COM Together — with Domino

Søren Peter Nielsen, Kjetil Meidell Andenes, Kevin P. Smith

International Technical Support Organization
www.redbooks.ibm.com
COM Together — with Domino

How to use your BASIC skills to build collaborative applications

May 2000
Take Note!
Before using this information and the product it supports, be sure to read the general information in
the Special Notices section at the back of this book.

First Edition (May 2000)
This edition applies to Lotus Domino R5.0.2b

Comments may be addressed to: IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the
information in any way it believes appropriate without incurring any obligation to you.

© International Business Machines Corporation 2000. All rights reserved.

Note to U.S. Government Users: Documentation related to restricted rights. Use, duplication or disclosure
is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Contents

Preface .............................................  v
The team that wrote this redbook .................  v
Comments welcome ................................. vi
1 Domino Object Model ......................... 1
   Domino services ............................... 1
      Object store ................................ 2
      Directory ................................... 2
      Security .................................... 2
      Replication ................................. 2
      Messaging ................................... 2
      Workflow ................................... 2
      Automation by agents ....................... 3
   Domino Designer ............................... 3
   Front-end and back-end classes ................. 3
   Object hierarchy ............................... 6
   Summary ....................................... 6
2 Introduction to COM and the Domino object components ............... 7
   What is COM? .................................. 7
      A real world example ....................... 8
   COM components ............................... 8
   Domino object components .................... 9
   What about OLE Automation ................. 10
   Early binding and late binding .......... 10
   C/C++ access to the Domino Object Model .... 11
   A simple example ............................. 12
   Microsoft Word 2000 ......................... 12
   Microsoft Internet Explorer ............... 14
   A brief look at the code ..................... 16
   Summary ..................................... 18
3 Using MS Office and VBA .................... 19
   General setup information for Domino and Microsoft Office .... 19
      Setting up access to the Domino objects .... 20
   Accessing Domino databases from Microsoft Word using COM and VBA .... 21
      Creating a dialog box to store the names retrieved from Domino .... 21
      Add the code to power your Domino lookup ...................... 24
      Summary of Lotus Domino Object classes used in this example ...... 33
   Accessing Domino databases from MS Excel using COM and VBA .... 34
      Modifying the Expense Statement template to add name lookup .... 35
      Modify the Domino Expense Statement to allow saving in Domino .... 42
      Summary of Lotus Domino Object classes used in this example ...... 53
   Summary ..................................... 53
4 Using Visual Basic ....................... 55
   Create the Visual Basic program ............ 56
      Create a new Visual Basic project ........ 57
      Configure the form objects ............... 58
      Activate the Lotus Domino Objects .......... 59
      Define the global variables and types .... 59
      Create the sub GetDocLibDbs ............. 60
      Create the function ViewCategories ....... 62

iii
5 Using VBScript from Active Server Pages .......... 73
Prerequisites ........................................ 73
Setup .................................................. 74
Domino server security .............................. 75
Making the connection ............................... 76
Creating an instance of the Domino COM object .... 76
Initializing the session .............................. 76
Developer tools ...................................... 78
MS Visual InterDev .................................. 79
IIS setup for developing with MS Visual InterDev ... 79
A simple ASP example ............................... 80
Building the example ............................... 82
Type libraries ....................................... 84
The code ........................................... 85
Comments on the code ............................. 89
To run the example ................................ 93
Extending ASP with components ................. 94
Summary ............................................ 94

6 Using Visual C++ ................................. 95
Prerequisites ....................................... 95
Building the component .......................... 96

Make a project using the ATL COM Wizard ........ 97
Files to enable the Domino COM connection ....... 103
Code for the DominoMail component ............ 105
Summary ............................................ 132

Appendix A Lotus Domino and Notes Toolkit for COM 133
Special notices ...................................... 135
Additional Web material ......................... 139
How to get the Web material ..................... 139
Related publications ............................... 141
International Technical Support Organization publications 141
Other Lotus-related ITSO publications .......... 142
Redbooks on CD-ROMs ............................ 143
Other resources .................................... 144
Web sites ........................................... 144

How to get IBM Redbooks ......................... 145
IBM intranet for employees ...................... 145

Index ................................................. 149

Evaluation Form ................................... 153
Preface

With Release 5.0.2b of Domino, BASIC programmers now can access over 30 objects with more than 600 methods and properties through the Domino COM support. They can take advantage of the wide range of Domino services like object store, directory, security, replication, messaging, workflow, automation through agents, and much more.

In this redbook we investigate how you program and integrate Domino in solutions using COM. We look at how you can integrate Domino with Microsoft Office applications and Visual Basic applications, access Domino objects from VBScript on ASP pages, and how to use Visual C++ to make Domino COM components.

The examples from this book are available on the IBM Redbooks Web site.

This redbook is written for Domino solution designers and programmers, customers, IBM Business Partners, and other members of the IBM and Lotus community who know how to program in BASIC and need a good technical understanding of how to use Domino with COM.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Center at Lotus in Cambridge, Massachusetts, USA.

Søren Peter Nielsen works for the International Technical Support Organization at Lotus Development, Cambridge, Massachusetts. He manages projects that produce redbooks about all areas of Lotus products. Before joining the ITSO in 1998, Søren worked as an IT Architect for IBM Global Services in Denmark, designing solutions for a wide range of industries. Søren is a Certified Lotus Professional at the Principal level in Application Development and System Administration.

Kjetil Meidell Andenas is a Software Architect who works for egroup eon in Norway. He graduated in Informatics from the University of Oslo, Norway. He has worked with product development, mainly using Notes/Domino, for 6 years. He has developed solutions ranging from system architecture of core Domino applications to integration and extension of Domino with external tools.
Kevin P. Smith works for Innovative Technology Services. Kevin provides computer consulting services including programming and training, specializing in Lotus Notes/Domino and office automation.

A number of people have provided support and guidance. We would like to thank the following people from Lotus (unless otherwise noted):

- Gary Devendorf
- Richard Adams
- Stephen Auriemma
- Janet McCann, Iris Associates
- Rohit Mehta, Iris Associates
- David Morrison, ITSO Cambridge
- Steve Nicopoulos, Iris Associates
- Bob Perron
- Alison Chandler, ITSO Poughkeepise
- Graphic Services, Lotus North Reading

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found at the back of this book to the fax number shown on the form.
- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Send your comments in an Internet note to redbook@us.ibm.com

vi  COM Together — with Domino
The Domino Object Model gives you as a developer access to a wide range of services — like object store, directory, security, replication, messaging, workflow, automation through agents, and more — in a consistent way.

Through more than 30 objects, with over 600 methods and properties, you can use all those services from COM-enabled languages like Visual Basic, VB for applications, VBScript, C++, and LotusScript, as well as from Java™ and other languages that can use Common Object Request Broker Architecture/Internet Inter-ORB Protocol (CORBA/IIOP).

Before we look further at the Domino Object Model we will give an overview of the services that can be accessed through the Domino objects and a brief description of Domino Designer, the dedicated Domino development client. We will then discuss the difference between front-end classes and back-end classes in Domino and what the Domino object hierarchy looks like.

Domino services

Domino services are offered by Domino servers, of which there are three types:

- Domino Mail Server
- Domino Application Server
- Domino Enterprise Server

You can utilize all three server types from COM programs. Depending on how you access Domino, you may also need a Notes client on your workstation. For example, to access Domino through COM from MS Office applications you must have a Notes client installed on the user’s workstation.

The most important services you can access through the Domino Object Model are described in the rest of this section.
Object store

Documents in a Domino database can contain any number of objects and data types, including text, rich text, numerical data, structured data, images, graphics, sound, video, file attachments, embedded objects, and Java and ActiveX applets. A built-in full text search engine makes it easy to index and search documents. The object store also lets your Domino applications dynamically present information based on variables such as user identity, user preferences, user input, and time.

Directory

A single directory manages all resource directory information for server and network configuration, application management, and security. Domino includes user account synchronization between NT and Domino, and it is Light Weight Directory Access Protocol (LDAP) compliant. The directory is the foundation for easily managing and securing your Internet and intranet applications.

Security

The Domino security model provides user authentication, digital signatures, flexible access control, and encryption. Domino security enables you to extend your intranet applications to customers and business partners.

Replication

Bi-directional replication automatically distributes and synchronizes information and applications across geographically dispersed sites. Replication makes your business applications available to users around your company or around the world, regardless of time or location.

Messaging

An advanced client/server messaging system with built-in calendaring and scheduling enables individuals and groups to send and share information easily. Message transfer agents (MTAs) seamlessly extend the system to Simple Mail Transfer Protocol (SMTP)/Multipurpose Internet Mail Extension (MIME), X.400, and cc:Mail™ messaging environments. The Domino messaging service provides a single server supporting a variety of mail clients: Post Office Protocol V3 (POP3), Internet Message Access Protocol V4 (IMAP4), Message Application Programming Interface (MAPI), and Lotus Notes clients.

Workflow

A workflow engine distributes, routes, and tracks documents according to a process defined in your applications. Workflow enables you to coordinate
and streamline critical business activities across an organization, and with customers, partners, and suppliers.

**Automation by agents**

Agents enable you to automate frequently performed processes, eliminating tedious administration tasks and speeding up your business applications. Agents can be triggered by time or events in a business application. Agents can be run on Domino servers or Lotus Notes clients.

---

**Domino Designer**

Domino Designer is general-purpose client software featuring an integrated development environment (IDE) that provides easy access to all features of the Domino server. You use Domino Designer for dedicated Domino application development using formula language, LotusScript, JavaScript and Java (in agents). When programming in LotusScript in Domino Designer you can access the Domino Object Model through COM. However, in this book we will focus on accessing Domino through COM from outside tools and application. If you want to learn more about Domino Designer and how to create applications for Domino, refer to *Lotus Domino R5.0: A Developer’s Handbook*, IBM form number SG24-5331, Lotus part number CC7EDNA.

---

**Front-end and back-end classes**

If you program applications in Domino Designer you can work directly with objects that the user has open in the Notes client (database, view, document and so on). You access this functionality through front-end and back-end classes.

Front-end classes require a user interface to access Domino functionality. Back-end classes, in contrast, do not require a user interface to achieve the same access.

The Domino Object Model comprises all the Domino back-end classes:

- **NotesSession**
  The Domino environment of the current script, providing access to environment variables, Domino directories, information about the current user, and information about the current Domino platform and release number.

- **NotesDbDirectory**
  The Domino databases on a specific server or local machine.
• NotesDatabase
  A Domino database.

• NotesACL
  The Access Control List (ACL) of a database.

• NotesACLEntry
  A single entry in an Access Control List. An entry may be for a person, a group, or a server.

• NotesAgent
  An agent.

• NotesView
  A view or folder of a database and provides access to documents within it.

• NotesViewColumn
  A column in a view or folder.

• NotesViewEntry
  A view entry, which represents a row in a view.

• NotesViewEntryCollection
  A collection of view entries, selected according to specific criteria.

• NotesViewNavigator
  A view navigator, which provides access to all, or a subset of, the entries in a view.

• NotesDocumentCollection
  A collection of documents from a database, selected according to specific criteria.

• NotesDocument
  A document in a database.

• NotesItem
  A piece of data in a document. All of the items in a document are accessible through LotusScript, regardless of what form is used to display the document in the user interface.

• NotesRichTextItem
  An item of type rich text.

• NotesRichTextStyle
  Rich text field attributes.

• NotesRichTextParagraphStyle
  Rich text paragraph attributes.
• NotesRichTextTab
  Rich text tab attributes.

• NotesEmbeddedObject
  Embedded objects, linked objects, and file attachments.

• NotesDateTime
  A date and time. Provides a means of translating between the LotusScript date-time format and the Notes format.

• NotesDateRange
  Contains a range of NotesDateTime. An object of type NotesDateTime represents a given date and time.

• NotesLog
  Enables you to record actions and errors that take place during a script’s execution. You can record actions and errors in a Notes database, a mail memo, or a file (for scripts that run locally).

• NotesNewsletter
  A document that contains information from, or document links to, several other documents. All of the NotesItem properties and methods can also be used on a NotesRichTextItem.

• NotesForm
  A form in a Notes database.

• NotesInternational
  This object contains properties that provide information about the international settings (for example, date format), of the environment in which Domino is running.

• NotesName
  Properties of this object contain information about a Domino user name.

• NotesRegistration
  Represents the creation or administration of an ID file.

• NotesOutline
  The Notes Outline attributes.

• NotesOutlineEntry
  An entry in a Notes Outline.

• NotesReplication
  The replication settings of a database.

To learn more about an individual Domino back-end class, its properties and methods, refer to the online help in Domino Designer.
Object hierarchy

There is a hierarchical relationship for Domino objects. Higher hierarchical objects contain the lower ones. The figure below is an example of the hierarchical relationship between a few of the Domino objects:

Each object has defined members, properties and methods. Using these members, you can access other objects. The relationship of containment and access means that the higher object has the property or the method to access the lower one.

For example, you can see all the views when you open the database. This means that the opened database(object) includes the views(object). Furthermore, you can see the documents when you select one of the views. This means that your selected view(object) contains the documents(object). This hierarchy is important when using Domino objects. NotesSession is the top level object in the Domino Object Model. You can work your way to any Domino object if you start from NotesSession.

With this basic introduction to the Domino Object Model we are now ready to discuss what COM is. We will do this and show some simple examples of accessing Domino through COM in the next chapter.

Summary

In this chapter we have given a brief overview of the Domino services object store, directory, security, replication, messaging, workflow and automation through agents. We have explained the difference between front-end and back-end classes, and listed and defined the objects that make up the Domino Object Model.
Chapter 2
Introduction to COM and the Domino object components

This chapter provides a brief overview of the Component Object Model (COM) and gives some simple examples of how to access the Domino objects from COM.

What is COM?

COM, or the Component Object Model, is an open software component specification developed by Microsoft. It defines a specification for developing reusable binary components across multiple software languages and platforms. COM components can be written and called by any language that can call functions through pointers, including C, C++, Delphi, and Basic, just to name a few. Because of this flexibility, COM components can be built by different software vendors and incorporated in endless combinations to create applications and systems. COM is one of the basic building blocks in newer Windows technology, including OLE services.

The COM specification provides:

- Rules for component-to-component interaction
- A mechanism for publishing available functions to other components
- Automatic use tracking to allow components to unload themselves when no longer needed
- Efficient memory usage
- Transparent versioning
A real world example

COM is kind of the ultimate black box of software components. Let us try to explain it using a simple analogy. Imagine that you could buy a single black box (component) for your home that provided heating and cooling capabilities to a variety of devices in your home using a standard plug (interface). Each thing in your home that needed heat or cold could plug into this box and request the amount of heating or cooling that it needed:

To upgrade or change manufacturers of the heating and cooling black box, you simply swap in a new one. None of the appliances would ever know that it changed. By the same token, the manufacturer of the black box could decide to change the way they produce heating or cooling inside the black box, perhaps to make it more efficient, and none of the appliances would ever know.

COM components

When it comes to COM components, they also offer an interface to applications. The mechanism to make this interface available to other software objects is covered by the COM standard. It is up to the COM component developer, though, to decide what the actual interface should look like. The decision of what properties and services (methods) to make available from the outside is an important one. Applications using the COM component will depend on having these methods available throughout the
components’ lifetime. The developer can change the internal mechanisms of a COM component, but the applications using that COM component should never need to know. They should continue working exactly as they always have because the interface to the component did not change. New interfaces can be added to the component, so long as the old ones still continue to function in the same manner. This provides an upgrade path without causing obsolescence to those components relying on the older interface. If a developer wanted to take advantage of the new interface, they could do so, but they would not have to in order for their applications to continue functioning.

As we said, this is a very simplistic view of COM. There are many other aspects to the COM specification that make it a very good component platform. For more information on COM, see http://www.microsoft.com/com

**Domino object components**

The Lotus Domino object COM components can be thought of like black box access to Domino, similar to our example for heating and cooling, as illustrated in the following:

The actual mechanism for accessing Domino is hidden from the application developer, who relies fully on the interface provided by Lotus. Knowing this interface will be your key to success in utilizing the features of Lotus Domino like creating workflow applications, document archives with integrated security, and all the other wonderful things that Lotus Domino is famous for providing.

Developers familiar with the Domino Object Model used when programming LotusScript against Lotus Domino will recognize most of the interface classes in the COM interface. Some changes have been made, mostly to make
sense in the new context. For example, when accessing the NotesSession object from LotusScript it has a property called currentDatabase, which identifies the database that the calling code resides in. At this time, the COM object does not have the concept of a current environment so it does not make sense to let the NotesSession interface have a currentDatabase property.

**What about OLE Automation**

Developers have had access to the Domino Object Model through OLE Automation since Notes R4.0. You may ask yourself why COM support is needed as well. There are two main reasons:

- OLE Automation requires Notes to be running.
  
  You cannot use Domino services through OLE Automation without having the Notes client launched in the foreground.

- OLE Automation only allows late binding.
  
  Using COM you can choose between early and late binding. We will discuss the differences between the two in the next section, but early binding gives better performance and stability during application execution.

On the other hand, if you want to use the Notes client to handle the user interface or parts of it in your application, you must use OLE Automation. You cannot control the Notes user interface via COM.

**Early binding and late binding**

The difference between early binding and late binding is like the difference between speed and flexibility.

In early (compile-time) binding, the object definitions are available. They can be typed correctly and syntax checking can be performed.

In late (run-time) binding, objects are generic during development. There is an overhead when casting an object to its correct type during run time, and there is also the danger of syntax errors.

Using OLE Automation you can only access Domino objects with late binding. Here is an example that opens the user’s mail database:

```vba
Dim session as Object
Dim db as Object

Set session = CreateObject(“Notes.NotesSession”)
Set db = session.GetDatabase(“”, “”)
Call db.OpenMail
```

10 COM Together — with Domino
The variables named `session` and `db` are of type `Object`, which means they can be anything. The compiler will not be able to tell whether the `GetDatabase` method is valid for the session object because it does not know what kind of type `session` is.

With COM you can do early or late binding as you please (or as demanded by the language you use). Here is an example of early binding with COM:

```vba
Dim session As New NotesSession
Call session.Initialize
Set notesdir = session.GetDbDirectory(""")
Set maildb = notesdir.OpenMailDatabase()
```

If the method `GetDbDirectory` or the method `OpenMailDatabase` is spelled wrong, or if you, for example, pass an integer as parameter to any of the methods, the compiler will show an error when you attempt to save your code.

To do late binding in COM your code can look like this:

```vba
Dim session
set session = CreateObject("Lotus.NotesSession")
session.Initialize
Set notesdir = session.GetDbDirectory(""")
Set maildb = notesdir.OpenMailDatabase()
```

While it may look similar to the example of late binding for OLE Automation, note that the argument for `CreateObject` here is:

`Lotus.NotesSession`

This is because we are using the COM implementation of the Domino Object Model, where Notes.NotesSession gives access to the LotusScript implementation of the model.

The examples in the next section show both early and late binding of Domino objects using COM.

**C/C++ access to the Domino Object Model**

It is worth mentioning that the COM support lets C and C++ programmers take advantage of the unified way to program Domino through its object model. Before the COM support C and C++ programmers only could access Domino through its API. The API consists of low level calls and has at times been lagging the releases of Domino objects.
A simple example

It is very easy to get started using the Domino objects from COM. In this example we will create a small, simple piece of code that checks how large a user’s Lotus Notes Mail file is, and if it is over 50MB, displays a message to the user asking them to perform some housekeeping. As our container applications, we are going to write this same piece of code for Microsoft Word 2000 using Visual Basic for Applications and Microsoft Internet Explorer using VBScript.

Microsoft Word 2000

This example will modify a commonly used word template called Normal.dot, which is the blank sheet template most people use to create a blank document. We will add our code to the new document event so that when a user creates a new document using this template we will check to see if their mail file is too large. Here are the steps to follow:

1. Launch Microsoft Word by choosing Starter Programs - Microsoft Word from the Windows Task Bar.
2. Choose Tools - Macro Visual Basic Editor from the Word menu bar to display the following screen.
3. Double-click the text ThisDocument in the left-hand panel under the Normal project to open it in the right-hand panel.
4. At the top of the right-hand panel are two drop-down list boxes; select Document from the first drop-down list and New from the second. Your screen should now look like the one below.

![Image of drop-down list boxes]

5. From the Menu bar choose Tools - References to display a list of available Active/X components on your system. Scroll down to Lotus Domino Objects and select it by clicking in the small square to the left of the text. The screen below shows this.

