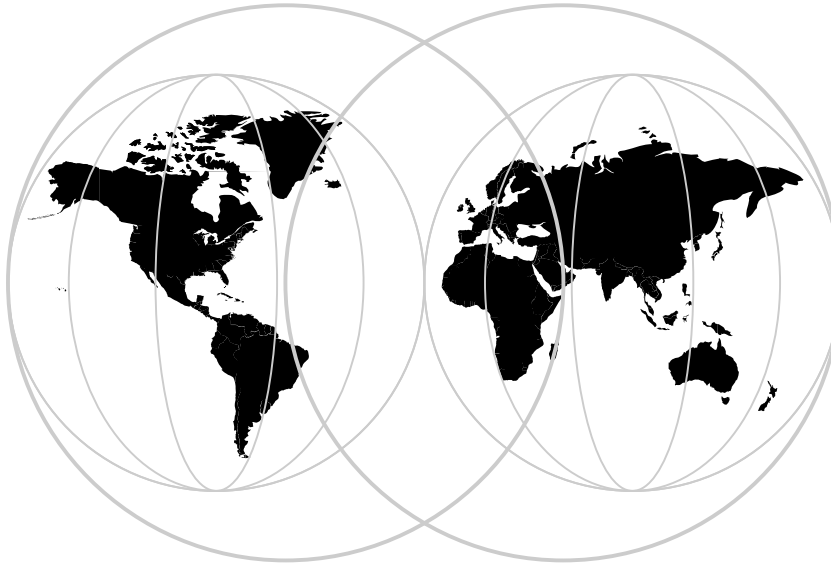
Lotus

IBM

LotusScript for Visual Basic Programmers



International Technical Support Organization

Edition Notice

First Edition (August 1996)

This edition applies to Release 4 of Lotus Notes.

Comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Abstract

This redbook describes how to work with LotusScript, a BASIC-like, object-oriented programming language that comes as part of Lotus Notes Release 4 and other Lotus products.

The introductory chapter compares LotusScript and Visual Basic. Other chapters cover Lotus Notes as an OLE 2 automation server and OLE 2 automation client. A description of the LotusScript Notes classes is also provided.

The redbook focuses on how to convert HiTest applications, and how to access the major Notes elements using:

- The HiTest Basic API
- Notes classes in Visual Basic through OLE automation
- LotusScript in Notes.

This redbook was written for Visual Basic programmers who want to learn about the features of LotusScript, and who are planning to migrate their HiTest applications.

Some knowledge of the HiTest Basic API and of Visual Basic is assumed.

Contents

Preface

This redbook describes how to work with LotusScript, a BASIC-like, object-oriented programming language that comes as part of Lotus Notes Release 4 and other Lotus products.

The introductory chapter compares LotusScript and Visual Basic. Other chapters cover Lotus Notes as an OLE 2 automation server and OLE 2 automation client. A description of the LotusScript Notes classes is also provided.

The redbook focuses on how to convert HiTest applications, and how to access the major Notes elements using:

- The HiTest Basic API
- Notes classes in Visual Basic through OLE automation
- LotusScript in Notes.

This redbook was written for Visual Basic programmers who want to learn about the features of LotusScript, and who are planning to migrate their HiTest applications.

Some knowledge of the HiTest Basic API and of Visual Basic is assumed.

Note This redbook is available in HTML format and in Adobe Acrobat format on the World Wide Web. The URL is http://www.lotus.com/devtools. Also, the code samples provided throughout the book are available for your use on http://www.lotus.com/redbook.

How This Redbook Is Organized

This redbook is organized as follows:

- Chapter 1, “LotusScript and Visual Basic: A Comparison”

  This chapter describes the differences and similarities between Visual Basic Release 4 and LotusScript.

- Chapter 2, “The Notes Integrated Development Environment”

  This chapter looks at the Integrated Development Environment (IDE) for creating and programming Notes forms, which is the most complex application development environment.

- Chapter 3, “LotusScript Notes Classes”

  This chapter discusses the LotusScript database classes and UI classes, how to access the classes, and the properties and methods of the classes.

- Chapter 4, “Lotus Notes as an OLE 2 Automation Server”

  This chapter describes how to develop Notes applications in Visual Basic using the Notes classes.

- Chapter 5, “Converting HiTest Applications”

  This chapter describes how to convert HiTest applications using the original HiTest code directly in Notes Release 4, using Notes classes in Visual Basic through OLE automation, and using LotusScript in Notes.

- Chapter 6, “Accessing Notes Sessions”

  This chapter discusses how to access Notes sessions in HiTest, using Notes classes through OLE automation in Visual Basic, and using LotusScript in Notes.

- Chapter 7, “Accessing Notes Databases”

  This chapter discusses how to access Notes databases in HiTest, using Notes classes through OLE automation in Visual Basic, and using LotusScript in Notes.

- Chapter 8, “Accessing Notes Views”

  This chapter discusses how to access Notes views and folders in HiTest, using Notes classes through OLE automation in Visual Basic, and using LotusScript in Notes.

- Chapter 9, “Accessing Notes Documents”

  This chapter discusses how to access Notes documents in HiTest, using Notes classes through OLE automation in Visual Basic, and using LotusScript in Notes.

- Chapter 10, “Accessing Notes Items”

  This chapter discusses how to access Notes items in HiTest, using Notes classes through OLE automation in Visual Basic, and using LotusScript in Notes.

- Chapter 11, “Using Notes as an OLE 2 Automation Client: Creating Objects”

  This chapter describes how to create embedded and linked objects in a Notes document manually and using LotusScript.

- Chapter 12, “Using Notes as an OLE 2 Automation Client: Managing Objects”

  This chapter describes how to edit and update an embedded or linked object in a Notes document both manually and using LotusScript.

- Appendix A, “HiTest and LotusScript Notes Classes: Comparing Functions”

  This appendix lists equivalent functions in HiTest and LotusScript.

