Using IBM Lotus Domino in a Microsoft .NET application

Tony Patton

September 05, 2006

IBM Lotus Domino and Microsoft toolsets are built on standards that allow interoperation of these two systems. This article covers techniques and approaches you can use to integrate Lotus Domino and Microsoft .NET, including Component Object Model (COM) and Lotus Domino Objects, Web services, and XML.

As every developer knows, having disparate systems live in harmony is a common challenge. Regardless of their size, many IT departments don't standardize on one development platform. Company mergers and acquisitions often introduce different elements, leading to disparate systems as older applications coexist with new platforms. Products such as IBM Lotus Domino provide a powerful platform for developing enterprise applications, but like many enterprise software products, it needs to interoperate with other systems.

This article examines Lotus Domino integration with the Microsoft .NET development platform; in particular, Lotus Domino 7, the Microsoft .NET Framework V1.1, and Microsoft Visual Studio .NET 2003 are discussed. This article assumes that you are an experienced Notes/Domino application developer familiar with LotusScript and the Domino Web services design element and that you have knowledge of the Microsoft .NET Framework.

**NOTE:** The solution described in this article works with both Microsoft .NET Framework V2.0 and V1.1.

The programming model: .NET and COM

Lotus Domino is an excellent platform for working with enterprise data. You can easily access non-Domino data sources through simple Open Database Connectivity (ODBC) or use the powerful functionality that IBM Lotus Enterprise Integrator and Domino Enterprise Connection Services provide. Sometimes, however, you may need to access Lotus Domino from other applications. Thankfully, you can expose the Domino environment to other applications with the Component Object Model (COM), which allows external applications to work easily with Lotus Domino through various objects and classes exposed through COM. A good example is integrating Lotus Domino functionality in an application built with the .NET Framework.

COM is an older technology, but .NET provides callable wrappers to allow .NET and COM to interoperate. With a run-time callable wrapper, you can use a COM component within .NET. It
sounds complicated, but it's simplified with a development tool such as Microsoft Visual Studio. Although you must create wrappers for the COM components to use them in a .NET application through interop, Visual Studio .NET creates them automatically when you perform these steps:

1. In Visual Studio, choose Project - Add Reference.
2. Select the COM tab of the Add Reference dialog box, and then double-click the appropriate Type Library file listed.
3. Click OK to add the reference.

At this point, Visual Studio .NET converts the objects and members of the COM Type Library file into equivalent .NET assemblies. When the .NET assemblies are generated, you can use their classes to create objects and to call members as though the COM objects and members were native .NET classes and members. Also, you can reverse the process to use a .NET assembly in a COM-based application.

NOTE: If you are not using Visual Studio .NET, the .NET Framework software development kit (SDK) provides the Type Library Importer (tlbimp.exe) utility, a command-line tool that converts the classes and interfaces contained in a COM Type Library to metadata. This tool creates an interop assembly and namespace for the type information automatically.

A good way to illustrate this approach is by making Lotus Domino objects available in a Visual Studio solution. Figure 1 shows the Add Reference dialog box with the Lotus Domino Objects highlighted.

Figure 1. Add Reference dialog box
Lotus Domino Objects

Beginning with Lotus Notes and Domino 5.02b, Domino objects are accessible through COM by using Lotus Domino Objects, which is included with installation of either a Domino server or one of the Notes clients (IBM Lotus Domino Designer, Lotus Domino Administrator, or Lotus Notes). Look for domobj.tlb in the base installation directory. Lotus Domino 6.x includes version 1.1 of the Lotus Domino Objects; Lotus Domino 7 includes version 1.2.

Although Lotus Domino must be installed, it doesn't need to be running. Another important point is that only backend classes are available when accessing Lotus Domino through COM, so no user interface (UI) classes are accessible.

The Domino mail application

Accessing Lotus Domino resources through COM is similar to using the Java or LotusScript programming language within Lotus Domino Designer; any and all interaction with Lotus Domino begins with a session. Lotus Domino Objects include all necessary classes for working with Lotus Domino through COM, and the COM objects will be familiar to any developer exposed to the Domino object model through the LotusScript or Java language.

Included in the Visual Studio integrated development environment (IDE) is the IntelliSense feature, which helps you work with the COM objects by providing context-sensitive help and automatically completing code as you type, thereby reducing spelling mistakes.