![Image of references dialog box]

**Important** If you do not do this, the code will not work

**Note** If you do not see an entry for Lotus Domino Objects and you have upgraded Notes or Domino from an earlier release using the incremental upgrade installer you may have to enable the COM support manually. Do this by selecting Run on the Windows Start menu and then execute the following command (substituting c:\lotus\notes with the path where your Notes Domino files are installed):

```
regsvr32 c:\lotus\notes\nlsxbe.dll
```
1. Type the following code in the right-hand panel, underneath Private Sub Document_New() and before End Sub

```vbnet
Private Sub Document_New()
    Dim session As New NotesSession
    Call session.Initialize
    Set notesdir = session.GetDbDirectory(""")
    Set maildb = notesdir.OpenMailDatabase()
    If maildb.Size > 52428800 Then
        Message = "Your Lotus Notes mail file is:"
        Message = Message & ((maildb.Size/1024)/1024) & "Mb."
        Message = Message & Chr(13) & Chr(13)
        Message = Message & "The company recommended mail file size is 50Mb."
        Message = Message & "Please perform housekeeping as soon as possible."
        MsgBox Message, , session.CommonUserName
    End If
End Sub
```

2. Select File - Save from the menu to save the code.

3. Close the Visual Basic Editor Window.

4. Switch back to Microsoft Word 2000 and choose File - New from the menu to bring up the New document dialog box. Select the Blank Document template and click OK. As the new document is opened the Visual Basic code is executed and you will see the following dialog box on your screen.

![Dialog Box](image.png)

### Microsoft Internet Explorer

In this example, we use VBScript in Internet Explorer to run a similar piece of code that has some extra HTML formatting.

1. Open Notepad from the Start menu by choosing Programs - Accessories - Notepad.
2. Type the following code into Notepad.

```html
<HTML>
<HEAD>
<TITLE>Initialize Session</TITLE>
<SCRIPT LANGUAGE="VBScript">
Sub CheckMailSize
  Dim session

  set session = CreateObject("Lotus.NotesSession")
  session.Initialize

  Set notesdir = session.GetDbDirectory(""
  Set maildb = notesdir.OpenMailDatabase()
  If maildb.Size > 52428800 Then
    document.write("Your Lotus Notes mail file is : ")
    document.write((maildb.Size / 1024) / 1024 & " Mb.<BR><BR>")
    document.write("The company recommended mail file size is 50Mb.<BR>")
    document.write("Please perform housekeeping as soon as possible.")
  End If
End Sub
</SCRIPT>
</HEAD>
<BODY onLoad="CheckMailSize"/>
</BODY>
</HTML>
```

3. Choose File - Save from the menu bar and type in a filename. We chose C:\NotesCOM.html. Make sure that you change the Save as type drop-down list to All Files before you click the Save button.


5. Open Internet Explorer; in the address bar type C:\NotesCOM.html and press Enter.
6. Depending on your Internet Explorer security settings, you may see the following dialog box appear. If you do, click Yes.

![Internet Explorer Security Dialog Box]

7. You should see a screen similar to the one below.

![Notes Mail Size Screen]

**Note** If you are not logged in to Notes, you will need to enter your Notes password when prompted.

**A brief look at the code**

As you can see, we’ve managed to write a very useful utility in a few lines of code. With the Microsoft Word 2000 code, users will be reminded to perform housekeeping whenever they create a new document. In Internet Explorer, this simple piece of code could be used in a company intranet home page to remind users when they need to clean up their mail file.

The code, while very simple, demonstrates the two methods you’ll need to use to access Domino via the COM interface.
In the Microsoft Word 2000 code we first declare a new variable to be of type NotesSession and activate it using the Initialize method.

```vbscript
Dim session As New NotesSession
Call session.Initialize
```

We then find the user’s Notes mail file, which is a property of NotesDirectory, but first we need to get the NotesDirectory by calling the GetDbDirectory method.

```vbscript
Set notesdir = session.GetDbDirectory(""
Set maildb = notesdir.OpenMailDatabase()
```

Once we have the mail database we can access its properties and compose a message if the mail file size is over 52428800 bytes in size, which is 50MB.

```vbscript
If maildb.Size > 52428800 Then
    Message = "Your Lotus Notes mail file......."
Finally, we display the message to the user using the MsgBox function and another property of the NotesSession object, CommonUserName.

```vbscript
MsgBox Message, , session.CommonUserName
```

The code for our Internet Explorer example is similar, except we have used some HTML to control the formatting and a different way of accessing the NotesSession object. Here we first declare a session variable, access it using the Visual Basic CreateObject function, and then activate it using the Initialize method.

```vbscript
Dim session
set session = CreateObject("Lotus.NotesSession")
Session.Initialize
```

We could have used a MsgBox function to send our message to the browser, as in the Word 2000 example, but in this case, we used the document.write method.

Also, note that this code is executed when the HTML page is loaded. This is done by adding a simple piece of HTML to the <BODY> tag:

```html
<BODY onLoad="CheckMailSize">
```

The onLoad event is triggered when an HTML document has finished loading. In this case it runs the CheckMailSize VBScript Function.
Summary

We have defined the Component Object Model and explained the concept of a plugable component infrastructure through the real world metaphor of heating and cooling. We have explained how COM works together with Domino objects, and discussed early and late binding. Finally, we have shown two simple examples of accessing Domino from MS Word and Internet Explorer.
Chapter 3
Using MS Office and VBA

You can develop Domino applications using Visual Basic for Applications (VBA) and Microsoft’s Component Object Model (COM) interface. In this chapter, we show you several examples to help you do this.

In this chapter, we discuss:

• The general setup instructions
• Examples using:
  • Microsoft Word
  • Microsoft Excel

General setup information for Domino and Microsoft Office

In order to use the COM classes for Domino with Microsoft Office, you must have at least one of the following installed on the machine where the code will be running:

• Domino Designer 5.0.2b or later
• Domino Server 5.0.2b or later
• Notes Client 5.0.2b or later

Note If you have upgraded Notes or Domino from an earlier release using the incremental upgrade installer you may have to enable the COM support manually. Do this by selecting Run on the Windows Start menu and then execute the following command (substituting c:\lotus\notes with the path where your Notes Domino files are installed):

```
regsvr32 c:\lotus\notes\nlsxbe.dll
```

For Microsoft Office, you will need Microsoft Office 97 or later. During development, you will need to enable the reference to the Lotus Domino Objects library in the Visual Basic Editor. Detailed instructions are provided in each of the following sections for particular Microsoft Office components. Your code will remember the reference, so it is not necessary to do this everywhere the code will be running.
Setting up access to the Domino objects

In Chapter 2 we showed how to set up access to the Domino objects in Microsoft Word. The same steps apply for both Microsoft Excel and Microsoft Access; to make sure you do not forget we repeat the steps here.

To access the Domino Objects in your VBA code:

1. Launch Microsoft Word, Excel, or Access.
2. After opening a document, workbook, or database, choose Tools - Macros - Visual Basic Editor (Alt + F11).
   **Tip** You may have to move your cursor down to the bottom of the drop-down menu to extend the menu choices far enough to show the Macros option.
3. Choose Tools - References. You will see the following dialog box.

![References dialog box](image)

4. Scroll down the selections and choose Lotus Domino Objects (If it has not been used recently, the Lotus Domino Objects library will be listed alphabetically).
   **Note** Deselect any objects (other than the defaults) that you don’t need, and make Lotus Domino Objects high in the priority list for better performance.
5. Click OK.
   The Lotus Domino Objects will now be available in the object browser while you are writing your application.
Accessing Domino databases from Microsoft Word using COM and VBA

In this example, we are going to create a button in Word that can be used to select a person from a Domino Personal Name and Address Book (the example could easily be modified to work with the Domino Directory as well). We will then populate our Word document with the person’s name, address, and a salutation. A high-level overview of the steps involved follows. The details of each step are described in the remainder of this section.

- Create the dialog box for selecting a person (which we will reuse in our next example).
- Add the code that looks up the list of people from the Domino Personal Name and Address Book.
- Add the code for getting information for the selected person and placing it in the Word document.
- Save the document as a Word template and add a toolbar from which to invoke the lookup dialog.

Note: The screens shown in the example are from Microsoft Word 2000. The example will also work in Microsoft Word 97.

Creating a dialog box to store the names retrieved from Domino

We will need a place to display the names we retrieve from our directory or address book, so we will create a form that we can display as a dialog box. The user can then make a selection from the form, which we will use to retrieve the associated data from Domino and place it in the Word document.

Create a user form which will be the basis for your dialog box

1. Enable the design mode in the VB Editor by clicking the Design Mode icon on the toolbar.
2. Insert a new user form by clicking the Insert User Form button on the toolbar.

3. Your screen should now look similar to this:

Change the form caption and add instructional text
1. In the properties box on the left side of your screen, change the caption to read:

   **Domino Name Lookup**

2. Click back on the form itself to make the form toolbox reappear, then select the label tool (second from the left, top row).
3. Click on your form with the label tool and drag to create a label where we will place instructions for the dialog box.
4. Click inside the label and edit the text to read:

   **Select a name from the list below and click OK:**
5. Click anywhere outside the label box; your screen should now look like this:

![Image of form design](image)

**Add a list box, and OK and Cancel buttons to the form**

1. To add a list box to your form, click on the ListBox tool (top right button on the toolbox).

2. Click on your form below the label text and drag to create the list box (leave room for OK and Cancel buttons below the list box).

3. Using the CommandButton tool (directly below the ListBox tool), create two side-by-side buttons for your OK and Cancel buttons.

   **Tip** After creating your first button, use copy and paste to create another button of the same size, which you can then drag into position.

4. Select the first button and change the Caption in the Properties box to **OK** and make sure the Default property is **True**.

5. Select the second button and in the Properties box, change the Cancel property to **True** and the Caption to **Cancel**.

Chapter 3: Using MS Office and VBA  23
6. Your dialog box should now look similar to this:

![Image of a dialog box]

7. If you haven’t done so, it would be a good idea to save your work at this point.

**Add the code to power your Domino lookup**

Now we will create the actual code that will power the dialog box we just built. This is where we actually get started with the Domino COM objects. In general, we’re going to:

- Create a new code module and add our Domino calls to populate our list box.
- Add code to our two command buttons (OK and Cancel) to make them functional.
- Save our document as a template.
- Create a toolbar and add a button so that the user has an easy way to call the macro that performs the lookup.

**Create a new code module and add code**

From the Microsoft Visual Basic editor choose Insert - Module from the menu bar. A new code module text box will appear.

Since we will be interacting with our form in another part of our project, we will need a public variable to store the user’s selection from the dialog box. We will get the user’s selection from the ListIndex property of our ListBox object, so declare a variable in the General Declarations area of the code module (which is the default section of the code module now displayed):

```vbnet
Public IntPerson as Integer
```
Insert the first sub procedure (sub). There are several ways to do this. If you are a LotusScript programmer, you might be accustomed to just typing `Sub subname` to start your sub. You can do the same thing here, or you can choose Insert - Procedure from the menu and fill out the properties box. Whichever method you choose is fine in this case. Call your procedure PopulateList so it will match our example; then press OK if you used the dialog box, or simply press Enter if you typed it in the code module text box.

**Note** All subs, by default, are public in scope. If you used the dialog box, the Public keyword is inserted for you, but it isn’t needed. Look up “Sub Statement” in the VB Online Help for more information on the sub statement.

Now we will declare (Dim) our variables for this sub as follows:

```vbnet
Dim DomSession As New NotesSession
Dim DomDir As NotesDatabase
Dim DomContacts As NotesView
Dim DomDoc As NotesDocument

Dim StrName As String
Dim doc As Document  ‘this will be our reference to the
                      ‘Word document, it has no relation to
                      ‘the NotesDocument.
Dim myRange As Range
```

Access to the Domino objects must start with the creation of a NotesSession, which can be declared as we have done here, or by using the CreateObject construct:

```vbnet
Dim DomSession As NotesSession
Set DomSession = CreateObject(“Lotus.NotesSession”)
```

DomDir will hold our Name and Address Book. DomContacts will hold the reference to the People view we will be using for populating our list box and document. DomDoc will be the reference to the Domino document (record) we are currently accessing. StrName is a regular VB String, where we will store the composite name that we piece together from the FirstName, MiddleInitial, and LastName fields in our Domino document. The variable doc will be used to reference the current Word document. The variable myRange will represent the current insertion range in our Word document.
Now we initialize the Domino (Notes) Session:

```
DomSession.Initialize
```

**Note** A password (which matches the Notes ID specified in the NOTES.INI in the KeyFilename= parameter) may be passed as an optional parameter. Without it, the user will be prompted to enter a password.

Once the session has been initialized, we have access to the other Domino objects, which we will set up as follows:

```
Set DomDir = DomSession.GetDatabase("", 
    "redbook\pernames.nsf")
Set DomContacts = DomDir.GetView("Contacts")
Set DomDoc = DomContacts.GetFirstDocument
Set doc = ActiveDocument
```

This code accomplishes the following steps:

1. Open the database we will be using for this example, which is a personal address book based on the Personal Name & Address Book template, pernames.ntf, as opposed to the Domino Directory template which uses the pubnames.ntf. The first parameter refers to the server: in this case, it is local, so we leave the server name blank. The second parameter refers to the database name relative to the Domino data directory: in this example, the file is in a subdirectory in the Domino data directory called “redbook”.

2. We will be using the Contacts view from the personal address book database, so we retrieve our reference to it by passing the name of the view to the GetView method of the database. The view names can be found by viewing the Notes database in the Domino Designer client.

**Note** If you are not familiar with Domino Designer, or do not have access to it, you could alternately use the Views property of the database (DomDir for this example) to retrieve a variant that contains all the views for the database. You would need to write a separate procedure to do this. If you do have access to Domino Designer, it is certainly easier to just open it and view the information interactively.

3. Once we have a reference to the view, the GetFirstDocument method of the view will provide us with a starting point to loop through the documents to populate the list box.

4. Finally, we set the doc variable to reference the active Word document using the VBA call, ActiveDocument, which returns a reference to the active Word document.
Now we are ready for our main processing to populate the list box. We will use a While loop to cycle through all the documents in the view until we run out of documents:

```vba
While Not (DomDoc Is Nothing)

  StrName = DomDoc.GetItemValue("LastName")(0) & ", " & _
  DomDoc.GetItemValue("FirstName")(0) & " " & _
  DomDoc.GetItemValue("MiddleInitial")(0)

  UserForm1.ListBox1.AddItem (StrName)

  Set DomDoc = DomContacts.GetNextDocument(DomDoc)

Wend
```

In the while loop, we use the GetItemValue method of our Domino (Notes) document object, DomDoc, to retrieve the LastName, FirstName, and MiddleInitial, to which we add commas and spaces to give us the proper formatting for our list box. The (0) subscript is necessary because the method returns a variant, which Domino always represents as an array even though these fields only contain one item. Then we use the ListBox AddItem method to add our formatted name to the list box.

**Note** We have to provide an explicit reference to ListBox1 (UserForm1) because it is not part of this code module. ListBox1 is declared in the code module for UserForm1. We could include all of our procedures in the UserForm1 code module and the explicit reference would not be necessary, but generally, only code that directly relates to the objects on the form is stored there.

To round out our while loop, we use the GetNext document method of our view object to retrieve the next document in the view relative to the one we are currently referencing.

Once the list box has been populated, we are ready to show it to the user so they can make their selection:

```vba
UserForm1.Show
```

**Getting the information for the selected person**

Once the user has responded to the dialog box, we’ll be returned here and will continue execution by first checking to see if the user canceled the dialog box. We can do this by inspecting IntPerson to see if it has been set to the value of -1 by the Cancel (CommandButton2) click event. If the user did Cancel, we can exit the sub.

Chapter 3: Using MS Office and VBA  27
If IntPerson = -1 Then 'User pressed Cancel (CommandButton2)
    Exit Sub
End If

If we get past the Cancel check, we can retrieve the document corresponding to the name the user selected from the list box, which populates IntPerson with the number of the selection. Using that value as an index, we’ll use the GetNthDocument method of the NotesView, DomContacts, to retrieve a reference to the document.

Set DomDoc = DomContacts.GetNthDocument(IntPerson)

Once we have the reference, we’ll again use the GetItemValue method of DomDoc to get all the components of the full name, and store the formatted result in StrName for later inclusion in our document. Since not all records will include a middle initial, we’ll check to see if there is one before including it.

StrName = DomDoc.GetItemValue("FirstName")(0) & " "
If DomDoc.GetItemValue("MiddleInitial")(0) <> "" Then
    StrName = StrName & DomDoc.GetItemValue("MiddleInitial")(0) & " . "
End If
StrName = StrName & DomDoc.GetItemValue("LastName")(0)

Now we’re ready to start inserting information into our document. Word allows us to set a “range” for insertion using x,y coordinates which correspond to the row and character position where we want to start our range. The coordinates 0,0 correspond with the first position in the document.

**Tip** There are many possibilities for positioning the insertion point in a Word document. See the MS Word Visual Basic Help for more options.

We’ll use a With block to save a little typing. The With block allows you to specify the object you’ll be operating on once in the beginning of the block, so you can avoid having to type it repeatedly when referencing the same object over and over again.

Set myRange = ActiveDocument.Range(0, 0)
With myRange
    .InsertAfter (StrName & vbCrLf)
    .InsertAfter (DomDoc.GetItemValue("StreetAddress")(0) & vbCrLf)
    .InsertAfter (DomDoc.GetItemValue("City")(0) & ", "

28 COM Together — with Domino
Using the InsertAfter method of Range, we’ll add our name and address at the top of the document, inserting vbCr (carriage return) and formatting where appropriate, and ending with the salutation for a letter. We’re almost done, but we still have to go back to our form and complete our command buttons (OK and Cancel), save our document as a template, and add a toolbar with a button to give the user access to our lookup from the document.

**Adding code to the OK and Cancel command buttons**
The easiest way to add code to the command buttons is to switch back to the form itself. Use the Window menu in the VB Editor and switch back to the window that contains UserForm1 (UserForm). Double-click the OK button; this will position you in the UserForm1 Code module in the click event for the OK command button. Add the following code:

```vbnet
If ListBox1.ListIndex = -1 Then
    MsgBox ("You didn’t make a selection, please select a name and click OK to proceed, or Cancel to quit.")
Else
    IntPerson = ListBox1.ListIndex + 1
    Unload UserForm1
End If
```

If the ListIndex property of our list box, ListBox1, returns -1, then the user pressed OK without making a selection. We use the the MsgBox function to prompt them to make a selection and click OK, or Cancel to quit. If they did make a selection, we’ll set the IntPerson variable to the item number they selected — incrementing by one because the list box starts at 0, but our Domino view corresponding to the document starts at 1. We are now done with the form, so we can unload it by calling the Unload function. At this point we’ll be returned to the point in the code where we called (show) the form in our general code module.
Now switch back to UserForm1 (UserForm), double-click the Cancel button, and insert the following code in the click event of that command button:

```
IntPerson = -1
Unload UserForm1
```

Setting IntPerson to -1 will be the signal to our calling module that the user canceled. Again, we’ll unload the form here because it is no longer needed. We’re almost done, we just need to add a toolbar and button to give the user a way to easily access our lookup macro.

**Save the document as a Word template**

In order to add a toolbar in the next step, we’ll need to specify a template where we’ll store the new toolbar. While we could store the toolbar in the default template Normal.dot, we’ll create a new template just for our example, using the following steps:

1. Switch back to the Word document window by selecting it on the Windows toolbar.
2. From the menu, choose File - Save As. In the Save As dialog box, scroll down and choose Document Template (*.dot) from the drop-down list box.
3. Name it Domino Name Lookup.dot. Your dialog box should look similar to this one:

   ![Save As Dialog Box](image)

4. Click Save.
Add the toolbar
1. Choose Tools-Customize from the menu and click the Toolbars tab.
2. Click New. Name your toolbar Domino Name Lookup. The template should default to the one we just saved, Domino Name Lookup.dot. Your dialog box will now look like this:

3. Click OK. A small toolbar will appear on your screen.
4. To add our macro to it, click the Commands tab on the Customize dialog box.
5. Select Macros from the Categories list box. Your screen should now look similar to this:

Chapter 3: Using MS Office and VBA  31
6. Drag the Project1.Module1.PopulateList Macro to the toolbar on the right and drop it.
7. Right click the toolbar text and choose Name. Name it “Insert Name from Domino”.
8. Close the Customize dialog box, save your form, and you’re done!

**Try it out**

Now you can choose File-New, choose your Domino Name Lookup template, and you’re ready to use your new macro to look up Domino names and addresses to add to your document. Here is what a new document based on the template should look like:
Click the Insert Name from Domino button and your screen should display a dialog box similar to this (you might be prompted for your Notes password first):

![Dialog box](image)

After selecting a name, your document should look similar to this:

![Document](image)

**Summary of Lotus Domino Object classes used in this example**

<table>
<thead>
<tr>
<th>Class</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotesSession</td>
<td>Required before accessing any other Lotus Domino Objects. Don’t initialize more than once for a program or your program may crash.</td>
</tr>
<tr>
<td>NotesDatabase</td>
<td>Provides access to Domino databases.</td>
</tr>
<tr>
<td>NotesView</td>
<td>Provides access to a Domino view, which is a formatted list of documents.</td>
</tr>
<tr>
<td>NotesDocument</td>
<td>Represents a Domino document (record).</td>
</tr>
</tbody>
</table>
Exporting the form and code module for reuse

We’re going to reuse the form and code we created in Word in our Excel example, so if you plan on working through that example as well, you should follow the instructions in the next section to export the form and code we created here.

1. From the project box in the Visual Basic Editor, right click on UserForm1 and choose export:

   ![Image of Visual Basic Editor]

   1. You’ll be prompted with a file save dialog box. Select a location to save your form and click OK. Repeat the process for Module1.

      We are now ready to reuse our code in the next example.

Accessing Domino databases from MS Excel using COM and VBA

In this example, we will show how to look up information from a Domino database and place it in an Excel spreadsheet, much in the same manner as we did in the previous example with Microsoft Word. We will then add the twist of saving the spreadsheet file in a Domino database, allowing shared access to the file. A high level view of the steps we will go through in this example is as follows:
• Modify the Excel Expense Statement template by adding a button that will allow the user to look up information from the Domino Personal Name and Address Book (reusing some of our code from the Word example we just completed).

• Create a Domino database in which to store expense statements.

• Create a button that will allow the user to store their expense statement in the Domino database.

To complete this example, you will need the following:

• The Notes R5.0 Client and the Doc Library Notes & Web (R5.0) template.

• Your exported form and code module from the last example.

  Note  If you did not work through the last example, but you are fairly familiar with Visual Basic, you should not have any trouble recreating the form we used in the last example. All the code (except for the command buttons on the form) will be reshown here as well.

• Microsoft Excel 2000 (you can also use Excel 97, but it may have minor user interface differences) and the Expense Statement template.

Modifying the Expense Statement template to add name lookup

Launch Excel and open the Expense Statement.xlt template that is found under Microsoft Office\Templates\Spreadsheet Solutions. Depending on your security settings, you may be prompted with a dialog box; if so, click Enable Macros.

  Important  Because the macros in the Expense Statement template are password protected, making changes to the template will be a little different from changing or adding macros to a normal Excel workbook. Changes you make will be saved to a linked workbook called COMMON.XLS, which is stored in Microsoft Office\Office\Library. This is especially important to remember if you want to install your modified template on another machine: you will need to make sure you install COMMON.XLS as well as your template file for your modified template to function properly.

Set up access to the Lotus Domino Objects library as described earlier in this chapter in “Setting up access to the Lotus Domino Objects.”

Import UserForm1 and Module1 from our Word example

  1. From the VB Editor in Excel, right click on the VBA Project (COMMON.XLS) and choose Import File.

  2. Navigate to the location where you saved your exported files and import the form UserForm1.frm (assuming you called it that).
3. Repeat steps 1 and 2 for Module1.bas.

**Modify our PopulateList sub for use with the Expense Statement**

UserForm1 does not need to be modified for this example, but we will be making changes to the PopulateList sub.

1. Load the Module1 code in the editor by opening the Modules folder in the project pane of the VB Editor and double-clicking on Module1.

2. Because we will be reusing our Domino Session in other subs, move the declaration from within the PopulateList sub to the (General) (Declarations) section and change Dim to Public. Your (General) (Declarations) should now look like this:

   ```vbnet
   Public IntPerson As Integer
   Public DomSession As New NotesSession
   ```

3. For our declarations in PopulateList, we no longer need a declaration for the Word document, but we will need a new string declaration for testing to see if our Domino Session has already been initialized, so your declarations should now look similar to this:

   ```vbnet
   Sub PopulateList()
   'While it is not required, we have preceded our Domino variable declarations with Dom for clarity.
   Dim DomDir As NotesDatabase
   Dim DomContacts As NotesView
   Dim DomDoc As NotesDocument
   Dim StrName As String
   'Dim doc As Document 'Delete
   'Dim myRange As Range 'Delete
   Dim StrTestDomSession As String 'Add this one
   On Error GoTo ErrorHandler
   ```

4. We did not do any error handling in our first example, but here it is necessary in order to avoid reinitializing DomSession, which caused Excel to crash when we tried it. Unfortunately, it crashed before the error handler was called, so we developed a workaround. Instead of generating an error due to reinitializing DomSession, we will try to use DomSession to retrieve a string value from one of its string properties, which will generate the error ERR_NOTES_SESSION_NOT_INIT. Then the error handler will initialize the session. Add these two lines after your declarations:

   ```vbnet
   On Error GoTo ErrorHandler
   ```
StrTestDomSession = DomSession.CommonUserName

5. We will get back to the error handler in a minute, but let’s go through the rest of the code in the sub first. This portion stays mostly the same as it did in our Word example. We will not need Set doc = ActiveDocument, so you can delete it. It has been commented out in the following code:

   Set DomDir = DomSession.GetDatabase("", "redbook\pernames.nsf")
   Set DomContacts = DomDir.GetView("Contacts")
   Set DomDoc = DomContacts.GetFirstDocument
   'Set doc = ActiveDocument 'Delete this

While Not (DomDoc Is Nothing)
   StrName = DomDoc.GetItemValue("LastName")(0) & ", " & 
   _
   DomDoc.GetItemValue("FirstName")(0) & 
   _
   DomDoc.GetItemValue("MiddleInitial") (0)
   UserForm1.ListBox1.AddItem (StrName)
   Set DomDoc = DomContacts.GetNextDocument(DomDoc)
   Wend

UserForm1.Show

If IntPerson = -1 Then
   Exit Sub
   End If
   Set DomDoc = DomContacts.GetNthDocument(IntPerson)
   StrName = DomDoc.GetItemValue("FirstName")(0) & " 
If DomDoc.GetItemValue("MiddleInitial") (0) <> " " Then
   StrName = StrName & 
   DomDoc.GetItemValue("MiddleInitial") (0) & ". "
   End If
   StrName = StrName & DomDoc.GetItemValue("LastName")(0)

Inserting data into the Excel worksheet
1. The way we insert the data into the Excel Worksheet is a little different from the way we added it to our Word document. After making sure
that the worksheet we want to write to is active, we set the value of
specific named ranges. These ranges are already named in the Expense
Statement, so we do not need to do that.

```vba
Worksheets("Expense Statement").Activate
Range("data1").Value = StrName 'Name
Range("data3").Value = DomDoc.GetItemValue("Department")(0) 'Department
Range("data5").Value = DomDoc.GetItemValue("JobTitle")(0) 'Position
Range("data6").Value = DomDoc.GetItemValue("Manager")(0) 'Manager
```

**Tip** It is not necessary to name a range in order to reference it. You can
also substitute the actual cell identifier. For example, instead of using
"data1" for the name, we could have used the cell identifier and
constructed our code like this:

```vba
Range("E13").Value = StrName
```

2. Now we come to our error handler. In order to skip over the error
handler when there is no error, we will insert a `GOTO Done` statement
that causes the program to skip over the error handler unless it is
specifically called. Then we provide the label for the error handler
`ErrorHandler` which is the partner of the `On Error Goto`
`ErrorHandler` statement we started with near the beginning of this
sub. Once we have that handled, we get to the actual code that deals
with the error. If it is the error we are looking for,
ERR_NOTES_SESSION_NOT_INIT, we will initialize the Domino
session and resume our program on the line following where the error
took place, which should be immediately following

```vba
StrTestDomSession = DomSession.CommonUserName.
```

For any
other error, we are just going to report it to the user and the program
will end. This is where you could add additional error checks if you
wanted to. Here is what the last portion of our sub looks like:

```vba
GoTo Done
ErrorHandler:
If Err = ERR_NOTES_SESSION_NOT_INIT Then
  DomSession.Initialize
  Resume Next
Else
```

38 COM Together — with Domino
Done:
End Sub

Add a button to the Expense Statement form to trigger a lookup
We are almost done reconstructing our lookup. The last thing we need to do
before saving our new version of the Expense Statement is add a button so
that the user has an easy way to access our Domino lookup.

1. Switch back to the actual worksheet in Excel. From the menu choose
   Tools - Customize and the Customize dialog box will appear for the
toolbars. Click the Toolbars tab:

   ![Customize dialog box]

2. Select the Forms toolbar and click Close. The Forms toolbar should
   appear on your screen:

   ![Forms toolbar]

Chapter 3: Using MS Office and VBA  39
3. Select the button tool (second row, right icon) and draw a small box at the end of the Name cell. Immediately, the Assign Macro dialog box will appear, prompting you to assign a macro to the button you just created:

4. Scroll down, choose COMMON.XLS!PopulateList, and click OK.

5. Right click on the button and choose Edit Text. Delete all the text and replace it with a single uppercase V (to resemble a down arrow).

6. Right click the button again and choose Format Control. Change the size to 8, and the style to Bold. Click OK, and then click on any other part of the worksheet to deselect the button. You should now have what looks like a drop-down box:

7. Choose File - Save As and save as Domino Expense Statement.xlt.
To try it out, choose File - New, look for your Domino Expense Statement template, and use it to create a new workbook. The name lookup should look similar to this:

After selecting a name, your filled-in expense sheet should look similar to this (depending on what fields you have completed in the Personal Address Book):

Chapter 3: Using MS Office and VBA  41
Modify the Domino Expense Statement to allow saving in Domino

In the next part of our example, we are going to add the capability to store an expense statement in a Domino Document Library database. We will start by creating a Domino database based on the Doc Library Notes & Web (R5.0) template. Then we will add code to the COMMON.XLS workbook linked to our Domino Expense Statement template and add a button to the Expense Statement form to give the user access to the functionality of saving in Domino. We will also change several small items in the Expense Statement workbook to make it work better with our Domino save functionality.

Create a Document Library database with the Notes Client

We will be using a special Domino database based on the Doc Library Notes & Web (R5.0) template. For those of you who may not be familiar with the process of creating a new Domino database from a template, we will review the steps here. If you are already familiar with this process, go ahead and create a new database based on the Doc Library Notes & Web (R5.0) template and skip to the next section.

Tip Make sure to create the new database on Local, relative to the Notes data directory, for this example. Normally, however, you can access databases on servers as well by simply providing the server name when you open the database. For example:

```lisp
Set DomDb = DomSession.GetDatabase(“Server4/Millenia”, “Redbook\pernames.nsf”)```

1. Launch the Notes R5.0 client and enter your password when prompted.
2. From the menu, choose File - Database - New (Ctrl + N). You should now see the following dialog box:
3. Leaving the server as Local, give your database a title and a filename (you can specify a subfolder as we have done, but do not specify a drive).

   **Caution**  If you use the Browse button, make sure you navigate to the Notes default directory before selecting a folder.

4. Scroll down through the template list box and select Doc Library Notes & Web (R5.0).

5. Your New Database dialog box should now look similar to this:

![New Database dialog box](image)

6. Click OK and you are done.

**Create new code modules to provide Domino save functionality**

We will create three new procedures, and add code to the Workbook_Open event handler. Here is an overview of the three modules and the event handler additions:

- **Sub EmbedInDomino()** - Creates a new document in the Domino Document Library and embeds the current workbook in the document.

- **Sub ReEmbed()** - Deletes the embedded item from the Domino document and re-embeds the changed workbook.

- **Sub SaveInDomino()** - Decides whether or not to call EmbedInDomino for a new expense sheet, ReEmbed to resave it, or if the workbook was launched from the embedded object in Domino, just save it.

- **Workbook_Open Event for COMMON.XLS** - Will delete the temporary file used for saving the workbook to the file system prior to embedding it in Domino.
Create new procedure: EmbedInDomino

Close any open Excel files and reopen the Domino Expense Statement.XLT
template file (it should be in your recent file list), enabling macros if
prompted. From the menu, choose Edit - Links. You should see a dialog
box that looks like this:

You should see your COMMON.XLS worksheet in the list. Choose Open
Source, enabling macros if prompted. Now you should be able to open the
Visual Basic Editor and have access to your code. We will need a few more
Global variables to share among these procedures, so we will add them
first. In the (General) (Declarations) section at the top of Module1, add the
following variable declarations:

Public OLEHandle As Variant 'Handle to OLE object
Public boolEmbedded As Boolean 'Flag to indicate initially
embedded
Public DomLibDoc As NotesDocument 'Reference to library
document
Public DomRTItem As NotesRichTextItem 'Reference to the rich
text field where we will embed our Excel file

Now we add our first procedure:

1. Choose Insert - Procedure from the menu and name the new procedure
EmbedInDomino.
2. Declare the following Domino object variable for referencing the
Document Library we created earlier.
   Dim DomDocLib As NotesDatabase
3. We will need two additional VB variables as well. We will use
StrTestDomSession to test whether our Domino session has been
initialized as we did in PopulateList. The other variable, tempFile will

44 COM Together — with Domino
be the handle to the saved workbook file that we will embed in the
document library.

```
Dim StrTestDomSession As String
Dim tempFile As Variant
```

4. As in the last sub, PopulateList, we will add a call to an error handler
and then check for an initialized Domino session by trying to assign one
of the string properties of DomSession to the StrTestDomSession
variable:

```
On Error GoTo ErrorHandler
StrTestDomSession = DomSession.CommonUserName
```

5. Initialize the Domino Document Library, passing the folder and
filename just as you named it earlier in “Create a Document Library
database in the Domino Client.” The first parameter, server, will be
blank (“”) because we are using a local database.

**Note** For an example of accessing a database on a server, see the tip in
the section “Create a Document Library database with the Domino
Client” earlier in this chapter.

Then create a document in the database by calling the CreateDocument
method of NotesDatabase (DomDocLib), and add a new field to the
Document using the CreateRichTextItem method of NotesDocument
(DomLibDoc) and passing it the name of the new field, “Body”:

```
Set DomDocLib = DomSession.GetDatabase(“”,
“Redbook\DocLib.nsf”)
Set DomLibDoc = DomDocLib.CreateDocument
Set DomRTItem = DomLibDoc.CreateRichTextItem(“Body”)
```

6. Now we will check to see if the user has already saved the workbook to
disk. If not, we will create a FileSystemObject for saving the file. While
we do delete this temporary file during the open event of
COMMON.XLS (we will get to that a little later in “Create event
handler for COMMON.XLS’ open event”) we will check here to see if it
exists, in case the user has created multiple expense statements during
this session. If the file already exists, we will use the office assistant to
let them know that they should replace the file when prompted. Then
we will save the file (Save as if the user hasn’t previously saved it).

Here’s what the code segment that accomplishes this looks like:

```
If ActiveWorkbook.Path = “” Then ‘If user hasn’t already
    saved the file, we will save it.
    Set tempFile =
CreateObject(“Scripting.FileSystemObject”)
```
If tempFile.FileExists("tempExpSheet.xls") Then
    With Assistant.NewBalloon
        .Animation = msoAnimationGetAttentionMinor
        .Heading = "Replace Prompt"
        .Text = "When prompted to replace tempExpSheet.xls, press Yes"
        .Show
    End With
End If
ActiveWorkbook.SaveAs Filename:="tempExpSheet"
Else
    ActiveWorkbook.Save
End If

7. Before we actually add our workbook as an embedded object, we will make sure the Expense Statement worksheet is the active worksheet by calling the Activate method of Worksheets. Then we will tell Domino what form to use for displaying this document in Domino, the “Document” form, by calling the ReplaceItemValue method of the NotesDocument (DocLibDoc) we created to replace the Form field value, or we will create the field if it doesn’t exist. We will categorize (NewCats) the document as “Expense Report” and fill in the Subject field by referencing named ranges (we will explore named ranges in “Modifying the Domino Expense Statement form” a little later) on the expense sheet as well.

Worksheets("Expense Statement").Activate
Call DomLibDoc.ReplaceItemValue("Form", "Document")
Call DomLibDoc.ReplaceItemValue("NewCats", "Expense Report")
Call DomLibDoc.ReplaceItemValue("Subject", "Expense Report for " & Range("data1").Value & ": " & Range("ExpFrom").Value & " - " & Range("ExpTo").Value)

8. We will then embed the workbook into the document using the EmbedObject method of NotesRichTextItem (DomRTItem). The first parameter tells Domino we want to embed an object. Other options are to embed a link (EMBED_OBJECTLINK), or an attachment (EMBED_ATTACHMENT). The second parameter represents the class, or application type of the embedded object, and is left blank if we use the next parameter, source, to pass the filename we want to embed. The
third parameter is the filename, including path, of the file we want to embed.

Call DomRTItem.EmbedObject(EMBED_OBJECT, "", 
ActiveWorkbook.FullName)

9. Before saving the document, tell Domino to calculate all of the formulas in the document (ComputeWithForm) to fill in any default or calculated field values, then save the document. The first parameter of the save command says to force the save even if the document has been modified by someone else, the second parameter says not to make it a response document.

Call DomLibDoc.ComputeWithForm(False, False)

Call DomLibDoc.Save(True, False)

10. So that our other procedures know that we have already embedded the expense statement in Domino, we will set a couple of flags that we can test later, one on the workbook itself in a named range, and the other in the public variable boolEmbedded:

Worksheets("Expense Statement").Range("SavedInDomino").Value = "True"

boolEmbedded = True

11. Finally, we will add the same error handling section that we used in the PopulateList sub:

GoTo Done

ErrorHandler:

If Err = ERR_NOTES_SESSION_NOT_INIT Then
    DomSession.Initialize
    Resume Next
Else
    MsgBox ("Run-time error " & Str(Err) & ":" & vbCr & 
    Err.Description)
End If

Done:

Create new procedure: ReEmbed()

We have handled the situation where we first create a Domino document in the Document Library and embed the current Expense Statement into it. Now we need to handle the situation where the user wants to re-save their
work. We will do this by deleting the field (Body) where we initially embedded the object, recreating the field and re-embedding the object.

**Note** It should be possible to just use the Remove method of NotesEmbeddedObject to delete the embedded expense statement, but we could not get it to work during our testing so we used this alternate method of just deleting the whole field that the object was embedded into.

1. Create the new procedure (Insert-Procedure) and name it ReEmbed.
2. Save the workbook to make sure we have a current copy on disk:
   ```vba
   ActiveWorkbook.Save
   ```
3. Remove the rich text field, Body, from the document by calling the Remove method of NotesRichTextItem, which will remove the embedded object as well. DomRTItem should still be referencing the Body field from when we created it in our EmbedInDomino sub. Save the Domino document.
   ```vba
   Call DomRTItem.Remove
   Call DomLibDoc.Save(True, False)
   ```
4. The next section should be easy to follow because it is almost exactly the same as the steps we followed in the EmbedInDomino sub. We do not need to rewrite the Form and Categories fields because they could not have changed; but the user could have changed some of the information we are using in the Subject field, so we will rewrite that field:
   ```vba
   Set DomRTItem = DomLibDoc.CreateRichTextItem("Body")
   Call DomLibDoc.ReplaceItemValue("Subject", "Expense Report for " & Range("data1").Value & ": " & Range("ExpFrom").Value & " - " & Range("ExpTo").Value)
   Set DomEmbObj = DomRTItem.EmbedObject(EMBED_OBJECT, "", ActiveWorkbook.FullName)
   Call DomLibDoc.ComputeWithForm(False, False)
   Call DomLibDoc.Save(True, False)
   ```

And that is all there is to the ReEmbed sub. Now we will move on to the procedure that ties these two together: SaveInDomino()

Create new procedure: SaveInDomino
This procedure is the one we will call from a button within the worksheet. It will decide whether the workbook needs to be embedded, re-embedded, or just saved in the case where the workbook was actually launched from within Domino.
1. Create the new procedure (Insert - Procedure) and name it SaveInDomino.