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 Cambridge Center.

This project was designed and managed by:

Marion Bergner-Hawker
International Technical Support Organization, Cambridge, MA USA

The authors of this document are:

Stella Yuan
IBM China

Benjamin Sondakh
IBM Indonesia

Bill Shadish
Fundamental Objects, Inc.
bills@fo.com

This redbook is the result of a residency conducted at the International Technical Support Organization, Cambridge, MA USA.

Thanks to the following people for the invaluable advice and guidance provided in the production of this redbook:

Alex Neihaus
Notes Product Marketing, Lotus Development

David Rosenbaum
Notes Product Management, Lotus Development

Gary Devendorf
Notes Product Management, Lotus Development

Graphic Services
Lotus Development

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address: redbook@vnet.ibm.com.

Your comments are important to us!

Chapter 1
LotusScript and Visual Basic: A Comparison

This chapter describes the differences and similarities between Visual Basic Release 4 and LotusScript, which comes as part of Lotus Notes Release 4 and other Lotus products, such as Word Pro, Freelance, and Approach. We will compare the syntactical language portions of LotusScript and Visual Basic. We will also look at the environments that these languages work within. Visual Basic relies on the Visual Basic design environment, access to external controls and host applications such as Excel. LotusScript relies on Notes, Word Pro and other Lotus products to provide the user interface it needs to operate within.

So, while the primary focus will be on the syntax, we will often explain the advantages gained by using these external tools with both LotusScript and Visual Basic.

Some of the topics covered are:

- A syntactical language comparison
- The programming model of each language
- The Object-Oriented development characteristics of each
- External objects; such as servers, controls, components that add to the language.

History

Microsoft released Visual Basic in 1989, but the language has actually been in use since the 1960s. In fact, there are now many versions of BASIC available, with implementations found in tools such as Powerbuilder, ToolBook, Visual Basic and Lotus Notes Release 4.

Prior to Release 4 of Notes, Lotus applications used a macro command language as the method of development. While this macro language is fairly robust in power, professional developers have come to prefer a more powerful development language.

LotusScript

With Lotus Notes Release 4 and other Lotus desktop applications in SmartSuite, Lotus has added LotusScript as a version of BASIC that can be used to extend the functionality of the applications and to integrate with other software. This is actually Release 3 of LotusScript. Lotus had released LotusScript Release 1 in Improv, with Release 2 appearing in Lotus Forms and Notes ViP, an earlier development tool created to interoperate with Notes. (Notes ViP is now developed and marketed exclusively by Revelation Technologies, Inc. as Revelation ViP for Lotus Notes.) The ViP version of BASIC was the first to truly implement Object-Oriented Programming (OOP) features, much of which has been carried forward into the current version of LotusScript. The implementation of LotusScript within Lotus Notes delivers an impressive combination of workgroup capabilities, programming environment and the standard reporting and database features necessary for application development. This chapter will use Lotus Notes to describe user interface features and LotusScript in the comparison to Visual Basic.

Visual Basic

There are three versions of Visual Basic: VBScript, VBA and VB. VBScript is the object-enabling language that is used within Web browsing tools to allow objects to be loaded, change their properties and react to changes (events) within the objects. VBA is a superset of VBScript, providing interaction with Microsoft's application tools. Visual Basic is a full development tool that provides reporting, database interoperation and screen design.

Programming Model Differences

There are some differences in the general flow of using LotusScript when compared to Visual Basic. Some of the more salient points are:

- Interface Tools Used

  Visual Basic (the package) supplies the ability to create user interface elements, like forms and dialog boxes. LotusScript depends upon Notes or the containing Lotus application to provide the actual user interface.

  Visual Basic allows you to create run time, royalty-free applications. These applications are distributed as pseudo-compiled modules and interpreted before being executed, at run time. LotusScript currently works within Lotus Notes and Smart Suite and is distributed to users of those applications. Note that LotusScript "scripts" are in fact compiled.

- Object-Oriented Terminology

  LotusScript uses the term “class” to describe the “contract” defining the properties, methods and potentially the events related to an object. Microsoft uses the term object in this case and uses Class to describe underlying low-level language constructs. LotusScript has the term Inheritance in its vocabulary — Visual Basic does not.

- Screen flow

  Notes screens flow in a manner similar to a script-enabled help file, within one form replacing several windows of a traditional, dialog-based application. A good deal of information (potentially screens’ worth) is often placed onto one form in Notes. This differs from the one main screen with supporting dialogs approach used by Visual Basic.

  One major reason for the single-screen approach is that a large amount of information can be placed onto the screen and then “scrolled” or graphically navigated to, as opposed to the more traditional dialog-box presentation of the data. A second major reason is that the data is then relatively flexible when presented into different screen resolutions or even different operating system environments.

  Presentation actually becomes more an issue of style and user training than one way being necessarily better than the other.

  This is something that will take a little getting used to from the design side as well as the eventual usage of your application.

- Control Placement

  When designing your screens in Notes, you will place controls in positions relative to other controls or text on the screen. Rather than worrying about finite twip or pixel level placement, you will just put a button to the right of the prior button or to the left of the button that comes next. Notes handles the final placement of the control, to resolve potential cross-platform resolution dependencies that occur when you specify the coordinate system within your application itself. If you really want to decide where things sit on the screen, you can add spaces or tabs between fields and controls to change the presentation of the controls.