The first example builds a .NET Windows application that retrieves data from a Domino Directory and uses Lotus Domino to send email messages (see figure 2). The .NET Windows application uses the following .NET elements:

- A ComboBox control to select the mail recipient and to populate a field with values from a Domino Directory view
- A TextBox control to enter the subject of the mail message
- A RichTextBox control for the body of the mail message
- A Label control to signal whether the message was sent or an error was encountered
- A Button control to send or attempt to send the actual mail message
Figure 2. Sending email from Lotus Domino with a .NET Windows application

Listing 1 shows the bulk of the application code (the Visual Studio-generated code for the application is omitted). The btnSend_Click method is called when you click the Send button. This method creates NotesSession, NotesDatabase, NotesView, and NotesDocument objects that in turn are used to create and send the message.

The Form_Load method is called when the application is loaded. It uses NotesSession, NotesDatabase, NotesView, and NotesDocument objects to populate the ComboBox. Individual values from the documents in the view are added to the ComboBox control. The domain name is appended to selected person names to aid message delivery.

Listing 1. The .NET Windows application

```vbnet
Private Sub btnSend_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnSend.Click
Try
    Dim ns As New NotesSession
    Dim db As NotesDatabase
    Dim doc As NotesDocument
    If Not (ns Is Nothing) Then
        ns.Initialize("password")
        db = ns.GetDatabase("ServerName//Domain", "names.nsf", False)
        If Not (db Is Nothing) Then
            doc = db.CreateDocument()
            doc.ReplaceItemValue("Form", "Memo")
            doc.ReplaceItemValue("SendTo", cboNames.SelectedItem + "/Domain")
            doc.ReplaceItemValue("Subject", txtSubject.Text)
            Dim rt As NotesRichTextItem
            rt = doc.CreateRichTextItem("Body")
            rt.AppendText(rtbBody.Text)
            doc.Send(False)
            rt = Nothing
            doc = Nothing
            lblMessage.Text = "Message Sent."
        End If
    db = Nothing
    ns = Nothing
End If
```

To send the mail message, the application does the following:

1. Instantiates the NotesSession object.
2. Instantiates the NotesDatabase object with the specified server database.
3. Creates a new NotesDocument object through the NotesDatabase object.
4. Populates fields in the document with values from the form.
5. Creates a new NotesRichTextItem object through the NotesDocument object for the body of the mail message.
6. Adds text from the form to the NotesRichTextItem.
7. Sends the email message through the NotesDocument Send method.
8. Sets the Label control to be displayed accordingly if the message is sent. (Otherwise, an error message is displayed if an exception is encountered. The whole operation is enclosed in a try block.)

Using COM isn't limited to Windows applications. You can use it in Web or other application types as well.

Next, let's focus on communicating with Lotus Domino using other avenues, such as Web services and XML.

**Domino Web services**

While Lotus Domino Objects provide an excellent way of accessing Domino resources within non-Domino applications, it's not the only approach available. Another way to integrate the two systems involves Web standards and consuming a Domino Web service within .NET code.

A Web service design element was added to Lotus Domino 7. Domino-based Web services are consumed like any standard Web service. You use Web Services Description Language (WSDL) to discover the available methods in the Web service. The WSDL for Domino-based Web services
is available through the Web services URL appended with the ?wsdl command. For example, the following URL retrieves the WSDL for a Web service named testws:

http://localhost/dominodatabase.nsf/testws?wsdl

The Visual Studio .NET IDE makes it easy to use a Web service in a project. To do so, add a Web Reference to the project by right-clicking the project, and then selecting Add Web Reference as shown in figure 3.

**Figure 3. Add a Web Reference to a project**

![Add Web Reference to a project](image)

### Domino XML support

Another Domino feature that you can use for integration is XML. The XML for common Domino elements is easily accessed. A good example is working with a Domino view. The view’s XML is easily accessed with its URL and the ?ReadViewEntries command. For example, using the following URL, you can access the XML for the _People view in Domino Directory 7:

http://domain/names.nsf/_People?ReadViewEntries

Listing 2 provides a sample of the XML generated using the ?ReadViewEntries command for the directory on a Lotus Domino 7 server.