2. To give the user some indication that we are actually saving their document, we will use the Microsoft Office Assistant’s animation for saving a document:
   Assistant.Animation = msoAnimationSaving

3. Our logic for determining how to save the expense statement is just an if statement where we first test the global variable boolEmbedded, which we set to True in EmbedInDomino. If it is True, we know we need to re-embed, so we call ReEmbed. If boolEmbedded is False, then we look at the named range in our workbook, savedInDomino. If it is True, then the expense statement must have been launched from within the Domino document, so we just save it, which will save the embedded object in the Domino container it was launched from. If neither of those is true, then we need to call EmbedInDomino because this workbook has never been embedded before.
   If boolEmbedded Then
       Call ReEmbedd
   ElseIf Range(“SavedInDomino”) = True Then
       ActiveWorkbook.Save
   Else
       Call EmbeddInDomino
   End If

4. The last thing we do is set the Office Assistant’s animation back to its idle state:
   Assistant.Animation = msoAnimationIdle

Add an event handler for the COMMON.XLS open event
Ideally, we would delete the temporary file we created on disk as the user closed the event statement they have been working on. However, we cannot do that because the document is still in use until after our code has been unloaded from memory. So we will delete it on the open event of COMMON.XLS to keep the user from getting a replace prompt the next time we need to save to the temporary file.
1. In the project window, right click on ThisWorkbook in the VBAProject(COMMON.XLS) folder and select View Code:

2. From the Object drop-down box, select Workbook:

3. The Workbook_Open procedure should open automatically; if not, use the pull-down to the right to select the Open event.

4. Declare an object to hold the reference to the temporary file:

   ```vba
dim tempFile as Object
```

5. Create a FileSystemObject handle and assign it to the object variable:

   ```vba
   set tempFile = CreateObject("Scripting.FileSystemObject")
   ```

6. Check to see if the temporary file exists so we do not generate an error trying to get a file that does not exist, then get the file and delete it:

   ```vba
   if tempFile.FileExists("tempExpSheet") then
   ```
tempFile.GetFile ("tempExpSheet.xls")

tempFile.DeleteFile "tempExpSheet.xls", True

End If

Modifying the Domino Expense Statement form
We need to add a button to allow the user to access our procedures for saving the expense statement in Domino. We will also make a few changes to the named ranges in the form (the data entry screen).

1. Switch back to the Domino Expense Statement.XLT worksheet in Excel and turn on the Forms toolbar (choose Tools - Customize, then check Forms) and close the Customize box.

2. Select the button tool (second row, right icon) and draw a button below the TOTAL cell large enough to display the text “Save in Domino.” Select COMMON.XLS!SaveInDomino when prompted to assign a macro and click OK. Change the text on the button by double-clicking on the button and labeling it “Save in Domino.” Click anywhere outside of the button and your form should now look similar to this:

3. Select the cell just below the button you just created (it should be N41 or N42 depending on the exact placement of your button) and from the menu choose Insert - Name - Define and name your selection “SavedInDomino,” and click OK.

Chapter 3: Using MS Office and VBA  51
**Tip** If you want to keep the text “True” from displaying to the user, change the font color for the SavedInDomino cell to White.

4. Now click on the cell to the right of From in the Pay Period box. Notice in the Name Box (upper left hand corner of the screen below File) the name of this cell is “data7.” For some reason, when this form is launched from within Domino, the Domino hierarchical name for the user will replace this named range, so we are going to move “data7” to another cell so that it will not interfere with the “From” date. Move down to cell L45, just above “Fine Print Here” and choose Insert - Name - Define, name the new location “data7” and click OK.

5. Now we need to give the From data a new name, so go back to that cell, choose Insert - Name - Define, and name that location “ExpFrom.”

6. Do the same thing for the To cell, calling it “ExpTo.”

7. Save the Domino Expense Form template, close it, and create a new Domino Expense Form to give it a try!

**Note** You will notice two differences in your display after saving in Domino: the name of the file in the title bar, and the “True” flag under the save button.
### Summary of Lotus Domino Object classes used in this example

<table>
<thead>
<tr>
<th>Class</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotesSession</td>
<td>Required before accessing any other Lotus Domino Objects. Don’t initialize more than once for a program or your program may crash.</td>
</tr>
<tr>
<td>NotesDatabase</td>
<td>Provides access to Domino databases.</td>
</tr>
<tr>
<td>NotesView</td>
<td>Provides access to a Domino view which is a formatted list of documents.</td>
</tr>
<tr>
<td>NotesDocument</td>
<td>Represents a Domino document (record).</td>
</tr>
<tr>
<td>NotesRichTextItem</td>
<td>Used to access Domino rich text fields.</td>
</tr>
</tbody>
</table>

### Summary

In this chapter we have walked through examples of using Microsoft Word and Microsoft Excel together with Domino. We have shown how you can get information from a Domino database and use it in your MS Office applications and we have shown how you can store your documents in a Domino database to share them easily with your coworkers.
Chapter 4
Using Visual Basic

This chapter shows an example of accessing a Domino database in a Visual Basic (VB) application. Access to Domino from within Visual Basic is very easy. Once the Lotus Domino Objects have been identified as a reference to Visual Basic, you can browse the Domino Objects in the Object Browser, and you will be able to take advantage of the Automatic Code Completion feature in VB.

For this example, we will build a Single Document Interface, which will essentially be used as a window into a Domino Document Library to view the review status of a given category of documents in the database.

Note  The Domino Document Library is an electronic filing cabinet that allows you to store any kind of document for access and review by a work group. Documents could be expense statements, appraisals, marketing materials, graphics, legal documents, or any type of data that can be represented electronically.

This chapter is primarily targeted toward Visual Basic programmers. Therefore, we will not be going into much detail about Visual Basic programming, except to show how to use the COM interface to Domino. While the Visual Basic COM interface to Domino is similar to LotusScript, LotusScript programmers might
want to read through the last section in this chapter, “Differences between accessing Domino through COM in Visual Basic and LotusScript,” to get a better understanding of those differences. Then come back and look through the rest of the chapter to see some examples of those differences.

We will be using the Domino database we created in Chapter 3 in the section “Accessing Domino databases from MS Excel using COM and VBA.” You do not need to have worked through that example, but you will need a Domino database based on the Doc Library Notes & Web (R5.0) template somewhere in your Domino data directory path. If you need help creating this database, refer to the section “Create a Document Library database in the Domino Client” in Chapter 3. The database may have any name as long as it is stored somewhere in the Domino directory structure. Once you have created the database, launch it in the Notes client and create a few sample documents, making sure to include a subject and category, then submit at least a few for review. It is not necessary to actually embed or attach any external documents for the example to work.

In addition to the Domino database, you will also need:

- Visual Basic 6.0 (Visual Basic 97 should work, but will look slightly different)
- The Domino server or Notes client, R 5.02b or later, installed on your local machine

**Note** If you have upgraded Notes or Domino from an earlier release using the incremental upgrade installer you may have to enable the COM support manually. Do this by selecting Run on the Windows Start menu and then execute the following command (substituting c:\lotus\notes with the path where your Notes Domino files are installed):

```
regsvr32 c:\lotus\notes\nlsxbe.dll
```

**Create the Visual Basic program**

For this example, we will describe the following steps in detail:

- Create a new project.
- Configure the form objects we will need.
- Define the global variables and types.
- Create the sub GetDocLibDbs, which will retrieve the databases for the Combo1 combo box.
- Create the function ViewCategories, which will return a list of categories in the Domino Document Library.
• Create the sub PopulateStatusList, which will retrieve the document information needed to populate ListView1.

• Create the function StatusText, which will translate the numeric status codes from the Domino Document Library to textual status information.

• Add code to the form events Form_Load, Combo1_Click, and Combo2_Click to populate the form objects with data.

Create a new Visual Basic project

We will use the Visual Basic Application Wizard to create all the code needed to set up the basics of an application.

1. Launch Visual Basic 6.0 and create a new project using the VB Application Wizard. You will find the VB Application Wizard in the New Project dialog box. Select it and click OK, which will start the application wizard.

2. Choose (None) for the Profile. Click Next.

3. Choose Single Document Interface and name your project. We are going to call ours DomDocLibWin. Click Next.

4. For menus, deselect everything but the File menu, and the Exit submenu. Click Next.

5. We will not be using the toolbar in this application, so we will take everything off of it. Remove everything by clicking on the button with the two left arrowheads :ś.

6. We want to accept the defaults for the rest of the screens, so click Finish (instead of Next). Your project should now look similar to the following figure. You may see a dialog box that confirms that the project has been created; if so, dismiss it by clicking OK.
Configure the form objects

1. Make the form about a third wider and a little longer. Add two combo boxes across the top of the form and label them “Document Library Database” and “Category.” Your screen should look similar to this:

2. Change the Style for both combo boxes to “2-Drop Down List.”

3. Add a large ListView in the remaining space on the screen. Your form should now look like this:

58 COM Together — with Domino
4. Change the properties for ListView1 as follows:
   - FullRowSelect: True
   - GridLines: True
   - View: 3 - lvwReport
5. Right-click on ListView1 and choose Properties.
6. Select the Column Headers tab and insert the columns listed below. Aside from the column headers, leave everything else at the default values. Insert these four columns:
   1. Subject
   2. Submitted By
   3. Reviewer
   4. Status

You are now finished creating the user interface for your application. Next we will provide the code to make it work.

**Activate the Lotus Domino Objects**

To have access to the Lotus Domino objects, you will need to activate the Lotus Domino Objects library under Project - References on the VB menu. You should find the Lotus Domino Objects listed alphabetically under available references. Select it and click OK.

**Define the global variables and types**

Open the Module1 code module and add the following declarations to the (General) (Declarations) section.

**Note** All of our Domino object variables are prefixed by the “dom” identifier, which is done as a convention to provide clarity and improve code readability. It is not required, but is strongly recommended. In larger programs, you may want to use a scope prefix for variables as well. For further discussion of this convention, search for “Constant and Variable Naming Conventions” in the electronic Visual Basic documentation.

1. A public variable for storing our reference to Domino (NotesSession). We will use the New keyword to create a new instance of NotesSession:
   
   **Important** You only need, and may only have, one reference to NotesSession. Trying to initialize more than one NotesSession may cause your code to crash.

   ```vba
   Public domSession As New NotesSession
   ```

2. A public variable for storing the reference to the Domino Document Library database (NotesDatabase):

   ```vba
   Public domDocLib As NotesDatabase
   ```
3. A type definition for storing a list of Domino databases and their titles:

   Public Type DbList
   strDbName As String
   strDbTitle As String
   End Type

4. A public dynamic array, based on the DbList type we just created, to store a list of databases we retrieve:

   Public udtDocLibList() As DbList

5. A type definition for storing information about the documents in the Document Library database:

   Public Type DocLibStatus
   strSubject As String
   strSubmittedBy As String
   strReviewer As String
   strStatus As String
   End Type

That completes the global variables needed for the example. Now we will add the sub routines and functions we need.

Create the sub GetDocLibDbs

The sub GetDocLibDbs will retrieve the databases for the Combo1 combo box. Since the Domino Document Library database is created from one of the standard database designs, called “templates,” that ship with Domino, we can inspect the DesignTemplateName of the databases to retrieve any databases that were created with that template. We will place these databases in the udtDocLibList array that we created from our DbList type definition. This list of databases will be used to populate the Combo1 combo box when the form is loaded.

1. Add the sub GetDocLibDbs. Type the following in the code area:

   Sub GetDocLibDbs()

2. Declare variables:

   Dim domDbDir As NotesDbDirectory 'The NotesDbDirectory 'object will hold a directory of Domino Databases
   Dim domDb As NotesDatabase 'The NotesDatabase object 'provides a handle to a Domino database

   60 COM Together — with Domino
Dim intIndex As Integer 'Index for the array of Domino databases
Dim strTemplate As String 'will receive the name of the template used to create the database

3. Call the GetDbDirectory method of domSession (NotesSession) to retrieve the list of Domino Databases in the Domino data directory. The parameter represents the Domino server from which you want to retrieve the directory. In this case, we want to look at the local directory, so we pass an empty string.
   Set domDbDir = domSession.GetDbDirectory(""")

4. Call the GetFirstDatabase method of domDbDir (NotesDbDirectory), passing it the type of Database you want to retrieve. The option constants are:

<table>
<thead>
<tr>
<th>Type of database</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any Notes/Domino database (.nsf, .nsg, or .nsh file)</td>
<td>DATABASE or NOTES_DATABASE</td>
</tr>
<tr>
<td>Any Notes/Domino database template (.ntf file)</td>
<td>TEMPLATE</td>
</tr>
<tr>
<td>Any Notes/Domino database or template not disabled for replication</td>
<td>REPLICA_CANDIDATE</td>
</tr>
<tr>
<td>Any Notes database or template</td>
<td>TEMPLATE_CANDIDATE</td>
</tr>
</tbody>
</table>

Note The VB editor will automatically convert the constants to proper case, which does not affect their meaning.

   Set domDb = domDbDir.GetFirstDatabase(Database)

5. Now we will enter a while loop to traverse all the databases in domDbDir. As we mentioned earlier, we will check the DesignTemplateName property to see if the database was created using the StdR50WebDocLib template. If it was, we will increase the udtDocLibList array by one element and store the database file path from the FilePath property of domDb and the database title from the Title property. We will use the GetNextDatabase property to get the next database in the list, repeating until we have exhausted the list (domDb is Nothing).

   While Not (domDb Is Nothing)
   
   strTemplate = domDb.DesignTemplateName
   
   If strTemplate = "StdR50WebDocLib" Then
   
   ReDim Preserve udtDocLibList(intIndex)
   
   udtDocLibList(intIndex).strDbName = domDb.FilePath
   
   udtDocLibList(intIndex).strDbTitle = domDb.Title

   End If

   domDb = domDbDir.GetNextDatabase(Database)

   End While
intIndex = intIndex + 1
End If
Set domDb = domDbDir.GetNextDatabase
Wend

Create the function ViewCategories

We need a function to return a list of categories from the Domino Document Library to be displayed in the Combo2 combo box, once we know which Document Library database the user is interested in. The Combo1 click event is a good place to call such a function because it will be triggered as soon as the user makes a choice of databases. We decided to pass the database name and view name to the function for reusability. Here is how to set up this function:

1. Add a new function, “ViewCategories” with the arguments strDb and strView:

   Public Function ViewCategories(strDb As String, strView As String) As Variant

2. Declare the necessary variables. The NotesDatabase object provides us with a handle to the Domino database. The NotesView object will represent the Notes view that is a formatted list of documents (records) meeting certain criteria. A Notes view navigator is similar to a view, but gives us access to other elements of the view like category headings and totals. The Notes view entry represents a single entry in the view navigator, and could be a document, a category, or a total. The strCategories dynamic array will be the receptacle for the categories we retrieve, and intInteger will be the index for the array.

   Dim domDb As NotesDatabase
   Dim domView As NotesView
   Dim domViewNav As NotesViewNavigator
   Dim domViewEntry As NotesViewEntry
   Dim strCategories() As String
   Dim intIndex As Integer

3. Now we will call the GetDatabase method of domSession, passing it the server, which is blank because we are using a local database, and the path and name of the database passed to us as strDb from the calling procedure:

   Set domDb = domSession.GetDatabase(“", strDb)

4. We will get domView by calling the GetView method of domDb and passing it the name of the view we were passed from the calling procedure:

   Set domView = domDb.GetView(strView)
5.  domViewNav gets populated by calling the CreateViewNav method of domView:

\[
\text{Set domViewNav = domView.CreateViewNav}
\]

6.  domViewEntry comes from the GetFirst method of domViewNav, which returns the first entry in the view navigator:

\[
\text{Set domViewEntry = domViewNav.GetFirst}
\]

7.  Now we enter our while loop to retrieve all of the categories from the view navigator by calling the GetNextCategory method of domViewNav, passing it the current view entry, domViewEntry, as a reference point. TheColumnValues property of domViewEntry returns information for the column indicated.

   Tip    Notice that we saved the first view entry’s first column to the category array. We knew it would be a category because, if a Notes view is categorized, the first entry is always a category.

\[
\text{While Not (domViewEntry Is Nothing)} \quad \\
\quad \quad \text{ReDim Preserve strCategories(intIndex)} \quad \\
\quad \quad \text{strCategories(intIndex) = domViewEntry.ColumnValues(0)} \quad \\
\quad \quad \text{Set domViewEntry = _} \quad \\
\quad \quad \text{domViewNav.GetNextCategory(domViewEntry)} \quad \\
\quad \quad \text{intIndex = intIndex + 1} \quad \\
\text{Wend}
\]

8.  Finally, we return strCategories to the calling procedure:

\[
\text{ViewCategories = strCategories}
\]

**Create the sub PopulateStatusList**

Once the user has chosen a database and the category of document for which they wish to view status information, we need a procedure to retrieve the documents that match the user criteria and populate the ListView (ListView1) that will display the data.

This sub, PopulateStatusList, will again use a view navigator which can be created for a single category in a view. Using the category chosen by the user, we will create a view navigator and use a while loop to go through all the documents in the view navigator. As in the ViewCategories function, we will use the view entry object. Unfortunately, it only gives us access to the information displayed in the view, which in this case does not have everything we will need, like Subject and Status, so we will also use the Document property of the view entry to gain access to the complete document.
1. Create a new sub called PopulateStatusList with arguments for udtStatusList, strDb, and strCategory:

   ```vba
   Public Sub PopulateStatusList(udtStatusList() As DocLibStatus, strDb As String, strCategory As String)
   ```

2. Declare the following variables. All of them should be familiar to you except domName, which is a NotesName object that is used to represent the name associated with a Notes user ID. It gives us the flexibility of representing a Notes user name in several different ways, including the more friendly Common Name version which shows the user name without any hierarchy references.

   ```vba
   Dim domDb As NotesDatabase
   Dim domView As NotesView
   Dim domViewNav As NotesViewNavigator
   Dim domViewEntry As NotesViewEntry
   Dim domDoc As NotesDocument
   Dim domName As NotesName
   Dim intIndex As Integer
   Dim strSubmitted As String
   Dim intStatus As Integer
   Dim strTemp as String
   ```

3. Initialize the Domino variables:

   ```vba
   Set domDb = domSession.GetDatabase(“", strDb)
   Set domView = domDb.GetView(“By Category“)
   Set domViewNav = _
   domView.CreateViewNavFromCategory(strCategory)
   Set domViewEntry = domViewNav.GetFirstDocument
   ```

   These should be familiar to you now, except for the last two:
   - The CreateViewNavFromCategory method of domView limits a view navigator to showing only the specified category.
   - The GetFirstDocument method of domViewNav gets the first document in the view; as opposed to the GetFirstEntry method we used before, which could be either a document, category, or total, depending on the view.

4. Most of the processing takes place in the while loop. Let us examine the details:

   The while loop will execute until we reach the end of domView (domViewEntry Is Nothing), collecting the information we need from the documents in the view navigator, skipping those that are response documents by checking the IsResponse property of domDoc. In Domino,
Response documents are “sub” documents that typically contain comments or prior versions of the main document.

```
While Not (domViewEntry Is Nothing)
    Set domDoc = domViewEntry.Document
    If Not domDoc.IsResponse Then
        ReDim Preserve udtStatusList(intIndex)
    We call the GetItemValue method of domDoc, passing the name of the item (field), which returns the data as a variant array.
    udtStatusList(intIndex).strSubject = domDoc.GetItemValue("Subject")(0)
    Tip The GetItemValue method always returns a variant array because any Domino item can be multi-valued. If you are reasonably sure you are dealing with a single value, or only want the first value, you can return just that value by accessing the 0 index as we have done here. Otherwise, you could assign the result to a variant to access the array, or you might explore the NotesItem data type which provides other possibilities.
    5. Notice that we are checking for the existence of the “Submitted” and “Status” items before trying to access them. This is to avoid an error condition caused when these items don’t exist in the document, which will be the case if the document is not using the Document Library review process.
    If domDoc.HasItem("Submitted") Then
        strSubmitted = domDoc.GetItemValue("Submitted")(0)
    Else
        strSubmitted = ""
    End If
    If domDoc.HasItem("Status") Then
        intStatus = domDoc.GetItemValue("Status")(0)
    Else
        intStatus = 0
    End If
```

In this last section of code, we use the CreateName method of domSession to create the value for the domName variable we spoke about earlier in our declarations. We also use the StatusText function that we will show in the next section to translate strSubmitted and intStatus into meaningful text.