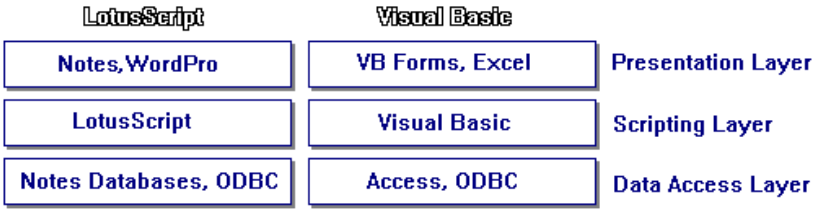
- OLE 2 Support

  Notes will allow you to place OLE 2.0 controls on the screen. As of Notes Release 4.1, Notes is a full OCX container. For example, events are not handled in exactly the same way and it is more difficult to change the properties of existing controls. Lotus is working on a Component Software Development Kit (SDK) to enable OLE control developers to create controls within Notes. This SDK will also allow

you to react to events generated within these controls, in the same way as other OLE2 containers do now.

- Structure

  With Release 4 of Lotus Notes, both LotusScript and VisualBasic operate within the same type of development environments. Each has a layer to present and retrieve information to the user and an underlying layer to store this information.



  With Release 4 of Notes, Lotus has added the LotusScript programming layer to bind together the Lotus applications packages as well as to increase the ability to use other packages with Notes as well, via OLE Automation.

Language Syntax Comparison

This section contains a comparison of the statements and functions making up LotusScript and Visual Basic.

Other References

This section covers the low-level syntactical language differences between LotusScript and Visual Basic. Much of the material in this section will be covered at a level just high enough to explain the basic concepts involved. If you wish to read more about the language specifics, see the *LotusScript Language Reference* (also available within the Notes Help database) and the *LotusScript Programmer's Guide.* The LotusScript manuals are available as a documentation pack from Lotus: they are not packaged inside Notes boxes. See the *Visual Basic 4.0 Language Reference* for more on the Visual Basic language.

The respective web sites of Lotus (http://www.lotus.com/devtools) and Microsoft (http://www.microsoft.com) will offer the most up-to-date material and announcements about each language and their supporting tools.

There are many good examples of how to extend the use of LotusScript by adding LotusScript Notes classes found in the Notes Programmer's Guide, Part 1 - Chapter 4.

Data Types

The data types supported under LotusScript and Visual Basic are very similar.

Typeless or Typed variables

Declaring the type of data that a variable holds is not strictly a requirement in the typeless, variant data type declarations possible in both LotusScript and Visual Basic. While you are able to specify what type of data a variable holds, in both languages you can also reserve a 16-byte piece of storage by simply using the variable without a type. You then assign the "type" of data for this variable whenever you first load actual data into the variable. For example in the first block of code, you'll notice that UserID is used without first being declared.

```
Dim rc                                  ' dim a return code as
variant
UserID = "Bob Sands"    ' a variant is created and 'typed'

' as a string"
rc = Int(UserID)        ' this will fail at run time since

' since UserID is now a string; which

' is an invalid parm for Int( ).
```

whereas the following code will both work and will actually catch the error at design time when compiled, not just at run time:

```
Dim rc                  ' dim a return code as variant
Dim UserID#                             ' a double is created
UserID = 22.34          '
rc = Int(UserID)        ' this will work as expected.
UserID = "Bill Sands"   ' and this will be caught at design
                        ' time.
```

Both LotusScript and Visual Basic provide a statement which, when entered in the general declarations sections of the code, will force the developer to explicitly declare the type of data of a variable when declared. The LotusScript equivalent of Visual Basic's **Option Explicit** is **Option Declare**, which forces the developer to declare all variables before they are used.

Data Type	*LS*	*VB*	*Suffix*	*Storage Size*	*Range*	*Notes*
Byte		**X**	none	1 byte	0 to 255	
Integer	**X**	**X**	%	2 bytes	-32,768 to 37,767	
Long (long integer)	**X**	**X**	&	4 bytes	-2,147,483,648 to 2,147,483,647	Short integer (2 byte whole number). In 32-bit Windows, VB4 represents an int with a 4 byte number.
Single (floating point)	**X**	**X**	!	4 bytes	-3.402823E38 to -1.401298E-45 or negative values; 1.401298E-45 to 3.402823E38 for positive values.	
Double (floating point)	**X**	**X**	#	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.	
Currency (scaled integer)	**X**	**X**	@	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.	
String	**X**	**X**	$	2 bytes per character*	10 bytes + string length	0 to approximately 2 billion (approximately 65,400 for Microsoft Windows version 3.1 and earlier)
Variant	**X**	**X**	none	16 bytes		
user-defined	**X**	**X**	none	As determined by its elements		
Date		**X**	none	8 bytes	January 1, 100 to December 31, 9999	
Boolean		**X**	n/a	2 bytes		True or False

LotusScript only allows arrays and lists up to 64k in size. Providing a way around this restraint should provide a nice after-market for Lotus add-in component builders.

Visual Basic and LotusScript support UNICODE strings. Unicode allow you to handle complex languages such as Japanese and Chinese using DBCS (double-byte, or more, character set) encoding. These languages require more than one byte of storage to represent all possible language characters. However, this requires the allocation of an extra byte (or more) of storage for each one desired by the programmer. This also requires separate sets of functions to "see" the number of bytes or characters expected.

For example:

LEN (str$**)** Returns the number of characters in a string, or the number of bytes used to hold a numeric value.

LENB (str$**)** Determines the length of a string in bytes rather than in characters.

The following code demonstrates the difference in returning byte counts or character counts for a string. Notice that the numeric expression returns 4 bytes (the size allocated for a single data variable) in each case.

```
Dim s As Single
Dim strTemp As String * 20
strTemp$ = "hello"
s = 12345

Msgbox(Format$(Len(s)))     ' returns 4
Msgbox(Format$(Len(strTemp$)))  ' returns 20
Msgbox(Format$(Lenb(s)))     ' returns 4
Msgbox(Format$(Lenb(strTemp$))) ' returns 40
```

Defining Default Variables

Both LotusScript and Visual Basic provide a way to determine default ranges for integer and other variable types. For example, saying DefInt a-z in your code will result in single letter variables from a to z being "typed" as integers when used, even if not explictly declared with **dim x as integer** or dim x% statements in your code.