**Listing 2. XML generated from using the ?ReadViewEntries command**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<viewentries toplevelentries="3">
  <viewentry position="1" unid="A819181482AC862F8525719997D02BC" noteid="1342" siblings="3">
    <!-- XML content for _People view -->
  </viewentry>
</viewentries>
```
You can combine this generated XML with .NET's XML support to exchange or use Domino data. For example, you can tie the generated XML to a Microsoft ASP.NET data control. The ASP.NET page shown in Listing 3 uses the XML from the _People view to populate a DropDownList control.

**Listing 3. ASP.NET page using the XML from the _People view**

```xml
<entrydata columnnumber="0" name="$18">
  <number>0</number>
</entrydata>
<entrydata columnnumber="1" name="$17">
  <text>Patton, Tony</text>
</entrydata>
<entrydata columnnumber="2" name="$12">
  <text/>
</entrydata>
<entrydata columnnumber="3" name="CompanyName">
  <text/>
</entrydata>
<entrydata columnnumber="4" name="$16">
  <text>Tony Patton/Patton@Test</text>
</entrydata>
<entrydata columnnumber="5" name="$21">
  <text>TestServer/Test</text>
</entrydata>
</viewentry>
</viewentries>

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
  If Not (IsPostBack) Then
    Try
      GetData()
      Catch ex As Exception
        Response.Write(ex.ToString())
    End Try
  End If
End Sub

Public Sub GetData()
  Dim strResult As String
  Dim objResponse As WebResponse
  Dim objRequest As WebRequest
  objRequest = HttpWebRequest.Create("http://192.168.1.103/names.nsf/_People?readViewEntries")
  objResponse = objRequest.GetResponse()
  Dim sr As StreamReader
  sr = New StreamReader(objResponse.GetResponseStream())
  strResult = sr.ReadToEnd()
  Dim doc As New XmlDocument
  sr.Close()
  doc.LoadXml(strResult)
  Dim nodes As XmlNodeList
  nodes = doc.SelectNodes("viewentries/viewentry/entrydata[@columnnumber="1"]")
  For Each node As XmlNode In nodes
    ' Process each node here
  Next
End Sub
```
This code is a standard ASP.NET page (with no codebehind) that uses Microsoft Visual Basic .NET. Essentially, the code does the following:

2. Instantiates an XmlNodeList object from the XmlDocument and an XPath expression to select certain entities from the XML. (This code is interested only in the entrydata elements in the first column, that is, the columnnumber=1 attribute.)
3. Uses the XmlNodeList as the data source for the DropDownList control.
4. Calls the databind method of the DropDownList control to bind the data to the control.

The results are visible on the Web page, and you can select a name from the list.

The first step uses the .NET System.Net and System.IO namespaces to work with network (Internet) requests and the data returned. The page uses a WebRequest object to call the appropriate URL, while a WebResponse object captures the response.

You combine the WebResponse object's ResponseStream property with a StreamReader object to read the XML that the Web request returns. In turn, you use the XML to populate and to instantiate an XmlDocument object. Individual nodes are selected from the XMLDocument object and stored in the XMLNodeList object. Using an XPath expression, you can select the appropriate nodes.

You use the collection of selected nodes to populate the data control with individual columns (innertext) designated as the displayed value in the control as well as the value returned when a user selects an individual entry. All operations related to retrieving the XML and populating the data control reside in the GetData method, which is called in the ASP.NET Web form's Page_Load event. You want to populate the control only when the page is initially requested, so you use the page's IsPostBack property to ensure that it's loaded only once, not when the page is refreshed or reloaded. Finally, the page uses a try block to catch any unforeseen errors with the error description displayed on the page. Figure 4 shows the ASP.NET page.
Figure 4. ASP.NET DropDownList control populated from the Domino view’s XML

For purposes of demonstration, the Domino mail application example was combined with a Domino Web service to retrieve and display Domino data through Web service calls. Before covering Web service use within a .NET application, though, you must first cover the Domino Web service.

Lotus Domino Web service to retrieve Domino data

New to Lotus Domino Web services?

If you’re new to Domino Web services, refer to the developerWorks Lotus article, "Lotus Notes/Domino 7 Web Services." It provides a good background that includes the following basics of coding Domino Web services with LotusScript:

- A class definition for the implementation code. (This class must become the PortType class named on the Basics tab of the properties window, and it must be public.)
- Within the class, a procedure (function, sub-procedure, or method) definition for each operation in the Web service. (These procedures must be public.)
- Inclusion of lsxsd.lss.
- Initialization of a NotesSession if Lotus Domino Objects are accessed. (This is best done in a new block.)

The Web service is simple. It accepts a user’s name (formatted as last name followed by first name and separated by a comma -- the format used in the previous example) and retrieves associated values for that user from the Domino Directory. The service is written in LotusScript.
The sample service resides in a class named GetUserInfo that includes the following public methods:

- **GetPhoneNumber**: Returns the office phone number for the user passed to it
- **GetAddress**: Returns the office address for the user passed to it
- **GetCity**: Returns the office city name for the user passed to it
- **GetState**: Returns the office state name for the user passed to it
- **GetZip**: Returns the office zip code for the user passed to it
- **GetCountry**: Returns the office country name for the user passed to it

All methods accept one string parameter (the name of user) and return a string value. The NotesSession, NotesDatabase, and NotesView objects are defined with their initialization occurring in the class' new block. Listing 4 shows the Web service code.

### Listing 4. Domino Web service