**Note** We used a temporary variable strTemp to first assign the result of domDoc.GetItemValue("CurrentEditor")(0) so that the example code would fit on one line for this book; it is not necessary to take this extra step and you could code it like the Set domName line.
.udtStatusList(intIndex).strStatus = _
StatusText(strSubmitted, intStatus) 

strTemp = domDoc.GetItemValue("CurrentEditor")(0) 
Set domName = domSession.CreateName(strTemp) 
.udtStatusList(intIndex).strReviewer = domName.Common 
Set domName = _
domSession.CreateName(domDoc.GetItemValue("From"))(0)) 
.udtStatusList(intIndex).strSubmittedBy = domName.Common 
intIndex = intIndex + 1 
End If 
Set domViewEntry = domViewNav.GetNextDocument(domViewEntry) 
Wend 
End Sub

Create the StatusText function

This is a fairly simple function that simply assigns a meaningful text status based on the value of strSubmitted and intStatus:

Public Function StatusText(strSubmitted As String, strStatus As Integer) As String
    If strSubmitted <> "1" Then 
        StatusText = "Not Submitted"
    ElseIf strStatus = "1" Then 
        StatusText = "New"
    ElseIf strStatus = "2" Then 
        StatusText = "In Review"
    ElseIf strStatus = "3" Then 
        StatusText = "Complete"
    Else 
        StatusText = "No Status"
    End If
End Function

66 COM Together — with Domino
Add code to the form events to bring the form to life

Now that we have all the working procedures we need, we can add the event triggers we need to actually have the form make use of them. We will need to add code to Form_Load, Combo1_Click, and Combo2_Click.

**Note** Right click on frmMain in the project box and choose view code to bring up the code window for the form.

**Form_Load**

1. Because calling the Initialize method of domSession again after it has been initialized can cause our code to crash, we will add a declaration for strTestDomSession which we will use to try to return a string property of domSession. If domSession hasn’t been initialized, we will test the error generated in our error handler and initialize the session there.

   ```vba
   Dim strTestDomSession As String
   ```

2. Declare an integer to use as a counter and index to load all of the databases collected in the udtDocLibList by GetDocLibDbs.

   ```vba
   Dim lngCounter As Long
   ```

3. Add the On error statement and the test of the domSession property:

   ```vba
   On Error GoTo ErrorHandler

   strTestDomSession = domSession.CommonUserName
   ```

4. Call the GetDocLibDbs sub and add each of the databases to the Combo1 combo box:

   ```vba
   Call GetDocLibDbs

   For lngCounter = 0 To UBound(udtDocLibList)
   Call Combo1.AddItem(
       udtDocLibList(lngCounter).strDbTitle, lngCounter)
   Next
   ```

5. Finally, add the error handler which will call the Initialize method of domSession.

   **Note** You may optionally pass a password to the Initialize method to avoid getting a password prompt from Domino: domSession.Initialize(“password”).

   ```vba
   GoTo Done
   ```

   ```vba
   ErrorHandler:
   ```

   ```vba
   If Err = ERR_NOTES_SESSION_NOT_INIT Then
   domSession.Initialize
   Resume Next
   ```
Else
    MsgBox ("Run-time error " & Str(Err) & ":" & vbCrLf & Err.Description)
End If

Done:

**Combo1_Click**
1. We need to declare a variant to store the returned categories from the ViewCategories function and a temporary string variable.
   ```vBA
   Dim vntCats As Variant
   Dim strTemp As String
   ```
2. Clear the Combo2 combo box in case the user has selected another database in Combo1 and there are already categories listed in Combo2.
   ```vBA
   Combo2.Clear
   ```
3. Call the ViewCategories function, passing the database path and name based on the user selection and the name of the view from which we wish to retrieve categories.
   ```vBA
   strTemp = udtDocLibList(Combo1.ListIndex).strDbName
   vntCats = ViewCategories(strTemp, "By Category")
   ```
4. Iterate through the categories and add them to the Combo2 combo box.
   ```vBA
   For Each category In vntCats
       Combo2.AddItem (category)
   Next
   ```

**Combo2_Click**
1. Declare udtStatusList, based on the DocLibStatus type we defined; declare lngIndex for iterating through the udtStatusList created by PopulateStatusList; declare a ListItem, itmS, to use for populating the ListView1 list items:
   ```vBA
   Dim udtStatusList() As DocLibStatus
   Dim lngIndex As Long
   Dim itmS As ListItem
   ```
2. Clear the ListView1 list box in case it was previously populated.
   `ListView1.ListItems.Clear`

3. Call the PopulateStatusList sub, passing the database path and name based on the user selection from Combo1, and the category the user selected from Combo2.
   ```vba
   Call PopulateStatusList(udtStatusList, _
   udtDocLibList(Combo1.ListIndex).strDbName, Combo2.Text)
   ```

4. Iterate through udtStatusList to populate the ListView1 list box.
   ```vba
   For lngIndex = 0 To UBound(udtStatusList)
   Set itmS = ListView1.ListItems.Add(, , _
   udtStatusList(lngIndex).strSubject)
   itmS.SubItems(1) = udtStatusList(lngIndex).strSubmittedBy
   itmS.SubItems(2) = udtStatusList(lngIndex).strReviewer
   itmS.SubItems(3) = udtStatusList(lngIndex).strStatus
   Next
   ```

The finished example
Summary of Lotus Domino Object classes used in this example

<table>
<thead>
<tr>
<th>Class</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotesSession</td>
<td>Required before accessing any other Lotus Domino Objects. Don’t initialize more than once for a program or your program may crash.</td>
</tr>
<tr>
<td>NotesDatabase</td>
<td>Provides access to Domino databases.</td>
</tr>
<tr>
<td>NotesDbDirectory</td>
<td>Provides access to a list of databases in the Domino data directory.</td>
</tr>
<tr>
<td>NotesView</td>
<td>Provides access to a Domino view, which is a formatted list of documents.</td>
</tr>
<tr>
<td>NotesViewNavigator</td>
<td>Provides access to the rows of a view, including categories, titles, and documents.</td>
</tr>
<tr>
<td>NotesViewEntry</td>
<td>Represents one row in a view navigator.</td>
</tr>
<tr>
<td>NotesDocument</td>
<td>Represents a Domino document (record).</td>
</tr>
<tr>
<td>NotesName</td>
<td>Used to access different formats of the Domino user name; for example, common, hierarchical, first name.</td>
</tr>
</tbody>
</table>

Differences between accessing Domino through COM in Visual Basic and LotusScript

For the most part, there are very few differences between accessing Domino objects through COM in Visual Basic (VB) and through the LotusScript (LS) Domino objects. Here are some of the key differences:

- Initializing the NotesSession. In LS and VB, you can declare a new NotesSession with the same syntax:

  ```vbnet
  Dim session as New NotesSession
  ```

  In VB, however, you must add an additional statement to initialize the NotesSession, essentially, logging in to Domino:

  ```vbnet
  Call session.Initialize
  Or:
  Call session.Initialize("password")
  Or:
  Call session.InitializeUsingNotesUserName("name", "pw")
  ```
The first format will result in a password prompt from Domino for the current Notes ID specified in NOTES.INI on the workstation. The second format will bypass the password prompt for the current Notes user by sending the password as a parameter. This is unnecessary in LS because you have already identified yourself to Domino. The third format can only be used on a Domino server. It allows you to authenticate as a Web user registered in the Domino Directory on the server.

- Notice that we used the `New` keyword when declaring `NotesSession`. With COM, `NotesSession` is the only class which allows the `New` keyword. Because of this, you will have to work through the other methods and properties available to you that return Domino objects. For example, instead of:

  ```vbscript
  Dim db As New NotesDatabase("server","filename")
  Dim name As New NotesName( "first last/ou/o" )
  Dim doc As New NotesDocument(db)
  ```

  Use:

  ```vbscript
  Dim db As NotesDatabase
  Set db = session.GetDatabase("server","filename")
  Dim name As NotesName
  Set name = session.CreateName("first last/ou/o")
  Dim doc As NotesDocument
  Set doc = db.CreateDocument
  ```

- COM access to Domino is only available for the Domino back-end classes. It does not support any of the front end or user interface classes, which include Button, Field, Navigator, NotesTimer, NotesUIDatabase, NotesUIDocument, NotesUIView, NotesUIWorkspace.

- COM does support handling items as extended properties. An easy way to access a database item in LS is by treating it as a `NotesDocument` property. For example, the following LS sets the value of the Subject item on a document to “Hello”:

  ```vbscript
  doc.Subject = "Hello"
  ```

  You cannot do this in VB. You have to code it like this:

  ```vbscript
  doc.ReplaceItemValue( "Subject","Hello" )
  ```

  and instead of using this line to get the content of the Address field:

  ```vbscript
  doc.Address(0)
  ```

  use this line in Visual Basic:

  ```vbscript
  doc.GetItemValue("Address")(0)
  ```
• The Domino COM support does not have a current environment. Thus the following methods and properties are not supported when using COM: UnprocessedDocuments, UnprocessedFTSearch, UnprocessedSearch in NotesDatabase. Also, CurrentAgent, CurrentDatabase and DocumentContext are not supported in NotesSession.

• VB does not support the OnError with an error number as a parameter, so you will have to check for specific errors in the error handler and deal with them there instead of having different error handlers for each kind of error, as is possible in LS.

For more information on the differences between accessing Domino objects through COM in VB and through the LS Domino objects, refer to the section “Accessing the Domino Objects through COM” in the Domino Designer R5.0.3 online help or the release notes for Domino R5.0.2b.

---

**Summary**

In this chapter we went through an example of accessing Domino from Visual Basic. We drilled down from a list of all available databases, through those based on a common template to the one we selected to work with. In that database we continued to drill down by getting a list of all categories used in the database and then showing all expense reports and some of their properties under the category we selected. Finally, we discussed differences in accessing Domino objects from Visual Basic and from LotusScript.
Chapter 5
Using VBScript from Active Server Pages

Active Server Pages (ASP) are used to make dynamic Web pages on Microsoft’s Internet Information Server (IIS). This is done by using a script language. Usually VBScript or JScript/JavaScript is used, but it is possible to plug in any script engine, as a COM object, to IIS.

In this chapter we do the following:

• Describe how to set up the Domino server to allow COM connections
• Explain how to make the COM connection to Domino in ASP
• Walk through a simple ASP example with authenticated access to data stored in Domino

Prerequisites

• MS IIS or MS Personal Web Server. As these are considered the same in this context, we from now on just refer to IIS.

• Lotus Domino server version 5.0.2b (or above).

  Note  If you have upgraded Notes or Domino from an earlier release using the incremental upgrade installer you may have to enable the COM support manually. Do this by selecting Run on the Windows Start menu and then execute the following command (substituting c:\lotus\notes with the path where your Notes Domino files are installed):

regsvr32 c:\lotus\notes\nlsxbe.dll

• A Web browser.
All programs could be run on the same machine for development and testing purposes. In real life we would have the IIS and Domino server running on a server machine and the Web browser on client machines. The following figure shows the relationship between the elements.

![Diagram showing the relationship between the elements: Client machine (Web browser) communicates via HTTP with the IIS on the server machine, and the IIS accesses the Domino server through the COM interface.]

The client machine communicates via HTTP with the IIS on the server machine. The IIS accesses the Domino server through the COM interface.

## Setup

To be able to access the Domino server through the COM interface, we must make sure that the security settings for the Domino server are appropriate. Otherwise the initialization of the COM interface may fail.

**Tip** To make things work on a machine that has both the Domino server and Notes client installed, the InprocServer32 sub key setting in the registry keys listed below must refer to the nlxsb.exe.dll in your Domino server program directory and not the Notes client program directory, as we can see in this figure:

![Registry Editor window showing the InprocServer32 sub key with the correct path to nlxsb.exe.dll in the Domino server program directory.]

74 COM Together — with Domino
The keys to look for in the registry are:

- \HKEY_CLASSES_ROOT\CLSID\{29131539-2EED-1069-BF5D-00DD011186B7}\n- \HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{29131539-2EED-1069-BF5D-00DD011186B7}\n
**Domino server security**

The COM interface uses the standard access mechanism for the Domino server. Security settings for the Domino servers are located in the Domino Directory. The Domino Directory holds configuration information for our Domino site. This is where we define users, groups of users, servers and network connections, among other things. If users do not have proper access to the Domino server, the initialization of the COM interface will fail.

To alter the access rights for the Domino server use the following steps:

1. Open the Domino Directory.
2. Open the document for the correct server.
4. Make sure the fields Access server and Not access server in the Server Access section list the right people and groups.

**Tip** For development purposes leave both fields blank. That will allow everyone access to the server.

5. Make sure the fields in the section Java/COM Restrictions list the right people and groups for access to execute COM calls.

**Tip** If you are using your own dedicated Domino server for development, you can leave an asterisk in both fields. That will allow everyone access through the COM interface.
Making the connection

We connect to Domino from ASP, through the COM interface, by creating an
instance of the Domino COM object. This object has methods and properties
which enable us to get to all of the objects and data in the Domino back end.

Creating an instance of the Domino COM object

During installation, the Domino server is registered as a COM server object
in the Windows Registry with the name:

(PROGID) Lotus.NotesSession

and the number:

(CLSID) {29131539-2EED-1069-BF5D-00DD011186B7}

These two IDs are used as a reference when we create an instance of the
Domino COM object.

We create an instance of the Domino COM object in one of two ways:

- Use the Server.CreateObject method using script commands
- Use the <OBJECT> HTML tag

If we use the Server.CreateObject method to create an instance of the Domino
COM object, we need to supply the registered PROGID as a parameter like this:

Dim nSession
Set nSession = Server.CreateObject(“Lotus.NotesSession”)

The other way to create an instance of the Domino COM object is by using
the <OBJECT> tag. We then can use either the registered PROGID or the
registered CLSID like this:

<OBJECT RUNAT=Server ID=nSession PRODID=“Lotus.NotesSession”>
</OBJECT>

Either way, the nSession object now holds a reference to the Domino COM
object which represents the back-end Domino Object Model.

Initializing the session

To access other objects in the Domino Object Model we need to call
initialize on the session object. This can be done using one of two methods:

- Initialize
- InitializeUsingNotesUserName

76 COM Together — with Domino
Initialize
The simple version of initialize uses the current user ID to authenticate.

```
call nSession.Initialize()
```

The current user ID file can be found on the line starting with KeyFilename=
in the NOTES.INI file on the Domino server. Initialize will also take
password as a parameter, but Domino server IDs are seldom password protected.

```
call nSession.Initialize("secretpassword")
```

**Caution** The IIS will go into a lock if the current user ID requires a
password and none is supplied. We then need to restart it.

With the session object initialized we now can get to all of what the Domino
back end has to offer. We can browse through databases on the server and
get to views, documents, and so on. We can even get to other Domino
servers and their databases. The usual Domino security applies though, and
we can only get to the things we have access to under the current user ID.
With that in mind, we would want users to log in as actual users and not as
the server.

InitializeUsingNotesUserName
For this we use the method InitializeUsingNotesUserName, which takes
both the Notes user name and Internet password as parameters. The Notes
user name and the Internet password for a user are defined in the Domino
Directory.

```
call nSession.InitializeUsingNotesUserName("Carl Tuesday", "password")
```

The authentication is done by comparing the user name parameter against
the user names listed in the Domino Directory. If no password parameter is
sent we get access as the entry for which we have a match. If the password
parameter is sent, this is checked against the Internet password specified for
the matching person. Either way we only get access to the current Domino
server. We can not open databases on other Domino servers.

**Caution** If the authentication is done to get access to protected data in
Domino, as opposed to personal settings (for example, if the user wants
frames or not), you should force the use of a password parameter when
calling the InitializeUsingNotesUserName method. Otherwise, the users
will be able to access protected data just by supplying a user name.

**Tip** To disable the possibility to authenticate without a password, add the
following line to the NOTES.INI file on the Domino server.

```
DisableComNP = 1
```

Chapter 5: Using VBScript from Active Server Pages  77
The Web server authentication setting for the Domino server indicates how the lookup based on the user name parameter is performed. This setting is in the Web Server Access section of the server document in the Domino Directory. We have two options:

- Fewer name variations with higher security
- More name variations with lower security (the default)

Selecting the option Fewer name variations with higher security allows users to authenticate by supplying either:

- full hierarchical name
- common name
- any alias names in User name field in their Domino Directory document

Selecting More name variations with lower security will allow user to authenticate by supplying, in addition to the names on the previous list:

- last name only
- first name only
- short name
- soundex value (The soundex value is an output based on a phonetic algorithm that offers different variants of the specified name.)

**Important** When using a Domino server upgraded from R4.x, the Domino Directory in use must be based on the R5.x template to use the selection Fewer name variations with higher security.

---

**Developer tools**

We can use either a text editor like Notepad or a more sophisticated tool, like MS Visual InterDev, for development. More advanced tools will help us set up and manage the project, and provide us with online documentation. For small projects it might be sufficient to use a simple text editor and look up documentation from the Web.

**Tip** See the Tools and Technology folder on the following Microsoft Web site for more information about ASP.

MS Visual InterDev

MS Visual InterDev is a Web development tool which can be purchased from Microsoft. It provides an integrated development environment for building Web applications and lets you visually design HTML pages using drag and drop techniques. The editor provides support for scripting in the sense that it highlights reserved words from the script language.

IIS setup for developing with MS Visual InterDev

To be able to access IIS from the MS Visual InterDev remotely, we need to enable server extensions on the IIS. This is done as follows:

1. Start up the Internet Services Manager. (Run inetmgr.exe)
2. Locate the Default Web Site folder under your machine name in the tree view on the left.
3. Right-click this entry and select All Tasks - Configure Server Extensions from the pop-up menu.

4. Follow the instructions in the Configuration wizard to complete the installation.

**Tip**  To get more information on the IIS, type the following URL in your Web browser on the IIS server machine:

http://localhost/iishelp

From a client machine use the server name in the URL, like this:

http://odin.lotus.com/iishelp/

---

**A simple ASP example**

We now present an ASP example showing how to access the Domino Object Model using VBScript and the session object described previously. We can run this on a single machine configured as outlined in the Setup section, and of course we need a Web browser.

The example is called COMpact mail and shows the basic steps for accessing Domino elements and data through COM from ASP.
First, users are presented with a page where they supply a user name and password.

When they click a submit button they are presented with the five most recent incoming messages in their mailbox.

This requires that the user is properly registered in the Domino Directory and that the user’s mailbox resides locally on the server, since we can not access other Domino servers. The path and filename for the user’s mailbox is read from the person document representing the user in the Domino Directory.
Building the example

Tip  If you do not have MS Visual InterDev, use a text editor and do the following:

1. Put the code in a text file with the name COMpactmail.asp
2. Put this file under the directory Inetpub\wwwroot on the IIS machine.

Setting up the project in MS Visual InterDev:
1. Select File - New Project in the MS Visual InterDev editor.
2. Give the project a name and supply the path where you want the project files. We used Compactmail in C:\Visual Studio Projects\Compactmail.
3. Click Open.
4. Supply the name of your server; using localhost will do if you have IIS locally on your machine.

5. Click Next.

6. Give the application a name. We used Compactmail.

7. We will use the default values for the rest of this project. Click Finish instead of Next and wait for the wizard to finish. We now have a project ready to use.
8. To add an Active Server Page to the project select Project - Add Web Item - Active Server Page from the menu.

9. Supply a name for the page. We used Compactmail.asp.

10. Click Open and you are ready to type in the code.

**Type libraries**

The new IIS 5.0, which ships with Windows 2000, is able to handle type libraries in much the same way we are used to in MS Visual Basic. Type libraries contain a description of the methods and properties that a COM object supports. They can also be used to provide constant definitions that we can use when calling the COM objects methods and properties. The following line uses NOTES_DATABASE as a constant instead of the numeric value 1247, to tell what type of file to get from the database directory.

```vbscript
Set NotesDB = DBDir.GetFirstDatabase(NOTES_DATABASE)
```

To include a type library in the project, including the following lines in the header section of the ASP file:

```vbscript
<!—METADATA TYPE="typelib"

FILE="c:\lotus\domino\domobj.tlb"—>
```

To make this setting global in a project with potentially more than one ASP file, we can place this line as the first line in the file Global.asa. Global.asa contains global settings like this and can also define actions for ASP session and application events. If we use Visual InterDev we can include the setting by using the following steps:

1. Choose Project - Project References from the menu.
2. Scroll down in the dialog box to the entry Lotus Domino Objects.
3. Select this item.