You can set up one-character variables to automatically represent the other basic datatypes, thereby avoiding possible portability problems when writing your code. For example, DefCur defines Currency variables in the same way DefInt defines Integer variables.

DefCur A-D sets the characters A, B, C and D to be typed as currency data types automatically. As with DefInt, subsequent reference to A through D will be treated as currency types. Notice that character settings are case sensitive; A-Z and a-z represent two possible sets of variables that can be defined through these DefType statements.

Collections

Collections is a Visual Basic Release 4 addition that allows you to set up array-like constructs; it can hold multiple types of data elements including class-based objects that you create yourself. Collections is handy in Visual Basic and has a number of operations (such as .Add, .Delete, etc.) that can be performed directly against the collection, but collection processing can be somewhat slower than processing against more traditional array constructs.

The Visual Basic Collection keyword is not supported in LotusScript. Using the line **Dim myUsers as New Collection** results in a "Class or type name not found: COLLECTION" message. LotusScript handles collections as a Notes class. Visual Basic collection-based code will need to be changed to array-based logic when ported to LotusScript.

Operators

Both Visual Basic and LotusScript share the same + - / * ^ \ MOD operators, which have the same precedence. The Visual Basic XOR, AND, NOT, OR, EQV and IMP operators are also found in LotusScript and support bit-wise operations. Both languages provide the Like pattern-matching operator for use on strings.

Commands

LotusScript uses most of the same syntactical statements and structures that you will find in Visual Basic. A very short list of LotusScript supported statements and functions includes: **Fix**, **FreeFile**, **GetObject**, **Hour**, **InStr**, **IsDate**, **Kill**, **LBound**, **Mid**, **Option Base**, **ReDim**, **Seek** and **Val**. **Now** even returns the system date and time in the same format as Visual Basic.

Both LotusScript and Visual Basic support multi-line statements, using the '_' character to end the line, preceded by white space.

LotusScript uses the C-like %Rem...%End Rem statements to comment or uncomment more than one contiguous line at once.

File I/O

Code such as the following, to open and convert an ASCII text file to UNIX format, works exactly the same in both LotusScript and Visual Basic:

```
Dim i%
Dim j%
Dim strChar$
Dim lLOF As Long
Dim iWinFilePtr%
Dim iUnixFilePtr%
        ' ... skipping the read from input file in here...
 Open strFilename & ".htm" For Output _
   Access Write As #iUnixFilePtr
 lLOF = LOF(iWinFilePtr%)
 For j% = 1 To lLOF
  strChar$ = Input(1, #iWinFilePtr%)
  If Asc(strChar$) = 13 Then
    Beep
  Else
    Print #iUnixFilePtr%, strChar$;
  End If
 Next j%
 Close iWinFilePtr%
 Close iUnixFilePtr%
```

LotusScript and Visual Basic share the ability to Lock and Unlock files processed in this way. This ensures that the records you are trying to process are not overwritten by others while you are working on them.

On a binary file, you can lock a record at a time. In an ASCII file, you lock and unlock the entire file at the same time. Lock Read, Lock Write, Lock Read Write are available in both Visual Basic and LotusScript as parameters to the **Open** statement, to control locking when initially opening a file.

Other Recent Language Additions

Recent VBA and VB4 syntactical enhancements have been included in the current release of LotusScript. Statements like **With** and **ForAll** are found in LotusScript. As an example, **ForAll** allows you to easily do something to every item within a LotusScript collection, such as a list or an integer array:

```
dim iSalary(20) as integer
forall x in iSalary
  x = x * 1.5 ' let's give them a raise they won't forget!
end forall
```

Other interesting statements include **Datatype** (variable) which returns the datatype associated with the object you pass it. Datatype is similar to Visual Basic's **If TypeOf** statement, but requires a lot less work.

Note that the LotusScript **End** statement is not directly equal to Visual Basic's **End**. The **End** statement in LotusScript stops the currently executing script, and to exit the application in the way you would with Visual Basic's **End**, you generally use the appHalt function. If you are within Lotus Notes, you can use the Close macro to easily (and safely) shut the application down.

Error Handling

LotusScript provides the same type of error handling syntax as Visual Basic Release 4, allowing for Err, Erl, Error, Resume, Resume Next, On Error Goto label, and so on.

For example, the following code can be used in both LotusScript and Visual Basic.

```
Sub cmdUpdateDB_Click ( )
            On Error GoTo cmdUpdateDBErr:
            ' process database updates here...
            Exit Sub
cmdUpdateDBErr:
            ' The HandleErr( ) call provides a way to
            ' centralize your error processing. The code for
            ' the HandleErr call is shown in the
            ' object-oriented section below.
            If HandleErr(Err, Error$, _
```

```
            "frmMain::cmdUpdateDB_Click") then
            resume
        Else
            resume next
        EndIf
End Sub
```

The code for the HandleErr function is broken out and defined for you in the Object-Oriented programming section below.

On...GoSub

On...GoSub exists in both variations of Basic as well. On...GoSub allows you to jump (and later return from) different routines using a Select Case type of structure. All of the labels referenced must reside within the same procedure.

```
On x% GoSub Label1, Label2, Label3
Label1:
 ' do stuff
 return

Label2:
 ' do stuff
 return

On...GoTo
```

Conditional Compilation

Conditional compilation provides a way for developers to code in platform or similarly specific sections of code, with only the code pertaining to the runtime platform actually being compiled when executed. The following LotusScript **If** statements, preceded by "%" character, are used for conditional compilation.

```
%If
%ElseIf
%End If
```

The above statements provide conditional compilation support and work in the same way as the #If ... #End If statements do within Visual Basic 4.0. You follow the %If or the %ElseIf with a constant that is provided by the language or declared explicitly by the developer.

Visual Basic allows you to define your own constants using a #Const compiler directive. This allows you to add #If DEBUGGING type of logic in the application, providing a way to compile out all debugging code simply by removing a #Const DEBUGGING line.