```vbscript
%INCLUDE "lsxsd.lss"
Dim session As NotesSession
Dim db As NotesDatabase
Dim vw As NotesView
Dim doc As NotesDocument
Public Class GetUserInfo
    Sub New
        Set s = New NotesSession
        Set db = s.CurrentDatabase
        Set vw = db.GetView("(PeopleLookup)"
    End Sub
    Public Function GetPhoneNumber(uName As String) As String
        Set doc = vw.GetDocumentByKey(uName)
        If Not (doc Is Nothing) Then
            GetPhoneNumber = doc.GetItemValue("OfficePhoneNumber")
        Else
            GetPhoneNumber = 
        End If
    End Function
    Public Function GetCity(uName As String) As String
        Set doc = vw.GetDocumentByKey(uName)
        If Not (doc Is Nothing) Then
            GetCity = doc.GetItemValue("OfficeCity")
        Else
            GetCity = 
        End If
    End Function
    Public Function GetState(uName As String) As String
        Set doc = vw.GetDocumentByKey(uName)
        If Not (doc Is Nothing) Then
            GetState = doc.GetItemValue("OfficeState")
        Else
            GetState = 
        End If
    End Function
    Public Function GetAddress(uName As String) As String
        Set doc = vw.GetDocumentByKey(uName)
        If Not (doc Is Nothing) Then
            GetAddress = doc.GetItemValue("OfficeStreetAddress")
        Else
            GetAddress = 
        End If
    End Function
    Public Function GetZip(uName As String) As String
        Set doc = vw.GetDocumentByKey(uName)
```
If Not (doc Is Nothing) Then
    GetZip = doc.GetItemValue("OfficeZip")(0)
Else
    GetZip = ""
End If
End Function

Public Function GetCountry(uName As String) As String
Set doc = vw.GetDocumentByKey(uName)
If Not (doc Is Nothing) Then
    GetCountry = doc.GetItemValue("OfficeCountry")(0)
Else
    GetCountry = ""
End If
End Function

Public Function GetFullName(uName As String) As String
Set doc = vw.GetDocumentByKey(uName)
If Not (doc Is Nothing) Then
    GetFullName = doc.GetItemValue("FirstName")(0) + " " + doc.GetItemValue("LastName")(0)
Else
    GetFullName = ""
End If
End Function
End Class

You can easily view the WSDL for this Web service by typing its URL into your browser: [http://dominoserver/names.nsf/GerUserInfo?wsdl](http://dominoserver/names.nsf/GerUserInfo?wsdl).

**NOTE:** A new view was added for use in each method's look-ups. This hidden view, called PeopleLookup, shows only documents created with the Person form with one column populated with the following formula:

```
@Trim(@Subset(LastName;1))+@If(Firstname !="";" ,
" +@Trim(@Subset(FirstName;1));" )+@If(MiddleInitial !="";"
" +@Trim(@Subset(MiddleInitial;1));" )
```

### The ASP.NET application

You can use this Web service in an ASP.NET application to populate fields on a Web page with data values retrieved from Web service calls. To begin, add a reference to the Web service in the .NET application. Figure 5 shows the added Web reference as well as the methods listed after a user enters the service address and clicks Go.
You type the name used to access the Web service in code in the Web reference name field in the Add Web Reference dialog box. After the Web reference has been added, the code is straightforward. Listing 5 shows the code with the ASP.NET page first followed by its codebehind file.

**Listing 5. ASP.NET Web form using the Domino Web service**