4. Click OK.
Now the system has included the setting into our Global.asa file.

The code

Here is a listing of the ASP code that makes up the example. The script parts are written in VBScript.

```
<%Option Explicit
Response.Expires = 0  'No cashing%>
<HTML>
<HEAD>
<TITLE>COMpact mail</TITLE>
</HEAD>
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
Dim sServer  ' String
Dim bLoggedIn  ' Boolean
Dim sUsername  ' String
Dim sPassword  ' String
Dim iNumMsgs  ' Integer
dim sCurrMsgNo  ' String
Dim sErrorMessage  ' String

Sub GetUserInput  ' Procedure for getting user input
```

Chapter 5: Using VBScript from Active Server Pages  85
sUsername = Trim(Request.Form("Username"))
sPassword = Trim(Request.Form("Password"))
End Sub

Sub GetMail ' Procedure that connect and prints the mail
Dim nSession ' NotesSession
Dim nDbDir ' NotesDatabaseDirectory
Dim nMailDb ' NotesDatabase
Dim nView ' NotesView
Dim nDoc ' NotesDocument
Dim nItem ' NotesItem
Dim nName ' NotesName
Dim i ' Integer
On Error Resume Next ' We handle the errors ourselves
bLoggedIn = false
set nSession = Server.CreateObject("Lotus.NotesSession")
If sUserName <> "" And sPassword <> "" Then
Call nSession.InitializeUsingNotesUserName(sUsername,_
    sPassword)
End If
If (sPassword = "") Then
    sErrorMessage = "You must provide a password"
ElseIf nSession.usernameobject Is Nothing Then
    sErrorMessage = "Could not validate username/password!"
Else
    bLoggedIn = True
End If
sServer = nSession.ServerName
Set nDbDir = nSession.GetDbDirectory(sServer)
Set nMailDb = nDbDir.OpenMailDatabase()
If (Not nMailDb.IsOpen) Then
Response.Write("Unable to open Your mailbox on " & sServer)
Else
Response.Write("<TABLE border=0 cellspacing=0 &_
"cellpadding=0 cols=1>")
Set nView = nMailDb.GetView("($Inbox)")
Set nDoc = nView.GetLastDocument()
if nDoc Is Nothing Then
    Response.Write("<B>No mail in mailbox!\</B>")
Else
    Response.Write("<TABLE border=0 cellspacing=1 &_
"cellpadding=1>")
    Response.Write("<TR><TD><B>Date</B></TD> &_
"<TD><B>From</B></TD> &_
"<TD><B>Subject</B></TD></TR>"
    Response.Write("</TABLE>")
For i=1 To 5
    If Not nDoc Is Nothing Then
        Set nName = nSession.CreateName(_
            nDoc.GetItemValue("From")(0))
        Response.Write("<B>Date</B></TD> &_
            "<TD><B>From</B></TD> &_
            "<TD><B>Subject</B></TD></TR>"
        Response.Write("</TABLE>")
        For i=1 To 5
            If Not nDoc Is Nothing Then
                Set nItem = nDoc.GetFirstItem("Body")
        Next i
    End If
Next i
If nItem.Text = "" Then
    Response.Write("<TR><TD><FONT size=2> [Empty body]" 
        "</FONT></TD></TR>")
Else
    Response.Write("<TR><TD><FONT size=2>" 
        Left(nItem.Text, 500) & "</FONT></TD></TR>")
End If
Set nDoc = nView.GetPrevDocument(nDoc)
End If
Next
End If
Response.Write("</TABLE>")
End If
End Sub
</SCRIPT>

<HTML>
<HEAD>
<TITLE>COMpact mail</TITLE>
</HEAD>
<BODY BGCOLOR="White" TEXT="Black">
<H2>COMpact mail</H2><HR noshade>
%sUserName <> "" Then ' Make sure we have a username
    GetMail ' call the procedure to get the mail
End If
If Not bLoggedIn Then ' if we did not log in, provide form %>
<form action="COMpactMail.asp" method="post" name="COMpactMail">
    <table border=0 cellspacing=0 cellpadding=2 cols=2>
        <tr>
            <td>Username:</td>
            <td><input type="text" name="Username" value="$sUserName">
        </tr>
    </table>
</form>
</BODY>
</HTML>
Comments on the code

We can split the code into two parts: the script part surrounded by the <SCRIPT> tag, and the HTML part surrounded by the <HTML> tag.

The HTML part
The HTML part consists of an HTML form and some inline script code.

HTML form
The HTML form enables the user to give a username and password as an input and provides a button to submit the page back to the server for processing.

```html
<FORM ACTION="COMpactMail.asp" METHOD="POST" NAME="COMpactMail">
   <TABLE border=0 cellspacing=0 cellpadding=2 cols=2>
      <TR>
         <TD>Username:</TD>
         <TD><INPUT type="text" name=Username value="<%=sUsername%>"></TD>
      </TR>
      <TR>
         <TD>Password:</TD>
         <TD><INPUT type="password" name=Password></TD>
      </TR>
   </TABLE>
   <%=sErrorMessage%>
   <BR><BR>
   <INPUT type="submit" value="Login" id=submit1 name=submit1>
</FORM>
```
Inline script code
Inline script code is put between <% and %> tags and is executed by the IIS before it returns HTML to the Web browser. Using this, we can send different chunks of HTML code to the Web browser depending on the value of functions and variables evaluated on the IIS. We can also use inline script code to display the value of these variables directly. The following inline script code in the form evaluates the value of the sErrorMessage variable so it is displayed on the page.

<%=sErrorMessage%>

We can put inline script code inside HTML tags to provide input fields with default values.

<INPUT type="text" name=Username value="<%=sUsername%>">

On top of the HTML part we have inline script code that calls a procedure to get the user input.

<%GetUserInput ' put the user input into variables

If the user name is provided we call a second procedure which prints the mail on screen.

If sUserName <> "" Then ' Make sure we have a user name
  GetMail ' call the procedure to get the mail
End If

Enclosing the HTML form we have inline script code to decide if the form should be displayed or not.

If not bLoggedIn Then ' if we did not log in, provide form %>
  ... form ...
<%End If%>

The script part
The script part of this sample code defines two procedures: GetUserInput and GetMail. Let's go through the code in each procedure.
GetUserInput
First we get the values from the fields in the HTML form and put them into script variables.

```vbscript
sUsername = Trim(Request.Form("Username"))
sPassword = Trim(Request.Form("Password"))
```

Request is a standard ASP object representing the request from the Web browser. It has properties like Form, ServerVariables, Cookies and so on. The Form property, we saw above, can be used to get values from input fields on the form.

After this procedure is run we have populated the script variables with the user name, password and Domino server name.

GetMail
First we want to make sure that we catch errors so we can provide error messages. We are then able to detect if errors have occurred.

```vbscript
On Error Resume Next 'We handle the errors ourselves
```

We have not logged in yet, so we set bLoggedIn to false. We then create an instance of the Lotus.NotesSession COM object.

```vbscript
bLoggedIn = false
Set nSession = Server.CreateObject("Lotus.NotesSession")
```

If the username and password variables have been set, we call initialize.

```vbscript
If sUserName <> "" And sPassword <> "" Then
   Call nSession.InitializeUsingNotesUserName(sUsername, sPassword)
End If
```

We test to see if an error has occurred. If not, we are logged in.

```vbscript
If (sPassword = "") Then
   sErrorMessage = "You must provide a password"
ElseIf nSession.usernameobject Is Nothing Then
   sErrorMessage = "Could not validate username/password!"
Else
   bLoggedIn = True
End If
```

Now we can use the session object to drill down into the Domino data store. First we get the name of the Domino server from the session object. Then we open the database directory of that Domino server. This object holds
references to all the databases and their location on the Domino server. We also have a method called OpenMailDatabase which does just what it says. It gets the authenticated user’s mailbox settings from the Domino Directory and opens the mailbox. This method of NotesDbDirectory is only available when we access Domino through COM.

```vba
sServer = nSession.ServerName
Set nDbDir = nSession.GetDbDirectory(sServer)
Set nMailDb = nDbDir.OpenMailDatabase()
```

We now must check that the mailbox really got opened. Remember, we are handling all the errors ourselves, in order to avoid having the server abort the script. If the mailbox cannot be opened, we should tell the user and end the script.

```vba
If (Not nMailDb.IsOpen) Then
    Response.Write("Unable to open Your mailbox on " & sServer)
End If
```

If the mailbox is open, we continue by getting the inbox. The inbox is where all incoming messages are stored in the Domino mailbox. We here use the GetView method, even if the ($Inbox) actually is a folder (but never mind that distinction if you’re not familiar with Domino). For now we can think of both folders and views as a table of documents.

```vba
Response.Write("<TABLE border=0 cellspacing=0 cellpadding=0 cols=1>")
Set nView = nMailDb.GetView("($Inbox)")
```

We start off by asking for the last document in the inbox which also is the most recent mail message.

```vba
Set nDoc = nView.GetLastDocument()
```

We make sure that we really got a document. If not the mailbox is empty.

```vba
If nDoc Is Nothing Then
    Response.Write("<B>No mail in mailbox!</B>")
End If
```

We send HTML code which puts the messages into a table for easier reading.

```vba
Response.Write("<TABLE border=0 cellspacing=1 cellpadding=1>"
Response.Write("<TR><TD><B>Date</B></TD><TD><B>From</B></TD><TD><B>Subject</B></TD></TR>")
Response.Write("</TABLE>"
```

We start to loop through the last five messages. For each message we want to print the delivered date, who it is from and the subject of the message.

```vba
For i=1 To 5
```

92 COM Together — with Domino
We always check that the object we are going to use is something. It could be that this user had fewer than five mail messages in the mailbox.

**If Not nDoc Is Nothing Then**

The item From in the mail message contains the name of the author. This is very likely presented in a hierarchical way. We just want the common name of the person and, therefore, we must provide a temporary NotesName object which we create using a method CreateName of the session object, using the value of the From item as the input parameter.

```
Set nName = nSession.CreateName( nDoc.GetItemValue("From")(0))
```

Now we can write all the data desired from the mail message.

```
Response.Write("<TR><TD><B>
& FormatDateTime(CDate(nDoc.GetItemValue("DeliveredDate")(0)), vbShortDate) & 
":<B></TD><TD><B>
& nName.Common & 
"--<B></TD><TD><B>
" & 
nDoc.GetItemValue("Subject")(0) & "</B></TD></TR>"
```

We also print the first 500 characters of the mail message. This is done by obtaining the body item of the message document using the GetFirstDocument method, and writing the 500 leftmost characters in the Text property of the item. If the Text property is empty, the body is empty.

```
Set nItem = nDoc.GetFirstItem("Body")
If nItem.Text = "" Then
    Response.Write("<TR><TD><FONT size=2> [Empty body]
</FONT></TD></TR>"
Else
    Response.Write("<TR><TD><FONT size=2>" & Left(nItem.Text, 500) & "</FONT></TD></TR>"
End If
```

We continue to traverse the inbox until the five most recent mail messages are written to the screen.

```
Set nDoc = nView.GetPrevDocument(nDoc)
End If
Next
```

**To run the example**

Start up the Web browser and open the URL that is appropriate for your server’s address. If the server’s address is odin.lotus.com, the URL is:

```
http://odin.lotus.com/COMpactMail.asp
```
Extending ASP with components

In the example above we did all the work directly in the script language. This is fine for experimenting and small procedures. If the procedures get large, however, we would probably want to make external components and put the code into methods. This will allow us to structure our page in a better way. The components will also make our code more easily reusable, and we can hide the complexity of large computations.

To include a component in ASP, we use the same procedure with which we made an instance of the Domino COM object above. Either use the script method Server.CreateObject, or the <OBJECT> HTML tag.

In Chapter 6 we will look at how to implement a component to be included in an ASP script, which will encapsulate a Domino user’s mailbox. The component will implement the connection to Domino through the COM interface and hide this from the script programmer. The component will provide simple methods which we can call and it will handle the details. For example, we can call the method SendMail and just supply three parameters:

- Comma-separated string with the names of the recipients
- String containing the subject of the message
- String containing the body of the message

<component>.SendMail(SendTo, Subject, Body)

The component will handle the creation of the mail document in the right mail database, and so on.

Summary

In this chapter we have looked at how to connect to the Domino server from ASP script using the COM interface. By doing this we get access to all that Domino has to offer. We then presented an example that logged us in to the Domino server and read out messages from our mailbox using only plain VBScript in ASP.
Chapter 6
Using Visual C++

The existence of the C++ language should be known to most programmers. It is an evolution of the C language with the concept of object-oriented programming added to it. Many programmers are devoted to C++ and do not think that programming in Visual Basic (VB), for example, is real work. Even though VB has come a long way in both features and efficiency, we can agree that you get even better control and execution speed for software using C++. Whether you need this control and are prepared to go the extra mile in terms of coding is up to you. This chapter will describe how to get to the Domino back-end through the COM interface using C++.

Prerequisites

We will be developing a COM component in C++ in this chapter. The COM component we build initializes the Domino COM interface on a Domino Server. We will include the component in an ASP script. The prerequisites for running the example are the following:

- Lotus Domino server version 5.0.2c (or above).
  
  **Note** If you have upgraded Notes or Domino from an earlier release using the incremental upgrade installer you may have to enable the COM support manually. Do this by selecting Run on the Windows Start menu and then execute the following command (substituting `c:\lotus\notes` with the path where your Notes Domino files are installed):

  ```
  regsvr32 c:\lotus\notes\nlsxbe.dll
  ```

- MS IIS or MS Personal Web Server. As these are considered the same in this context, we from now on just refer to IIS.

- A Web browser.

For development you will need the following:

- Domino and Notes Toolkit for COM (see Appendix A for instructions on how to get it)

- MS Visual C++ 6.0
**Note** You should be able to use Visual C++ 5.0, but there might be a need to apply minor differences.

All programs can be run on the same machine for testing purposes. In real life we would have the IIS and Domino server running on a server machine and the Web browser on client machines. The figure below shows the relationship between the elements.

![Diagram showing the relationship between client machine, server machine, Web browser, IIS, Domino server, ASP, and C++ component.](image)

The client machine uses HTTP to send requests to the IIS on the server machine. The IIS, which runs the ASP script where our component is embedded, accesses our C++ component through COM. The C++ component, in turn, accesses the Domino server through the COM interface.

**Tip** In this example we are creating a server component. If we change the initialization call of the Domino COM interface, we will be able to use this component on a client machine with the Notes client installed instead. We would then use the method Initialize to authenticate, using the current Notes user ID file. Refer to the section “Making the connection” in Chapter 5 for more information on authentication against the Domino server.

---

**Building the component**

To distinguish between the code that is of general purpose for building C++ components, and the code that is unique to our component, we will divide building the component into the following three parts:

- Make a project using the ATL COM Wizard in Visual C++.
- Include files to the project to enable the Domino COM connection.
- Build code for the DominoMail component.

**Note** ATL stands for Active Type Library. Later in the example we will use the STL, or Standard Template Library. A template library is a new and alternative way to include prebuilt functions in C++ programs, as opposed to class libraries. One nice thing about template libraries is that only a file
has to be included in our project; we do not have to link with a library. ATLS
do not offer any help in building a user interface, but are excellent to use in
server-side components to minimize both size and dependency.

**Make a project using the ATL COM Wizard**

Once in your Visual C++ environment, follow these steps:

1. Choose New from the File menu.
2. Select the ATL COM Wizard on the left and provide a name for the
   project. We used DominoObjExt as a project name.

   **Tip** This name will be the first part (before the dot) of the PROGID.

3. Click OK.
4. In the next dialog, we accept the default settings. Server type is Dynamic Link Library (DLL) and no check boxes are checked.

5. Click Finish, and the wizard will provide us with a new project. Your screen should now look like this:
By default, ATL projects exclude exception handling to reduce the size of the component. But we are using the Standard Template Library (STL) for string handling (see include statements later in this section), and STL requires exception handling. For this reason we need to alter the following project settings:

- Remove the /D " _ATL_MIN_CRT" statement in Project Options for the Release configuration.
- Select Enable exception handling in the C++ Language settings.

To avoid problems when compiling the include files, we want to alter the setting for precompiled headers to:

- Select Automatic use of precompiled headers.

To apply these changes use the following steps:

1. Right-click the project name in the left pane (you have to activate the FileView tab if that is not active).
2. Choose Settings from the pop-up menu.
3. Select the Win32 Unicode Release MinDependency entry in the Settings for combo box. This is the configuration we are going to use to build our release version of the DLL.

4. In the lower right text field, mark the setting `/D "_ATL_MIN_CRT"`.

5. Press the Delete button to erase this setting.

6. Now select the All Configurations entry in the Setting for combo box.

7. Activate the C/C++ tab in the right pane.
8. Select the C++ Language entry in the Category combo box.

9. Select Enable exception handling.
10. Select the Precompiled Headers entry in the Category combo box.

11. Select the option button Automatic use of precompiled headers and enter the name of a header file in the field Through header. We used stdafx.h.

12. Click OK.

Now we have the project set up for making a standard COM component. To be on the safe side you should choose Save All from the File menu.
Files to enable the Domino COM connection

Expanding the tree in the left FileView pane you should have the following:

To provide for the use of Domino COM we want to include the following three files:

- `domobj.idl` defines the Domino COM interface using the Interface Definition Language (IDL). IDL is a standard language for describing the interface of an object.
- `domobj_i.c` defines the actual numeric IDs used for the Domino COM classes. This file is generated from the `domobj.idl` file.
- `domobj.h` declares the Domino COM classes with their methods and properties. This file is also generated from the `domobj.idl` file.

**Note** These files are shipped with the Domino and Notes Toolkit for COM. The first file, `domobj.idl`, is optional, but nice to include since we then get to see the defined Domino COM classes with properties and methods in the IDE (the Integrated Development Environment/the Visual C++ environment).

The third file is a header file which we will include later in the project by using the following code in the proper place:

```
#include "domobj.h"
```
To include the domobj_i.c and the domobj.idl file, use the following steps:

1. Locate these files and copy them into the project directory. This is an optional step, done to simplify finding them later.

2. Right-click the project name in the left pane as we did earlier, but this time choose Add Files to Project from the pop-up menu.

3. Select the file domobj_i.c.

4. Click OK.
5. Again, right-click the project name and choose Add Files to Project from the pop-up menu.

6. Alter the Files of type Combo box to say Interface Definition Files (.odl;.idl).

7. Select the file domobj.idl.

![Image of Insert Files into Project dialog box]

8. Click OK.

**Tip** You can now switch to the ClassView tab in the left pane and expand the project classes to see the listing of all the Domino COM classes.

We are now finished setting up a standard project for making a COM component which is able to access the Domino COM interface. To be on the safe side you should choose Save All from the File menu again.

**Note** A similar set of files will be generated automatically for our project, which in turn can be used for including our component in C++ applications. These are called DominoObjExt.idl, DominoObjExt_i.c and DominoObjExt.h if you use the same filenames as we do.

**Code for the DominoMail component**

To make our DominoMail component we now add what is called an ATL object to the project. If you have not done so yet, switch to the ClassView in the left pane so the screen looks something like the next figure.
Follow these steps to make the ATL object:

1. Right-click the project name in the ClassView tab in the left pane.
2. Choose New ATL Object from the pop-up menu.
3. Select the default, which is Simple Object in the Objects window.

4. Click Next.

5. Enter a name for the object. We used DominoMail.

Tip This name will be the second part (after the dot) of the PROGID. If you used the same names as in our example the PROGID is DominoObjExt.Dominomail.

6. Click OK.
The skeleton of our object is now generated. We can expand the ClassView to see the classes in our project.

Next we want to add some code in the include file DominoMail.h to provide us with the right environment. Follow these steps to open the include file:

1. Right-click the CDominoMail entry in the ClassView pane, as shown in the next figure.
2. Choose Go to Declaration. This will open up the DominoMail.h file and let us make declarations for our object.

We want to include the file domobj.h to handle the Domino classes and we need to include a string class from the STL. Type in the following right below the other include statements:

```cpp
#include "domobj.h"
#include <string>
using namespace std;
```

**Note** This requires that we have copied the domobj.h file to the project directory. Otherwise, enter the proper path in the include statement. The last statement makes us able to use the string template without any prefixes.
Declaring internal properties
While we are in the DominoMail.h file, we include code to declare the object’s internal properties (private variables) as well. Scroll down to the bottom of the open DominoMail.h file and include the following code:

private:
CComBSTR currUserName; // The current username
CComBSTR currServerName; // The current servername
CComBSTR sMailDBTitle; // DBTitle of the mailbox
ISession* m_pSession; // Pointer to NotesSession
IDatabase* m_pDatabase; // Pointer to NotesDatabase

The next steps will be to add methods and properties to our object.