Otherwise, both LotusScript and Visual Basic come with preset constants to use with conditional compilation statements. Code written like this shows the Win32 and OS/2 conditional compilation constants in use:

```
%If Win32
  messageBox("Running on Win32")
%ElseIf
  %If OS/2
   messageBox("Running on OS/2")
  %End If
%End If
```

This will display a message box telling the user what platform the code is running on. The unreachable code is not executed at all.

Lotus recognizes more platforms than Microsoft as being legitimate porting possibilities. Microsoft supports Win32 and Win16 as constants provided by the language. In LotusScript you will find compiler directives for almost everything from SOLARIS to OS/2. Lotus also adds the MAC as a real alternative to Win"xx" as well. Using LotusScript within Notes provides a way to build truly cross-platform, portable applications.

The %If, %End If statements are not provided in Lotus Notes.

MessageBox

Visual Basic developers will note the **MessageBox** statement in the preceding code. LotusScript defaults to using MessageBox rather than MsgBox, although the latter command actually works as well in Lotus Notes. Ported MsgBox statements will work, using the parameters specified for MessageBox.

Visual Basic's message box statement adds support for two new parameters; a helpfile file name and helpcontext id. These are used to refer to a specific help topic within the named helpfile, if the user pushes **F1** for help over the message box at run time.

LotusScript allows you to make the MessageBox dialog an application modal or system modal dialog. With Visual Basic, it is always an application modal dialog.

Constants

Both LotusScript (with LTSSL30.LSS and LSCONT.LSS) and Visual Basic (WIN16API.TXT and WIN32API.TXT) provide a number of platform specific constants and Application Programming Interface (API) declarations in a standardized format. Visual Basic extends this idea by bundling a number of the constants into the development environment, providing several useful features:

- Teams do not have to guess at a naming sequence for constants as they add them to their projects.
- Code becomes easier to share.
- The space wasted by unused constants is eliminated.

Examples of this include the vbHourglass and vbYesNo internal constants.

Errors, Error Constants

Both Visual Basic and LotusScript define constants that map to trappable internal error codes, that may occur at run time. Both languages allow you to handle system, OLE and data access errors using code-based error handling routines at run time. The Basic **On Error** statement provides the means to trap for these errors. Having both languages represent the same type of internal errors with the same numbers is an important step in being able to share code from one of the BASIC implementations to the other.

As an example, a "Device Unavailable" error occurs if your application tries to write to a drive that is not ready. This will happen if you try to copy files to the floppy A: drive without a diskette in this drive. Rather than simply displaying a system error message and ending the application, Visual Basic and LotusScript treat this as a trappable error, and each gives the developer

the chance to deal with it by adding an error handler. If the developer does not add the error handler, the application will then display the system error message and most likely terminate the application.

Visual Basic represents the Device Unavailable error with an internal number 68. LotusScript also represents this error with the internal number 68. Most of the system-oriented errors that are trappable errors in Visual Basic can be handled by LotusScript as well, using the same numbers.

LotusScript provides an ASCII file LSERR.LSS, that defines the system errors it allows you to trap. Visual Basic provides a list of system, Access and OLE errors that it allows you to trap on as part of its online documentation.

There is a general guideline to how the trappable errors are grouped. The Visual Basic error groupings are:

1-94	System Errors
260-428	DDE or Form Errors
429-451	OLE Automation Errors
444-521	System Errors
2400-3622	Data Access Errors
4000-	Notes Container Errors
31000-31050	OLE Container Errors

Lotus maps a number of the trappable errors for you into constants declared in the LSERR.LSS file. A snippet of the errors defined within this file are shown in the following listing:

```
'          Run Time LOI Errors
Public Const ErrFileAlreadyExists     = 58
Public Const ErrBadRecordLength      = 59
Public Const ErrDiskFull         = 61
Public Const ErrInputPastEndOfFile    = 62
Public Const ErrBadRecordNumber      = 63
Public Const ErrBadFileName        = 64
Public Const ErrTooManyFiles       = 67
Public Const ErrDeviceUnavailable     = 68
```

Visual Basic provides the ability to Raise an error for an object, which allows you to handle the error through external applications or OLE servers. The code for this looks like this:

object.Raise(Number, Source, Description, HelpFile, HelpContext)

and works in a similar fashion to using the **Error** errNumber statement within Basic code. The system believes that the error has occurred once you issue the **Error** statement or **.Raise** method for an object, with all expected error traps enabled as well.

Note that you can generate your own application-level errors in both Visual Basic and LotusScript, by using the **Error** statement with an unused error number. For example, if you have an appropriate **On Error Goto** trap in a section of code, the statement **ERROR 147** will generate a user-defined error in both languages.

In Visual Basic you'll receive the error string "Application-Defined or Object-Defined Error". In LotusScript the error message generated would be "User-defined error".

This allows you to define errors that can be handled the same way, sharing the user-defined error numbers across multiple applications.

Extending the Code

Anyone developing complex applications knows that you will need more than just the BASIC language syntax itself, in order to add complex or custom processing to your program. Usually this means accessing the underlying environment's API calls, accessing external data engines or using the many custom controls available from third-party companies.

Application Programming Interface Calls

Using API calls allows you to extend your programming reach into the system-oriented functions, available in the underlying operating systems. Note that once you code an API call in a language like LotusScript or Visual Basic, you have added complexity as well as reliance on the presence of the underlying service. LotusScript and Visual Basic differ on how APIs are enabled in the respective languages.

32-bit issues

An example of this can again be found in the **WriteProfileString** API call. In Windows 3.1x, this API call places a string into the WIN.INI ASCII parameter file. GetProfileString can be later used to retrieve the string.

If you move your code to Win32 (Windows 95 or Windows NT) the API call no longer works. A new API call GetProfileStringA has been created to handle the UNICODE requirements of strings under the 32-bit platform.