```html
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="DominoTest.aspx.vb" Inherits="DominoIntegration.Test2"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Domino Web Service Demo</title>
</head>
<body>
<form id="frmDominoIntegration" method="post" runat="server">
  <asp:Label id="lblSelect" runat="server" Width="144px">Select Name:</asp:Label>
  <asp:DropDownList id="ddlNames" runat="server"
    OnSelectedIndexChanged="ddlNames_SelectedIndexChanged"
    AutoPostBack="True" Width="336px"></asp:DropDownList><br>
  <asp:Label id="lblName" runat="server" Width="144px">Address:</asp:Label>
  <asp:TextBox id="txtFullName" runat="server" Width="360px" /><br>
  <asp:Label id="lblAddress" runat="server" width="144px">Address:</asp:Label>
  <asp:TextBox id="txtAddress" runat="server" Width="360px" /><br>
  <asp:Label id="lblCityState" runat="server" Width="144px">City, State</asp:Label>
  <asp:TextBox id="txtCity" runat="server" Width="264px" /><br>
    ,
  <asp:TextBox id="txtState" runat="server" Width="80px" /><br>
  <asp:Label id="lblZip" runat="server" Width="144px">Zip Code:</asp:Label>
```
Listing 6 shows the codebehind file for the ASP.NET Web form written in Visual Basic.

**Listing 6. ASP.NET Web form's Visual Basic. NET code**

```vbnet
Imports System.Net
Imports System.IO
Imports System.Xml

Public Class DominoIntegration
    Inherits System.Web.UI.Page

    Protected WithEvents ddlNames As DropDownList
    Protected WithEvents lblSelect As Label
    Protected WithEvents txtFullName As TextBox
    Protected WithEvents lblName As Label
    Protected WithEvents lblPhone As Label
    Protected WithEvents lblCountry As Label
    Protected WithEvents txtCountry As TextBox
    Protected WithEvents lblZip As Label
    Protected WithEvents lblCityState As Label
    Protected WithEvents txtZip As TextBox
    Protected WithEvents txtState As TextBox
    Protected WithEvents txtCity As TextBox
    Protected WithEvents txtPhoneNumber As TextBox
    Protected WithEvents txtAddress As TextBox
    Protected WithEvents lblAddress As Label
    Protected WithEvents lblComma As Label

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Init
        InitializeComponent()
    End Sub

    Public Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        If Not (Me.IsPostBack) Then
            Try
                GetData()
            Catch ex As Exception
                Response.Write(ex.ToString())
            End Try
        End If
    End Sub

    Public Sub GetData()
        Dim strResult As String
        Dim objResponse As WebResponse
        Dim objRequest As WebRequest
        objResponse = objRequest.GetResponse()
        Dim sr As System.IO.StreamReader
        sr = New StreamReader(objResponse.GetResponseStream())
        strResult = sr.ReadToEnd()
        Dim doc As New XmlDocument
        sr.Close()
        doc.LoadXml(strResult)
        Dim nodes As XmlNodeList
        nodes = doc.SelectNodes("viewentries/viewentry/entrydata").
```

<asp:TextBox id="txtZip" runat="server" width="200px" /><br>
<asp:Label id="lblCountry" runat="server" width="144px">
    Country:</asp:Label>
<asp:TextBox id="txtCountry" runat="server" width="200px" /><br>
<asp:Label id="lblPhone" runat="server" width="144px">
    Phone:</asp:Label>
<asp:TextBox id="txtPhoneNumber" runat="server" width="200px" /><br>
</form>
</body>
</html>
The Web form includes a GetData method as described in Listing 3, but it now has text fields to receive data retrieved through Web service calls. The Web service is called when a selection is made in the DropDownList control. The control's SelectedIndexChanged event is called automatically (the AutoPostBack property of control is set to True) when a user selects a value from the list.

The selected value from the control is passed to the Web service to retrieve each piece of data (such as address and city) through separate calls. The code uses the WebReference object added from the Add Web Reference dialog box to access the Web service.

### Conclusion

Integrating disparate systems is a common development task for enterprise developers. One such endeavor is mixing Lotus Domino with the Microsoft .NET platform. You can accomplish this integration in any of several ways, using COM technology as well as the XML and Web services readily available. These options make it easy to connect IT resources to meet your business needs.
Related topics

- developerWorks Lotus article, "Lotus Notes/Domino 7 Web Services"
- Microsoft .NET Framework Developer Center
- Microsoft COM site
- Download a trial version of Lotus Notes from developerWorks.
- Download a trial version of Lotus Domino from developerWorks.
- Download the Microsoft .NET Framework SDK.

© Copyright IBM Corporation 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)