Adding methods to the component
We want our component to have the following four methods:

- **Connect** method initializes the Domino COM interface and connects to the current user’s mailbox. It takes the username and password as input parameters, and uses a string as output parameter to report if something went wrong.

- **GetMail** method reads the five most recent incoming messages. It does not need any input, but we use a string as output parameter to supply the email messages in HTML, much as we did in chapter 5.

- **SendMail** method lets the user post mail to the Domino server. It takes a list of recipients, a subject and the message itself as input parameters, and we use a string as output parameter to tell if the message was posted.

- **Disconnect** — method to clean up and free resources.
Follow these steps to declare a method (we use the declaration of the Connect method as an example):

1. Right-click the IDominoMail entry in the ClassView pane as indicated below.

2. Enter a name for the method. We used Connect.

3. Type in the following in the parameters field:
   
   ```
   [in] BSTR sUsrName, [in] BSTR sPswd, [out, retval] BSTR *sMsg
   ```

4. Click OK.

Now we have the method Connect declared, and the wizard has even made us a skeleton for the definition. Expand the ClassView and see the entry for the method.
5. Right-click the entry for the method and choose Go to Definition from the pop-up menu.
6. Here we have the definition skeleton where we can put our code.

Now we can type in the actual code for the method beneath, or instead of, the comment:

// TODO: Add your implementation code here.

The Connect method
The Connect method has the following parameter list, as we saw previously:

[in] BSTR sUsrName, [in] BSTR sPswd, [out, retval] BSTR *sMsg

Here is the code for the Connect method:

STDMETHODIMP CDominoMail::Connect(BSTR sUsrName, BSTR sPswd, BSTR *sMsg)
{
    HRESULT hResult;
    IDbDirectory* m_pDbDirectory; // NotesDBDirectory pointer
    CComBSTR tmpMsg("");

    // Initialize the COM Library.
    if (FAILED(hResult = CoInitialize(NULL)))
    {

Chapter 6: Using Visual C++  113
tmpMsg = CComBSTR(  
    "Unable to initialize the COM library!";
}
else {
    if (FAILED(hResult = CoCreateInstance(
        CLSID_NotesSession,   // Class Identifier
        NULL,          // No Aggregate Object
        CLSCTX_INPROC_SERVER,  // Local In process
        IID_ISession,      // Interface
        (LPVOID*)&m_pSession))) // A pointer to the interface.
    {
        tmpMsg = CComBSTR(
            "Unable to Create a NotesSession Object!";
    }
else {
    // Initialize the NotesSession.
    if (FAILED(hResult = m_pSession->
                    InitializeUsingNotesUserName(sUsrName, sPswd)))
    {
        m_pSession->Release();
        tmpMsg = CComBSTR(
            "Unable to Initialize NotesSession!";
    }
else {
    m_pSession->get_ServerName (&currServerName);
    m_pSession->get_UserName (&currUserName);
    if (FAILED(hResult = m_pSession->GetDbDirectory(
                    currServerName, &m_pDbDirectory)))
    {
        tmpMsg = CComBSTR(
            "Unable to get NotesDBdirectory Object!";
    }
else {
    if (FAILED(hResult = m_pDbDirectory->
                    OpenMailDatabase(&m_pDatabase)))
    {

114 COM Together — with Domino
tmpMsg = CComBSTR("Unable to open mailbox!");
} else {
  m_pDatabase->get_Title(&sMailDBTitle);
  hResult = S_OK;
}
  m_pDbDirectory->Release();
}

*sMsg = tmpMsg.Detach();
return hResult;
}

The method starts by initializing the COM library, and if that is successful, goes on to create an instance of the Domino COM object using the method CoCreateInstance. We supply the class ID by using a constant defined in the files included from the Domino and Notes Toolkit for COM: CLSID_NotesSession.

Once we have this object, we must call initialize. Notice that we use the version where we supply a username and password. Assuming this is successful, we get the name of the current server and user and put them into properties for later use, currServerName and currUserName respectively.

Then we open the database directory of the current server and try to open the current user’s mailbox. The filename and path for the user’s mailbox is picked from the Domino Directory by the Domino COM object when we call OpenMailDatabase. We store a handle to the mailbox in the property m_pDatabase.

If we were able to open this database, we also get the database title for later use. This is stored in the property sMailDBTitle.

Now we are connected to the user’s mailbox, and necessary information and handles are put into properties for later use by the other methods.

**The SendMail method**
The SendMail method has the following parameter list:

[in] BSTR SendTo, [in] BSTR Subject, [in] BSTR Body,
[out, retval] BSTR *sMsg
Here is the code for the SendMail method:

```c
STDMETHODIMP CDominoMail::SendMail(BSTR SendTo, BSTR Subject,
BSTR Body, BSTR *sMsg)
{
    IDbDatabase  *m_pMailBox;
    IDbDocument  *m_pDocument, *m_pCopyDocument;
    IDbItem   *m_pItem;
    IDbDateTime  *m_pDateTime;
    CComBSTR   sMailBox("mail.box"), retMsgBSTR;
    CComVariant value;
    short    shortvar;
    int     pos, oldpos;
    wstring  wSendTo;

    shortvar = 0;
    m_pItem = NULL;
    if(CComBSTR(SendTo).Length() > 0) {
        m_pSession->GetDatabase (currServerName, sMailBox, true,
           &m_pMailBox);
        if (m_pMailBox) {
            m_pMailBox->CreateDocument (&m_pDocument);
            if (m_pDocument) {
                m_pDocument->AppendItemValue (CComBSTR("Memo"),
                   CComVariant("Memo"), &m_pItem);
                wSendTo.assign(SendTo);
                pos = wSendTo.find(',,');
                if (!(pos < 0)) {
                    m_pDocument->AppendItemValue (CComBSTR("SendTo"),
                        CComVariant(wSendTo.substr(0, pos).c_str()),
                        &m_pItem);

                    while (!(pos < 0)) {
                        oldpos = pos + 1;
                
```
pos = wSendTo.find(',', oldpos);
if (!(pos < 0))
    m_pItem->AppendToTextList(
        CComVariant(wSendTo.substr(oldpos, 
            pos-oldpos).c_str()));
else
    m_pItem->AppendToTextList(
        CComVariant(wSendTo.substr(oldpos, 
            wSendTo.length()-oldpos).c_str()));
} // end while
} // end if
else {
    m_pDocument->AppendItemValue(CComBSTR("SendTo"), 
        CComVariant(SendTo), &m_pItem);
}

m_pItem->get_Values (&value);
m_pDocument->AppendItemValue (CComBSTR("Recipients"), 
    value, &m_pItem);
m_pDocument->AppendItemValue (CComBSTR("From"), 
    CComVariant(currUserName), &m_pItem);
m_pDocument->AppendItemValue (CComBSTR("Subject"), 
    CComVariant(Subject), &m_pItem);
m_pDocument->AppendItemValue (CComBSTR("Body"), 
    CComVariant(Body), &m_pItem);
m_pDatabase->CreateDocument(&m_pCopyDocument);
if (m_pCopyDocument) {
    m_pDocument->CopyAllItems (m_pCopyDocument);
    m_pSession->CreateDateTime (CComBSTR(""), 
        &m_pDateTime);
    if (m_pDateTime) {
        m_pDateTime->SetNow();
}
}

Chapter 6: Using Visual C++  117
SendMail starts off by insuring that we supplied some recipients to send to. We then use the property m_pSession, which holds a handle to the active Notes session, to open the Mail.box file on the server. This is the database that the mail router works on.

We create a new Notes document in the Mail.box database and start appending some necessary items to it. First we need an item called Form, which tells Domino what kind of document this is.
Next we traverse the SendTo input parameter, splitting up each entry separated by a comma into a textlist in the Notes document. The list of recipients is put into both the item SendTo and Recipients.

We use the property currUserName to give the value for the From item. For the last two items, Subject and Body, we use the in parameters Subject and Body respectively.

Next we want to make a copy of this message in the user’s mailbox. We have a handle to the user’s mailbox in the property m_pDatabase and we use the CreateDocument method to create a new document in it. We call the method CopyAllItems on the message document we created earlier, m_pDocument. We send a reference to the newly created document, m_pCopyDocument, as parameter. This copies all the items in the first document to the new one.

To make the copy document turn up in the Sent mail view of the user’s database, we must add a last item, PostedDate. First we must create a new datetime object. This is done by calling the CreateDateTime method on the session object referenced by the property m_pSession, using m_pDateTime as parameter. We call SetNow on m_pDateTime to assign it the current date and time. By using the getLSLocalTime on the m_pDateTime object call we extract the value of it and store it in the variable value. This variable is in turn stored in the item PostedDate.

We save the copy document in the mailbox and release the handle. We then save the mail message in the mail.box database and clean up all the handles used. Last, if everything went well, we return the string “OK”.

**The GetMail method**
The GetMail method has just an out parameter so we use the following parameter list:

[out, retval] BSTR *sHTMLstring

And, here is the code for the GetMail method:

```cpp
STDMETHODIMP CDominoMail::GetMail(BSTR *sHTMLstring)
{
    IView   *m_pView; // ($Inbox) folder in the mailbox
    IDocument *m_pCurrDocument; // Current mail message
    IName   *m_pName; // Used to clean up senders name
    IDateTime *m_pDateTime; // Formats the message date
    IItem   *m_pItem; // Items in the mail message
    CComBSTR  retBSTR, valueBSTR;
    // Strings to be used for sending HTML back to the ASP script
```
int i;
short vBool;
wstring wBody;
retBSTR = CComBSTR("");
m_pDatabase->get_IsOpen (&vBool);
if(vBool) {
m_pDatabase->GetView(CComBSTR("($Inbox)"), &m_pView);
if(m_pView) {
m_pView->GetLastDocument(&m_pCurrDocument);
if(m_pCurrDocument) {
retBSTR.AppendBSTR(TableStart);
for(i=0; i<5; i++) {
if(m_pCurrDocument) {
retBSTR.AppendBSTR(TableStart);
retBSTR.AppendBSTR(RowStart);
retBSTR.AppendBSTR(ElemStart);
m_pCurrDocument->
GetFirstItem(CComBSTR("DeliveredDate"), &m_pItem);
if(m_pItem) {
m_pItem->get_DateTimeValue (&m_pDateTime);
}
m_pDateTime->get_DateOnly (&valueBSTR);
retBSTR.AppendBSTR(valueBSTR.Detach());
m_pDateTime->Release();
}
retBSTR.AppendBSTR(ElemEnd);
retBSTR.AppendBSTR(ElemStart);
m_pCurrDocument->GetFirstItem(CComBSTR("From"),
    &m_pItem);
if(m_pItem) {
    m_pItem->get_Text (&valueBSTR);
    m_pSession->CreateName(valueBSTR, NULL,
        &m_pName);
    m_pName->get_Common (&valueBSTR);
    retBSTR.AppendBSTR(valueBSTR.Detach());
    m_pName->Release();
}
retBSTR.AppendBSTR(ElemEnd);
retBSTR.AppendBSTR(ElemStart);
m_pCurrDocument->GetFirstItem(CComBSTR("Subject"),
    &m_pItem);
if(m_pItem) {
    m_pItem->get_Text (&valueBSTR);
    retBSTR.AppendBSTR(valueBSTR.Detach());
}
retBSTR.AppendBSTR(ElemEnd);
retBSTR.AppendBSTR(ElemStart);
retBSTR.AppendBSTR(TableEnd);
retBSTR.AppendBSTR(BodyStart);
m_pCurrDocument->GetFirstItem(CComBSTR("Body"),
    &m_pItem);
if(m_pItem) {
    m_pItem->get_Text (&valueBSTR);
}
wBody.assign(valueBSTR.Detach());
retBSTR.Append(wBody.substr(0, 500).c_str());
}
m pItem->Release();
retBSTR.AppendBSTR(BodyEnd);
retBSTR.AppendBSTR(RowEnd);
m pView->GetPrevDocument(m pCurrDocument,
    &m pCurrDocument);
} // End if
} // Next
retBSTR.AppendBSTR(TableEnd);
if(m pCurrDocument)
m pCurrDocument->Release();
} else {
    retBSTR.AppendBSTR(NoMail);
}
m pView->Release();
}
*sHTMLstring = retBSTR.Detach();
return S_OK;
}

In the declaration part of the GetMail method we notice a lot of strings being initialized containing HTML code. These are used to make the rest of the code less messy and easier to keep track of.

First we initialize the return string that will contain all the HTML making up the contents of the five most recent mail messages.

Then we make sure that the user’s mailbox is open. If that is the case we open the folder ($Inbox), which is where all incoming mail messages are put.

Note We call GetView on the database to open the folder. Folders and views are somewhat the same in Notes databases. They both contain documents, but the ways the documents get into them are different. We use the same calls to get to the documents in both a folder and a view.
Next we pick the last document in the folder by calling GetLastDocument. This is the most recent mail message. We then loop five times and call GetPrevDocument each time. This way we have traversed the five most recent mail messages. For each document we extract some properties and add their contents to the return string retBSTR. In between the item values we use the predefined strings with HTML code.

First we get the item DeliveredDate which holds the date the message arrived. This item is of type NotesDateTime. To display it properly, we put the value of the item into the NotesDateTime object m_pDateTime. We then use the property DateOnly, that is calling get_DateOnly in the code, to get the date as a string.

Next we want to display who sent this message. The item From contains the name of the sender. This name could be stored in a number of formats, so we want to format it to clean up the display and reduce the length. The value of the From item could, for example, be one of the following:

- CN=Carl Tuesday/O=Lotus
- Carl Tuesday/Lotus
- carl.tuesday@lotus.com

Much like what we did with the DeliveryDate above, we use a temporary object to get the format we want on the string. We create a new object of the type NotesName, which can be done using a method of the Notes session object m_pSession. The value from the From item is sent as initial value to the NotesName object m_pName. The Common property (get_Common in the code) of the NotesName object returns a string containing only the plain name of the sender, such as Carl Tuesday in the example above.

The Subject of the mail message is contained as text in the item Subject. We get this by using the Text property (get_Text in the code) and appending it to the return string.

The Body item of the mail document contains the mail message itself. This is a rich text field in Domino and can contain all sorts of things like text, graphics and file attachments. To simplify this example and make it all display nicely on one screen, we use the Text property (get_Text in the code) to only focus on the text in the Body item, and we pick only the 500 leftmost characters of the text.

**The Disconnect method**

The Disconnect method does not require any parameters, so we leave the parameter list blank.
Here is the code for the Disconnect method:

```c
STDMethodIMP CDominoMail::Disconnect()
{
    HRESULT hResult;

    // Release the Session and Database pointers
    m_pDatabase->Release();
    m_pSession->Release();
    // UnInitialize the COM Library.
    CoUninitialize();

    return S_OK;
}
```

Here we simply call Release on each of the global variables and finally uninitialize.

**Build and register the component**

All coding on our component is done now. In Visual C++ under the Build menu select the DominoObjExt.dll file to build. If you want to test the component on another machine remember to register it using the regsvr32-program. You do this by selecting Run on the Start menu in Windows, writing regsvr32 and giving the path and filename of the dll file an as argument, like this

```bash
regsvr32 c:\comdll\dominoobjext.dll
```

**Note** Also make sure that the location of the Domino dll files is in the path statement on your server.

**Testing the component using ASP**

To test the DominoMail component we build an ASP script that will create an instance of it. We make the second generation of CompactMail from the previous chapter and call it COMpactMail2. Using the DominoMail component we can log in, read and send mail. We called the file COMpactMail2.asp and the code for the ASP script we used is as follows:

```html
<%Option Explicit
Response.Expires = 0 'No cashing%
<HTML>
<HEAD>
```
<TITLE>Send mail</TITLE>

<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
Dim nMail  ' Holds handle to the DominoMail object
Dim sUsername ' Holds the username supplied by user
Dim sPassword ' Holds the password supplied by user
Dim sSendTo ' Holds the list recipients
Dim sSubject ' Holds the input for the Subject item
Dim sBody ' Holds the input for the Body item
Dim sErrorMsg ' Supply error message to the user
Dim sMode ' Mail mode (Read, Compose, Send)
Dim sTitle ' The DBTitle for the users mailbox
Dim bLoggedIn ' Boolean - indicate if we are logged in

bLoggedIn = false ' We are not logged to start with

Sub ScriptInput  ' Procedure that read the input from user
    sUsername = Trim(Request.Form("Username"))
    sPassword = Trim(Request.Form("Password"))
    sSendTo = Trim(Request.Form("SendTo"))
    sSubject = Trim(Request.Form("Subject"))
    sBody = Trim(Request.Form("Body"))
    sMode = Trim(Request.Form("Mode"))
    If sMode = "" then
        sMode = "Read"
    End If
    sTitle = "COMpactMail II"
    If sUsername <> "" And sPassword <> "" Then
        set nMail = Server.CreateObject("DominoObjExt.DominoMail")
        sErrorMsg = nMail.Connect(sUsername, sPassword)
        If sErrorMsg = "" Then

Chapter 6: Using Visual C++  125
Sub ScriptAction 'Reading and sending the mail message
If bLoggedIn then
    Select Case sMode
    Case "Send"
        sErrorMsg = nMail.SendMail(sSendTo, sSubject, sBody)
        If sErrorMsg = "OK" Then
            Response.Write("<H3>Message sent!</H3>"
        Else
            Response.Write(sErrorMsg)
        End If
    Case "Read"
        Response.Write(nMail.GetMail())
    End Select
    call nMail.DisConnect()
    set nMail = Nothing
End If
End Sub

Sub ToggleState
    Select Case sMode
    Case "Send"
        If sErrorMsg = "OK" Then
            sMode = "Read"
        Else
            sMode = "Compose"
    End If
    End Select
End Sub
End If
Case "Read"
sMode = "Compose"
Case "Compose"
sMode = "Send"
End Select
End Sub
</SCRIPT>

<BODY BGCOLOR="white" TEXT="black">
<%ScriptInput%>
<H2><%=sTitle%></H2><HR noshade>
<FORM ACTION="CompactMail2.asp" METHOD="post" NAME="Mail">
<%If Not bLoggedIn then%>
<TABLE border=0 cellspacing=0 cellpadding=2 cols=2>
<TR><TD>Username:</TD><TD><INPUT type="text" name=Username value="<%=sUsername%>"></TD></TR>
<TR><TD>Password:</TD><TD><INPUT type="password" name=Password value="<%=sPassword%>"></TD></TR>
</TABLE>
<INPUT type="submit" value="Log In" id=submit1 name=submit1>
<%=sErrorMsg%>
<%Else%>
<INPUT type="hidden" name=Username value="<%=sUsername%>">
<INPUT type="hidden" name=Password value="<%=sPassword%>">
</%Else%>
</FORM>
</BODY>

Chapter 6: Using Visual C++  127
This ASP contains an HTML form, three VBScript subroutines and some inline VBScript code. The layout and contents of the HTML form changes depending on the state of the sMode variable. The normal flow goes like this:

1. Initially we are presented with a login request; we supply a username and password.
2. When we click Log In we are given our five latest mail messages.

3. When we click Compose Mail we are presented with a form to supply recipients, subject and the message.
4. We click Send and get the message “Message sent!” if all is well.

5. We click Back to go back to reading the mail.
This simple example shows how we can apply the component we have made. By calling a few methods, we get much of the same output as in the plain ASP sample in chapter 5.

**Comment on the ASP script code**
Most of this ASP code should be familiar from chapter 5. The main new part is in the subroutine ScriptAction where we interact with the object created in the ScriptInput subroutine. It might also be a bit difficult to see how the form layout changes. This is done by including a hidden input control named Mode in the form at the bottom which holds the state, that is the contents of the sMode variable.

```html
<INPUT type="hidden" value="<%=sMode%>" name=Mode>
```

For each submission of the form we toggle the value of sMode and post it back on the next submission through the hidden input control.
In this chapter we have looked at how to use the Domino COM interface to connect to the Domino server from C++. The C++ API for Domino has been around for some time, but using the COM interface we get a real session to the Domino server. We showed how to make a component in C++ that encapsulates a Domino user’s mailbox. By including this component in an ASP script, we were able to both read and send mail on the Domino server.
Appendix A
Lotus Domino and Notes Toolkit for COM

Together with the built-in COM support in Domino R5.0.2b Lotus has made a toolkit available: The Domino and Notes Toolkit for COM (“the COM Toolkit”). It is a comprehensive source of samples and documentation to help get you started on creating COM applications using Domino back-end objects from your favorite Windows development environment.