New Visual Basic Statements

In its most recent release, Visual Basic now supports GetSetting, SaveSetting and DeleteSetting commands that provide access to the registry on a 32-bit platform (in order to save parameters) and to standard INI files on 16-bit platforms.

Wrapper Examples

Lotus provides LotusScript wrappers around a number of common APIs, such as the Win16 **WriteProfileString**, in an .LSS file called LTSSL30.LSS. If you desire to use the WriteProfileString, GetProfileInt, GetProfileString API calls directly in Visual Basic, you will need to provide wrapper functions similar to those provided in the Lotus LTSSL30.LSS file.

```
'_____________________________

' GetProfString

' This function returns a profile string from the specified

' ini file. If the filename passed is "", then the string
will

' be searched for in the WIN.INI file

'_____________________________

Public Function GetProfString(Section as String, Entry as
String, Filename as String, DString as String) as String

Dim retstr as String*256

Dim retval as Integer

If filename = "" then

  retval = GetProfileString(Section, Entry, DString,

           retstr, 256)

Else

  retval = GetPrivateProfileString(Section, Entry, DString,

           retstr, 256,Filename)

End If

GetProfString = Left$(retstr, retval)

End Function
```

Lotus Components

You can add Windows OCX controls into Lotus Notes today. They will visually react when clicked on, but you cannot easily react to events fired within the control itself. For example, you will see the arrow depress if you add a spin button to Notes and click on the arrow. But there is no easy way to add code to handle this event when it occurs.

Lotus Components Software Development Kit (SDK) is the way Lotus will provide to support controls within Notes. The SDK enables you to build controls.

Automation Servers

```
Sub Click(Source As Button)
   Dim session As New NotesSession
   Dim db As NotesDataBase
   Dim doc As NotesDocument
   Set db = session.currentDatabase
   Set doc = New NotesDocument (db)
   doc.Form = "Memo"
   doc.Subject = "Look at the attached code for the " & _
              "way to mail information within code ..."
   doc.Memo = "This code explains how to use " & _
                            "automation to create a mail
message."
   Call doc.Send (False, "Shadish")
End Sub
```

Execute Text

Execute allows you to create a temporary module, at run time, that executes and is released when done. Vertical bars | | or { } are used to allow the text to execute to span several lines.

The Execute doc.txtScript line below pulls script from a text box (txtScript) on the form and executes it at runtime.

```
Sub Click(Source As Button)
   Print "Starting Script"
   Dim ws As New NotesUIWorkspace
   Dim uidoc As NotesUIDocument
```

```
    Dim doc As NotesDocument
    Set uidoc = ws.CurrentDocument
    Set doc = uidoc.Document
    Execute doc.txtScript
    Print "Completed Script"
End Sub
```

Other Interesting LotusScript Commands

The **Width** #filenumber, Width statement forces a set ASCII file width, with autowrap to the beginning of the next line when the width has been exceeded.

The **YIELD** statement is equivalent to doEvents in Visual Basic 4. And the Yield statement requires a return value, unlike the doEvents in Visual Basic, but equivalent to Visual Basic's doEvents function.

The TypeName (strFileName$) statement will return the word "STRING". This allows you to determine the datatype of a variable at run time, which is useful when you have passed variables that may contain any type of data in them.

LotusScript "feels" a little more like C++ than like BASIC sometimes. Beyond the standard BASIC keywords, LotusScript provides for additional syntax such as "Declare"ing a forward reference to a function or subroutine. This allows you to code and test the sub or function before it's actually included in your main app — something that's particularly useful in team development environments. Developers familiar with Object-Oriented programming will quickly find themselves creating and instantiating objects using new (constructors), delete (destructors), and data-hiding features you would expect to see in C++.

On Event Click from <object> **call** <routine name> allows you to react to events happening in other Lotus products. For example, this would allow you to create a page-turning presentation to use in a Lotus Notes training application.

Evaluate (Macro, object) allows you to easily call another (Lotus only) product macro. This allows you to pass a string of numbers to a spreadsheet to be totaled, etc.

The %**Include** command gives you the ability to include LotusScript variable, constant and API declarations into your code from external ASCII files. This allows you to sort through all of the various files provided by

Lotus, creating a cut-down version that contains just the statements that you need. You can then add all of this to a new application with one line of code: %**Include "MY-DEFS.LSS"**. This behaves the same as if you were using <include mydefs.h> files in C or <include mydefs.hh> files in C++. The %**Include** command can only be used in the general declarations sections of an object; not within the event code itself.

Uni (strExp**)** and **UChr (**longValue**)** functions work together like Visual Basic's **Asc** and **Char** statements, but for Unicode characters. **Uni ( )** returns the Unicode numeric character code for the first character passed as the string parameter. **UChr ( )** returns the character represented by the long number passed as an argument.

Using the **Print** stringRef command in LotusScript by itself will place the string in the last message status bar area of Lotus Notes, as is shown in this figure.

This makes for a useful display area when debugging your program and is a good place to drop critical run time messages if you are writing code for agents, monitors or other background tasks.

Other Interesting Visual Basic Commands

Visual Basic supports Named and Optional parameters in its subs, functions and property procedures, allowing you to specify the parameters in any order and to choose which parameters you actually want to call the routine with.

Visual Basic allows you to create OLE Servers (EXEs and DLLs in the 32-bit version).

Object-Oriented Programming (OOP)

Object-oriented (OO) development features include inheritance, polymorphism, data encapsulation and creating classes.

Classes

A class can be thought of as a template that can be used to create multiple instances of objects at run time. Each object contains its own copy of data, which can be changed via properties provided externally to the user (or

developer). The objects also contain code, which is used to manipulate or inspect this data at run time.

