the COBOL object
  COBOLid cobol = new COBOLid();
  cobol.Show_NotesID();

  Console.WriteLine("(C#) Done.");
 }
}
```

The VBid object retrieves the current Notes' ID and prints it to the console:

```
Imports System
Imports domobj

Public Class VBid
  Dim s As New NotesSession
  Public username as String

  Public Sub Show_NotesID()
    ShowMsg()

    Call s.Initialize
    Console.WriteLine("{0}",s.CommonUserName)
  End Sub

  Public Overridable Sub ShowMsg
    Console.WriteLine("(VB) The Notes ID you are using
is:")
  End Sub
End Class
```

The COBOLid writes a message to the console by overriding the ShowMsg function in the VBid object:

```
000010 CLASS-ID. COBOLID AS "COBOLid" INHERITS VBid.
000020 ENVIRONMENT DIVISION.
000030 CONFIGURATION SECTION.
000040 REPOSITORY.
000050  CLASS VBID AS "VBid".
000060 OBJECT.
000070 PROCEDURE DIVISION.
000080 METHOD-ID. SHOWMSG AS "ShowMsg" OVERRIDE.
000090 DATA DIVISION.
000100 PROCEDURE DIVISION.
000110  DISPLAY "(COBOL) The Notes ID you are using is:"
000120 END METHOD SHOWMSG.
000130 END OBJECT.
000140 END CLASS COBOLID.
```

If you compile the program, you'll get the output shown in figure 4.

What does this tell us? It means that it doesn't matter what skill set you have, you can access Notes in any language! (I suspect the Notes developers didn't ever expect to see COBOL programmers being able to access Notes functionality!)

Access .NET components from Notes.

You've seen how .NET components can access Notes, but can Notes access .NET components? The answer is yes—and it's straightforward! Here's a simple C# program that prints a message to the console:

```
using System;

public class Notes
{
 public string ShowMsg(string msg)
 {
  return msg + " (C#)";
 }
}
```

Compile the program to IL:

```
csc /t:library Notes.cs
```

Notes and other Windows applications don't recognize .NET components, so you have to create a type library to let COM components access it, using the type library exporter utility, like this:

```
tlbexp Notes.dll /out:notes.tlb
```

Register the type library in the Windows registry, like this:

```
regasm Notes.dll
```

Copy the .DLL and the type library into the Notes program directory or the system path.

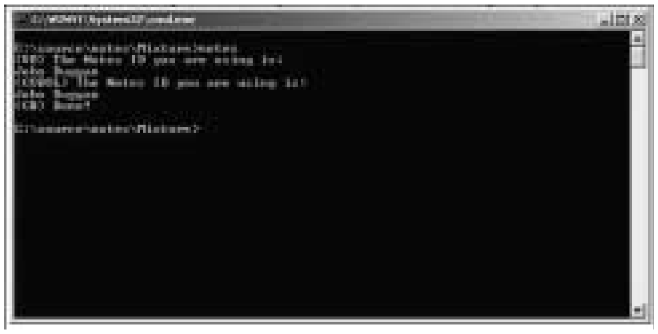
You can now write LotusScript code that calls the .NET component by using this code:

```
Set object = CreateObject("Notes")
Msgbox object.ShowMsg("Hello")
```

To unregister the type library from the registry, type:

```
regasm notes.dll /unregister".
```

It's possible to access the Lotus Notes C API directly from the .NET components, but there are performance implications and it isn't recommended unless it's your only option. I



```
C:\source\notes\Mixture\notes
(VB): The Notes ID you are using is:
[illegible]
(COBOL): The Notes ID you are using is!
[illegible]
(C#) Done!

C:\source\notes\Mixture>
```

Figure 4: **Let's talk**—The result of accessing Notes via a mixture of languages!

suspect for most Notes developers this won't be required, so I won't go into further detail accessing the Lotus Notes API.

Debugging

If you do get a problem with your code, a graphical debugger is included with the .NET Framework. To activate the debugging, put in the "/debug" parameter when compiling the program and load the graphical debugger located in the Microsoft.NET/GuiDebug folder:

```
Vbc /r:c:\lotus\notes\domobj.dll notesvb.vb /debug
C:\program files\Microsoft.net\guidebug\Dbgclr
```

Wrap up

A lot of people are confused about the benefits of .NET. It's still in its early days, and it's still evolving. Currently, the .NET Framework is essentially an upgrade to previous forms of Windows development. Any developer who had problems writing message routines, registering .DLLs, and memory management will automatically appreciate the benefits of .NET. In return, this will result in software that's easier to maintain and support.

This article has shown you that Domino and Notes happily co-operate with the .NET environment. There are many areas I haven't touched upon regarding .NET, but those are left for another article! ■

Lotus Domino ??versions??

Lotus Notes ??versions??

PERFORMANCE

Expand and Collapse Single Category Views without a Server Call

By Stephen Linn

Subscriber Download

- Sample database.
- http://Advisor.com/Article/LINNS01

Most of you are quite familiar with the way Notes Views appear in a browser: Categorized views appear with a blue "twistie" you can expand and collapse. Each time you expand or collapse a category, a call is made to the Web server, and the view is served back to the browser with the appropriate section expanded or collapsed. In many cases, there is a good reason why expanding and collapsing a category in a view requires a Web server call. If you have a contract system that lists thousands of contracts, categorized by date, loading the entire view up front so you can expand and collapse it without a server call may be practical due to lenghty load times.

There are many times, however, when categorized views contain significantly less data or are served over a high-speed LAN. Forcing Web users to make a call to the server every time they want to expand or collapse a category in a view is time-consuming and wasteful, especially when an alternative exists. Figure 1 shows a simple categorized view.

Each time you select a blue twistie, a call is made to the server to reload the page with the selected category expanded or collapsed. Furthermore, you're limited to two choices for expansion: Expand All or Expand One. The default behavior of a Domino view, when displayed on the Web, doesn't let you expand 08/23/2001 and 08/24/2001, and leave 08/27/2001 collapsed. Notice also that the Bushels column is set to total.

Figure 2 shows the same data filtered through the ProcessSCView agent. This view displays the same information as figure one. The view has one category column, Date, that you can expand or collapse without a call to the Web server.

Notice I've expanded two of the dates and left the third one collapsed. All expand and collapse requests are processed on the client side. In this article, I'll show you how to expand and collapse single category views without a server call.

How it works

There are four basic elements that let you expand and collapse a category without a server call: Lotus Script agent, HTML Components (HTC) file1, Notes process view, and Notes Links view.

NOTE: If you aren't familiar with Microsoft Internet Explorer behaviors, HTC may not be a familiar term for you. I'll explain the HTC file in detail later in this article. Also,

???Stephen, you included a sample database. We need to refer to it somewhere in the text where it makes sense for the readers to use it. Any suggestions on the most appropriate place???em

???Hi Stephen, I need a short bio here for you. You're more than welcome to include your e-mail address if you'd like???em