You can get the toolkit from the Lotus Developer Network at

http://www.lotus.com/developer/

Follow the link to toolkits or search for “COM Toolkit”.

The Toolkit contains the following:

• COM support files (source files).
• Sample applications that demonstrate using the COM interfaces, written in Visual C++, Visual Basic, VBA, and VBScript.

  Each sample application includes source files, build files, and Notes databases as needed, and documentation in a Readme.txt file.

• Documentation in Microsoft HTML Help format.

You can access the Domino Objects using COM without the Toolkit, but the Toolkit can ease your task significantly. The Readme files for the samples describe, for each application, what the requirements are; how to build the application and set up to run it; and what results to look for. Even if your development language is not one of those illustrated, the large number of samples should give you valuable information about how to proceed.
Special notices

This publication is intended to help you create customized solutions with Lotus Domino R5.0.2b.

The information in this publication is not intended as the specification of any programming interfaces that are provided by Lotus Domino. See the publications section of the announcement for Lotus Domino and related products for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available subject to appropriate terms and conditions, including, in some cases, payment of a fee.
The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

This document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
AS/400
DB2
IBM®
MQSeries
OS/2
OS/Warp
The following are trademarks of Lotus Development Corporation in the United States and/or other countries:

Domino.Doc
LotusScript®
Lotus SmartSuite®
Notes Mail®
NotesPump
NotesSQL
Lotus®
Lotus Domino
Lotus Notes®
RealTime Notes
SmartIcons®

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power to Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer — Tivoli A/S.

C-bus is a trademark of Corollary, Inc.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

SET, SET Secure Electronic Transaction, and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product or service names may be the trademarks or service marks of others.
Additional Web material

Additional Web material is referenced in this book and can be found on the IBM Redbooks Web site. The material is:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5670-ie.zip</td>
<td>Zipped Microsoft Internet Explorer file with simple VBScript example from Chapter 2</td>
</tr>
<tr>
<td>5670word.zip</td>
<td>Zipped Microsoft Word sample files from Chapter 3</td>
</tr>
<tr>
<td>5670excl.zip</td>
<td>Zipped Microsoft Excel sample files from Chapter 3</td>
</tr>
<tr>
<td>6670-vb.zip</td>
<td>Zipped Visual Basic sample files from Chapter 4</td>
</tr>
<tr>
<td>5670-asp.zip</td>
<td>Zipped ASP sample files from Chapter 5</td>
</tr>
<tr>
<td>5670-cpp.zip</td>
<td>Zipped Visual C++ sample files from Chapter 6</td>
</tr>
</tbody>
</table>

How to get the Web material

To get the Web material point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG245670

Alternatively, you can go to the redbooks Web site at:

http://www.redbooks.ibm.com

Select Additional Materials and open the file that corresponds with the redbook form number.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

International Technical Support Organization publications

For information on ordering these ITSO publications see, “How To Get ITSO Redbooks.”

- *LotusScript for Visual Basic Programmers*, IBM form number SG24-4856, Lotus part number 12498
- *Lotus Domino 5.0: A Developers Handbook*, IBM form number SG24-5331, Lotus part number CC7EDNA
- *Developing e-business Application Using Lotus Enterprise Solution Builder R3.0*, IBM form number SG24-5405, Lotus part number CT69TNA
- *Performance Considerations for Domino Applications*, IBM form number SG24-5602, Lotus part number CT7V6NA
- *Building a Portal with Lotus Domino R5*, IBM Redpaper REDP0019 (only available in softcopy from the IBM Redbooks Web site)
- *Developing Web Applications Using Lotus Notes Designer for Domino 4.6*, IBM form number SG24-2183, Lotus part number 12974
- *Lotus Notes 4.5: A Developers Handbook*, IBM form number SG24-4876, Lotus part number AA0425 (available in soft copy only)
- *Creating Customized Solutions with Domino.Doc*, IBM form number SG24-5658, Lotus part number CC6X3NA
- *Using VisualAge for Java to Develop Domino Applications*, IBM form number SG24-5424, Lotus part number CT6ENNA
- *Connecting Domino to the Enterprise Using Java*, IBM form number SG24-5425, Lotus part number CT6EMNA
- *Lotus Domino for AS/400: Integration with Enterprise Applications*, IBM form number SG24-5345, Lotus part number CT7BMNA
- *Lotus Solutions for the Enterprise, Volume 1. Lotus Notes: An Enterprise Application Platform*, IBM form number SG24-4837, Lotus part number 12968
- Lotus Solutions for the Enterprise, Volume 4. Lotus Notes and the MQSeries Enterprise Integrator, IBM form number SG24-2217, Lotus part number 12992
- Lotus Solutions for the Enterprise, Volume 5. NotesPump, the Enterprise Data Mover, IBM form number SG24-5255, Lotus part number CT69DNA
- Enterprise Integration with Domino for S/390, IBM form number SG24-5150

Other Lotus-related ITSO publications

The publications listed in this section may also be of interest:

- Lotus Notes and Domino Take Center Stage: Upgrading from R4 to R5, IBM form number SG24-5630, Lotus part number CT6FDNA
- A Roadmap for Deploying Domino in the Organization, IBM form number SG24-5617, Lotus part number CT6P8NA
- The Three Steps to Super.Human.Software: Compare, Coexist, Migrate; From Microsoft Exchange to Lotus Domino, Part One: Comparison, IBM form number SG24-5614, Lotus part number CT7QTNA
- The Three Steps to Super.Human.Software: Compare, Coexist, Migrate; From Microsoft Exchange to Lotus Domino, Part Two: Coexistence and Migration, IBM form number SG24-5615, Lotus part number CT7QWNA
- Lotus Notes and Domino R5.0 Security Infrastructure Revealed, IBM form number SG24-5341, Lotus part number CT6TPNA
- The Next Generation in Messaging. Moving from Microsoft Mail to Lotus Notes and Domino, IBM form number SG24-5152, Lotus part number CT7SBNA
- Eight Steps to a Successful Messaging Migration: A Planning Guide for Migrating to Lotus Notes and Domino, IBM form number SG24-5335, Lotus part number CT6HINA
- Deploying Domino in an S/390 Environment, IBM form number SG24-2182, Lotus part number 12957
- The Next Step in Messaging: Upgrade Case Studies on Lotus cc:Mail to Lotus Domino and Lotus Notes, IBM form number SG24-5100, Lotus part number 12992
- The Next Generation in Messaging. Moving from Novell GroupWise to Lotus Notes and Domino, IBM form number SG24-5321, Lotus part number CT7NNNA
- High Availability and Scalability with Domino Clustering and Partitioning on Windows NT, IBM form number SG24-5141, Lotus part number CT6XMIE
- From Client/Server to Network Computing. A Migration to Domino, IBM form number SG24-5087, Lotus part number CT699NA (available in soft copy only)
• *Netfinity and Domino R5.0 Integration Guide*, IBM form number SG24-5313, Lotus part number CT7BKNA

• *Lotus Domino R5 for IBM RS/6000*, IBM form number SG24-5138, Lotus part number CT7BHNA

• *Lotus Domino Release 4.6 on IBM RS/6000: Installation, Customization and Administration*, IBM form number SG24-4694, Lotus part number 12969

• *High Availability and Scalability with Domino Clustering and Partitioning on AIX*, IBM form number SG24-5163, Lotus part number CT7J0NA

• *Lotus Domino for AS/400: Installation, Customization and Administration*, IBM form number SG24-5181, Lotus part number AA0964

• *Lotus Domino for S/390 Release 4.6: Installation, Customization & Administration*, IBM form number SG24-2083

• *Lotus Domino for S/390 Performance Tuning and Capacity Planning*, IBM form number SG24-5149, Lotus part number CT6XNIE

• *Porting C Applications to Lotus Domino on S/390*, IBM form number SG24-2092, Lotus part number AB1720 (available in soft copy only)

• *Managing Domino/Notes with Tivoli Manager for Domino, Enterprise Edition, Version 1.5*, IBM form number SG24-2104 (available in soft copy only)

---

**Redbooks on CD-ROMs**

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at [http://www.redbooks.ibm.com/](http://www.redbooks.ibm.com/) for information about all the CD-ROMs offered, updates and formats.

<table>
<thead>
<tr>
<th>CD-ROM Title</th>
<th>Collection Kit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lotus Redbooks Collection</td>
<td>SK2T-8039</td>
</tr>
<tr>
<td>Tivoli Redbooks Collection</td>
<td>SK2T-8044</td>
</tr>
<tr>
<td>Application Development Redbooks Collection</td>
<td>SK2T-8037</td>
</tr>
<tr>
<td>Netfinity Hardware and Software Redbooks Collection</td>
<td>SK2T-8046</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (BkMgr Format)</td>
<td>SK2T-8040</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (PDF Format)</td>
<td>SK2T-8043</td>
</tr>
<tr>
<td>AS/400 Redbooks Collection</td>
<td>SK2T-2849</td>
</tr>
<tr>
<td>Transaction Processing and Data Management Redbooks Collection</td>
<td>SK2T-8038</td>
</tr>
<tr>
<td>Networking and Systems Management Redbooks Collection</td>
<td>SK2T-6022</td>
</tr>
<tr>
<td>System/390 Redbooks Collection</td>
<td>SK2T-2177</td>
</tr>
<tr>
<td>IBM Enterprise Storage and Systems Management Solutions</td>
<td>SK3T-3694</td>
</tr>
</tbody>
</table>

Related publications 143
Other resources

These publications are also relevant as further information sources:


Web sites

These Web sites are also relevant as further information sources:

  Lotus Developer Network — Lotus’ primary destination for the latest developer information and resources. Contains articles about new and current technologies along with relevant tips and techniques to help you build dynamic collaborative e-business applications.
- [http://notes.net/](http://notes.net/)
  Notes.net from Iris — the developers of Notes and Domino. A technical Web site with discussion forums, documentation and the Webzine Iris Today with many good articles about performance in Domino.
  The IBM developerWorks Web site is designed for software developers, and features links to a host of developer tools, resources, and programs.
- [http://support.lotus.com/](http://support.lotus.com/)
  Lotus Support’s Web site — Search using keywords or browse the Lotus Knowledge Base and locate helpful and informative tech notes and technical papers for the entire Lotus Product family. This source of information contains the latest technical information updated hourly.
How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

  Search for, view, download or order hardcopy/CD-ROM Redbooks from the IBM Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are redbooks in progress; not all redpieces become redbooks and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**
  Send orders by e-mail including information from the IBM Redbooks fax order form to:
  
  e-mail address

  In United States: usib6fpl@ibmmail.com

- **Telephone Orders**
  United States (toll free) 1-800-879-2755
  Canada (toll free) 1-800-IBM-4YOU

- **Fax Orders**
  United States (toll free) 1-800-445-9269
  Canada (toll free) 1-800-267-4455

The latest information for customers may be found at the IBM Redbooks Web site.

---

IBM intranet for employees

# IBM Redbook fax order form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

First name ___________________________  Last name ___________________________

Company _____________________________

Address _____________________________

City _____________________________  Postal code ________  Country ________

Telephone number ____________________  Telefax number ____________________  VAT number ____________________

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date ___________  Card issued to ____________________  Signature ____________________

---

*We accept American Express, Diners, Eurocard, MasterCard, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.*
Index

Symbols
<OBJECT> tag, 76, 94

A
Active Server Pages. See ASP
Active Type Library (ATL), 96
ActiveDocument. See under VBA
Agents, 3
Application Wizard, 57
ASP, 73
  COMpact mail example, 80
  Domino server security, 75
  Global.asa file, 84
  HTML form, 88, 127
  include a component, 94
  script loading component, 124
  ATL. See Active Type Library
  ATL COM Wizard, 97
  Automation, 3

B
Back-end classes, 3
Black box, 8

C
CLSID, 76
COM, 7
  COM components, 8
  create instance of Domino session using <OBJECT> tag, 76
  create instance of Domino session using CreateObject, 76
  creating a component using Visual C++, 95
  differences from LotusScript in accessing Domino, 70
  Domino object components, 9
  Domino server security, 75
  what is COM, 7
  Component Object Model. See COM
  CreateObject, 25, 45, 50, 76,
  86, 94, 125

D
Directory, 2
Document Library database, 42, 55
Domino and Notes Toolkit for COM
domobj.h, 103
domobj.id, 103
domobj_i.c, 103
Domino Designer, 3
Domino Object Model, 1, 3
object hierarchy, 6
Domino server
  automation by agents, 3
directory, 2
  front-end and back-end classes, 3
  Java/COM Restrictions, 75
  messaging, 2
  object store, 2
  replication, 2
  security, 2, 75
  Web server authentication setting, 78
  workflow, 2
  Domobj.h, 103, 109
  Domobj.id, 103
  Domobj_i.c, 103

E
Early binding, 10
ErrorHandler, 38, 67
Excel Expense Statement
template, 35

F
Front-end classes, 3

H
HTML
  <OBJECT> tag, 76
  in ASP, 88

I
IIS, 73, 95
  handling type libraries, 84
  setup for MS Visual InterDev, 79

L
Late binding, 10
Lotus Domino and Notes Toolkit for COM, 133
LotusScript
differences from Visual Basic in accessing Domino, 70

M
Messaging, 2
Microsoft Access
  setting up access to Domino objects, 20
Microsoft Excel
  Activate, 38
  ActiveWorkbook, 45
  COMMON.XLS, 35, 40, 42,
  44, 45, 49, 51
  expense statement example, 34
  setting up access to Domino objects, 20
Microsoft Internet Explorer
  simple example, 14
Microsoft Internet Information
  Server. See IIS
Microsoft Office
  setup for Domino objects, 19
Microsoft Personal Web
  Server, 73, 95
Microsoft Visual C++, 95
  Active Type Library, 96
  adding Domino header files, 103
  adding methods to component, 110
  ATL COM Wizard, 97
  creating a COM component, 95
  declaring internal component properties, 110

149
Normal.dot, 12, 30
Notes classes
  hierarchical relation, 6
  NotesACL, 4
  NotesACLEntry, 4
  NotesAgent, 4
  NotesDatabase, 4
  NotesDateRange, 5
  NotesDateTime, 5
  NotesDbDirectory, 3
  NotesDocumentCollection, 4
  NotesEmbeddedObject, 5
  NotesForm, 5
  NotesInternational, 5
  NotesLog, 5
  NotesName, 5
  NotesNewsLetter, 5
  NotesOutline, 5
  NotesOutlineEntry, 5
  NotesRegistration, 5
  NotesReplication, 5
  NotesRichTextItem, 4
  NotesRichTextStyle, 4
  NotesSession, 3
  NotesView, 4
  NotesViewColumn, 4
NOTES.INI
  DisableComNP, 77
  KeyFilename, 26
NotesACL class, 4
NotesACLEntry class, 4
NotesAgent class, 4
NotesDatabase class, 4, 25, 36, 44, 59, 60, 62, 64
  CreateDocument, 45, 116, 117
  DesignTemplateName, 61
  FilePath, 61
  get_IsOpen, 120
  get_Title, 115
  GetView, 26, 37, 62, 64, 87, 120
  IsOpen, 86, 92
  getTitle, 61
  NotesDateRange class, 5
  NotesDateTime class, 5
  get_DateOnly, 121
  get_LSLocalTime, 118
  SetNow, 117
  NotesDbDirectory class, 3, 60
  GetFirstDatabase, 61
  GetNextDatabase, 62
  OpenMailDatabase, 14, 15, 86, 92, 114
  NotesDocument class, 4, 25, 36, 44, 64
  AppendItemValue, 116, 117
  ComputeWithForm, 47
  CopyAllItems, 117
  CreateRichTextItem, 45
  GetFirstItem, 87, 93, 120, 121
  GetItemValue, 27, 28, 37, 38, 65, 66, 87, 93
  GetView, 92
  HasItem, 65
  IsResponse, 65
  ReplaceItemValue, 46, 118
  Save, 47, 118
  NotesDocumentCollection class, 4
  NotesEmbeddedObject class, 5
  NotesForm class, 5
  NotesInternational class, 5
  NotesItem class
    get_DateTimeValue, 120
    get_Text, 121
    Text, 88, 93
  NotesLog class, 5
  NotesName class, 5, 64
    Common, 66, 87, 93
    get_Common, 121
  NotesNewsLetter class, 5
  NotesOutline class, 5
  NotesReplication, 5
  NotesRichTextItem class, 4, 44
    EmbedObject, 47
  NotesRichParagraphStyle class, 4
  NotesRichTextStyle class, 4
  NotesRichTextTab class, 5
  NotesSession class, 3, 14, 15, 25, 36, 59, 76, 86, 114
    CommonModule, 14, 36, 45, 67
    CreateDateTime, 117
    CreateName, 66, 87, 93, 121
    get_ServerName, 114
    get_UserName, 114
  NotesDatabase, 37, 42, 45, 62, 64, 116
  GetDbDirectory, 14, 15, 6, 61, 86, 92, 114
  InitializeUsingNotesUserName, 77
  Initialize, 14, 15, 26, 38, 67, 70, 77
  InitializeUsingNotesUserName, 70, 86, 114
  ServerName, 86, 92
  usernameobject, 86
  NotesView class, 4, 25, 36, 62, 64
    CreateViewNav, 63
    CreateViewNavFromCategory, 64
    GetFirstDocument, 26, 37
    GetLastDocument, 87, 92, 120
    GetNextDocument, 27, 37
    GetNthDocument, 28, 37
    GetPrevDocument, 88, 93, 122
  NotesViewColumn class, 4
  NotesViewEntry class, 4, 62, 64
    ColumnValues, 63
  NotesViewEntry class, 4, 62, 64
    Document, 65
  NotesViewEntryCollection, 4
  NotesViewNavigator class, 63
    GetFirst, 63
  NotesViewNavigator class, 4
    GetNextCategory, 63
  NotesViewNavigator class, 4, 62, 64
    GetFirstDocument, 64
    GetNextDocument, 66

O
  Object hierarchy, 6
  Object store, 2
  OLE Automation, 10

P
  PROGID, 76, 97, 107

R
  Replication, 2

S
  Security, 2
  Standard Template Library (STL), 96
T
Template
Document Library database, 42
Excel Expense Statement
template, 35
MS Word, 30
Toolbar
adding in MS Word, 31
Toolkit. See Domino and Notes
Toolbar adding in MS Word
Toolkit for COM

V
VBA
Activate, 38, 46
ActiveDocument, 26, 28
ActiveWorkbook, 45
CreateObject, 25
expense statement example, 34
exporting modules for reuse, 34
name lookup example, 21
setting up access to Domino
objects, 20
simple MS Word example, 13
VBScript, 73, 85
simple example in Internet
Explorer, 14
Visual Basic
Application Wizard, 57
create new project, 57
dexample of getting information
don Domino, 55
Single Document Interface, 57
Visual Basic for Applications. See
VBA

W
Workflow, 2

Index 151
**IBM Redbooks review**

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook “made the difference” in a task or problem you encountered. Using one of the following methods, please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.

- Use the online **Contact us** review redbook form found at [http://www.redbooks.ibm.com/](http://www.redbooks.ibm.com/)
- Fax this form to: USA International Access Code +1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

<table>
<thead>
<tr>
<th>Document Number</th>
<th>SG24-5670-00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redbook Title</td>
<td>COM Together — with Domino</td>
</tr>
</tbody>
</table>

**Review**

---

**What other subjects would you like to see IBM Redbooks address?**

---

**Please rate your overall satisfaction?**

- [ ] Very Good
- [ ] Good
- [ ] Average
- [ ] Poor

**Please identify yourself as belonging to one of the following groups:**

- [ ] Customer
- [ ] Business Partner
- [ ] Solution Developer
- [ ] IBM, Lotus or Tivoli Employee
- [ ] None of the above

**Your email address:**

The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.

- [ ] Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.

**Questions about IBM’s privacy policy?**

The following link explains how we protect your personal information. [http://www.ibm.com/privacy/yourprivacy/](http://www.ibm.com/privacy/yourprivacy/)

© Copyright IBM Corp. 2000
COM Together — with Domino

Søren Peter Nielsen, Kjetil Meidell Andenes, Kevin P. Smith

International Technical Support Organization
www.redbooks.ibm.com