A very simple example might be to create a class to display a message. We'll do this in LotusScript by creating a class called DisplayMsg. In the following listing, the DisplayMsg class is shown, where you will see a property called *Text* and a method called *Display*.

```
Class DisplayMsg
   ' Declare member variables in a LotusScript class
   mMsg As String
   mMsgNumber As Integer

   Public Property Set Text As String
     mMsg = Text
   End Property

   Public Sub Display
     Msgbox(mMsg)
   End Sub
End Class
```

The text for the class is entered into the declarations of a Notes form. The code in the next listing is entered into the Sub Click event of a button on this Notes form. Here you will see a variable created as a variant called objMsg, which will be used to hold the instance of our message object. The set objMsg = New DisplayMsg line actually creates the instance of the object and loads a reference of it into the objMsg variable.

```
Sub Click (Source As Button)
   Dim objMsg as Variant
   Set objMsg = New DisplayMsg
   objMsg.Text = "(" & Format$(68) & ") - " & Error$(68)
   objMsg.Display
   Delete objMsg
End Sub
```

Once you have created the object, you can set the text property by referring to it as object.property, or objMsg.Text in this case. You can cause the code in the Display method (function) to execute using the same process.

While this is a simple example, you might easily imagine additional possibilities here. The message may actually be "displayed" on a pager, or logged to a database, all by adding more code to this basic structure. The mechanism to create the object, load its properties and call its methods remains the same.

LotusScript will allow you to use most of the most important OO constructs, like: Building classes, Private and Public data within objects and instantiating an object (stored outside your application) from within a running application.

LotusScript also supports single inheritance. LotusScript Notes classes support inheritance with the ability to create classes and then subclasses of objects. Note that the embedding product (in this case, Notes) exposes the products' functionality as a set of classes. It is these "product classes" that can be inherited into your own private classes.

Visual Basic supports all of the object-oriented features above, with the exception of inheritance.

Visual Basic Class Modules

Visual Basic classes are handled through a new "class" of code module, a .CLS file. The CLS file is called a Class file. Class files behave like .BAS code modules in structure. The code within the class is used by creating an object at run time and then referring to the object's methods and properties.

Special code is executed when you create an instance of the object. Similar code is executed when you delete the object. Visual Basic classes have a Class object, which holds Class_Initialize and Class_Termination events. Whenever you create a new object, the initialization event is executed. When you delete an object, either by leaving the scope where the object was created or explicitly by saying **Set objRef = Nothing**, the termination event is executed.

LotusScript Notes Classes

Classes within LotusScript are added inline, directly within the code modules themselves. Like Visual Basic, LotusScript provides initialization and termination procedures within the classes; in this case, called Sub Delete and Sub New and Sub Delete. Like Visual Basic, code within these procedures is executed automatically. Sub New is executed whenever you create an object from the class. Sub Delete is executed whenever you destroy the object using the Lotus **Delete** *object* statement.

The following example shows how to create and use a LotusScript class to create an error handling object. This will demonstrate the use of the ***HandleErr*** ( ) function in error processing.

We'll show the class ErrObject, which will be used to hold and display errors. This code was ported from Visual Basic and the comments following the class describe the changes that were made to allow the port.

This is followed by the actual HandleErr function call; that would be called by the code in our error-handling discussion earlier.

```
Class ErrObject
   Private mErrLogFile As String
   Private mErrNumber As Long
   Private mErrMsg As String
   Private mErrLocation As String
   Private mErrLogMethod As Integer
   Private mErrSeverity As Integer ' change dim to private



   Public Property Get ErrMsg
     ErrMsg = mErrMsg
   End Property

   Public Property Get ErrNumber
     ErrNumber = mErrNumber
   End Property

   Public Property Set ErrNumber ' changed let to set
     mErrNumber = ErrNumber
   End Property

   Public Property Set ErrMessage ' changed let to set
     mErrMsg = ErrMessage
   End Property
```

```
Public Function PostError(ErrNumber As Variant,
 ErrLocation As Variant, LogMethod As Variant) As Integer
' pulled the optionals and boolean
  PostError = RESUME_NEXT_AFTER_ERR

' use object properties if parameters were ommitted.
  If Isempty(ErrNumber) Then ErrNumber = mErrNumber
  If Isempty(ErrLocation) Then ErrLocation = _
        mErrLocation
  If Isempty(LogMethod) Then LogMethod = mErrLogMethod

' build the message. Note, DB, file output may differ
' slightly, as Date/Time, Err, etc may be written to
' separate columns (fields)

 mErrMsg = Error$(ErrNumber) & " (#" & _
  Format$(ErrNumber) & ") " & "occurred at " & _
  Format$(Now, "mm-dd-yy hh:mm:ss") _
  & "; at location->" & ErrLocation & "."

  Select Case LogMethod
  Case LOG_DISPLAY_MESSAGE
     Beep
     Msgbox (mErrMsg)

  Case LOG_TO_DISK
     Dim iLogPtr%
     iLogPtr% = Freefile
     Open mErrLogFile$ For Append Access _
           Write As #iLogPtr%
     Print #1, mErrMsg
```

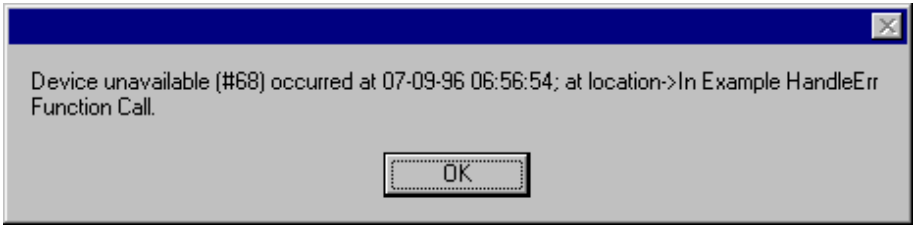
```
            Close #iLogPtr%

         Case LOG_TO_DB
            ' not implemnented
         Case LOG_SEND_TO_PAGER
            ' not implemented
         Case LOG_SEND_EMAIL
            ' not implemented
         Case Else
            Msgbox (mErrMsg)
         End Select

         Select Case ErrNumber
         Case 5 ' invalid procedure call
            ErrSeverity = RECOVERABLE
            PostError = SHUTDOWN_AFTER_ERR

         Case 68 ' device not available (like when user
                 ' accesses A: without a diskette. Show a
                 ' message and then resume.
            ErrSeverity = SHUTDOWN_APP
            PostError = RESUME_AFTER_ERR

         Case Else
            ErrSeverity = RECOVERABLE
            PostError = RESUME_NEXT_AFTER_ERR
         End Select
      End Function
   End Class
```

In moving the code from Visual Basic to LotusScript, we make the following slight modifications.

1. Consts move out of the class into the scope of the object the class is in.
2. Dim statements within the class change to Private.
3. **isMissing** is not provided. Replace isMissing with **isEmpty**.
4. Optional arguments are not available. Replace Optional with explicit variable declarations.
5. For property functions, Set is used, rather than Let. Set does not have the special (but rarely used) object passing features that are present in Visual Basic.

```
Function HandleErr (iErr%, strErrLocation$, iAction%) As
Integer
   On Error Goto MetaErr
   Dim objErr
   Set objErr = New ErrObject
   rc = objErr.posterror (iErr%, strErrLocation$, iAction%)
   Delete objErr
   Exit Function
MetaErr:
   ' An error here might come from a failure to create the
object.
   Msgbox(Format$(Err) & ", " & Error$(Err))
   Resume Next
End Function
```

The only real change in moving HandleErr from Visual Basic to LotusScript is that the object is destroyed explicitly with the **Delete** objErr statement. Notice that the MsgBox call comes across without change. Using HandleErr () results in a message box that looks like this:



CreateObject, GetObject

One advantage for both language dialects is that external OLE servers can be added into your applications using these statements. This exposes the full power of OLE automation servers, like Word Pro and Notes, to applications written in BASIC.

Mail Enabling

Fortunately, the ability to create and send e-mail is fairly easy in both LotusScript and Visual Basic.

In LotusScript, to send a message in Notes requires the following type of code:

```
Sub Click (source as Button)' a button to send info with...
  dim session as new NotesSession
  dim db as NotesDataBase
  dim doc as NotesDocument
  set db = session.currentDatabase
  set doc = new NotesDocument (db)
  doc.Form = "Memo"
  if txtSubject = "" then
          doc.Subject = "Here is a quick thought ..."
  else
          doc.Subject = txtSubject.text
  endif
  if txtMemo = "" then
          doc.Memo = "Example Text"
  else
          doc.Memo = txtMemo.text
  endif
   Call doc.Send (False, "Shadish")
End Sub
```

In Visual Basic sending a message is equally easy, as a mail control is available that embodies much of the mail system's features. The following Visual Basic code demonstrates a MAPI-based mail transfer. If you wish to handle cc:Mail or pass through a generic VIM layer, to move mail from a Visual Basic system to a VIM-based system, third-party controls are also available to do this. After dropping the mail control on your form, enter this code behind the send button:

```
Sub cmdSend_Click
  frmMain.MapiMess.Action = MESSAGE_COMPOSE
  frmMain.MapiMess.RecipDisplayName = Address$
  frmMain.MapiMess.Action = MESSAGE_RESOLVENAME
  if txtSubject = "" then
          frmMain.MapiMess.MsgSubject = _
            "Here is a quick thought ..."
  else
          frmMain.MapiMess.MsgSubject = txtSubject.text
  endif
  if txtMemo = "" then
          frmMain.MapiMess.MsgNoteText = "Example Text"
  else
          frmMain.MapiMess.MsgNoteText = txtMemo.text
  endif
  frmMain.MapiMess.Action = MESSAGE_SEND
End Sub
```

Code Sharing Concerns

As far as the languages' syntax itself, this is not a major area of concern. Much of the code written for Visual Basic should be able to move relatively easily into LotusScript.

There are some things that might need to be pointed out.

Close but Different

Obviously, the code associated to user interface elements will need to be reviewed and most likely rewritten. The way controls are handled in Lotus

Notes differs significantly from Visual Basic. It is easier to change the properties of controls in Visual Basic when compared to LotusScript. Different Windows events are provided for the same controls and the names of these events are also sometimes different.

For example, a button in Visual Basic has the following events:

```
Click
DragDrop
DragOver
GotFocus
KeyPress
KeyDown
KeyUp
LostFocus
MouseDown
MouseMove
MouseUp
```

In addition, you can cause the Visual Basic Button_Click event to execute by setting the implied .Value property of the button to true, as in **cmdCancel = True**.

In LotusScript, the events provided for a button are:

```
Click
Terminate
Initialize
ObjectExecute
Options
Declarations
```

Notice the ability to include declarations directly within an object, such as a button.

Where LotusScript commands and syntax are ***like*** Visual Basic's (for example, the %If ... %Endif as compared to the #If ... #End If conditional compilation commands), they will need to be converted, redone or not used in order to create code that is truly portable between Visual Basic and LotusScript.

Unlike Visual Basic, LotusScript is not centered around the Properties, Methods, and Events metaphor. LotusScript is built around interacting with the objects provided within the respective host application and development tools provided by Lotus. For example, objects don't have Tags or Captions. A static text object uses a text property rather than a caption property to read and change its value. So any Visual Basic code that refers to captions for labels needs to be changed. Similarly, LotusScript doesn't support the tag property. But LotusScript does have a Name(variable) function that returns an item's name, which can then be used to operate on objects dynamically at runtime.

Control Arrays

LotusScript does not support control arrays or indexes on individual objects in the same way as Visual Basic. So your Visual Basic code that sets up an array of text objects with a common code routine — as with Sub txtAmtField_Click (amtIdx as index) — will need to be rewritten. Buttons under LotusScript are set up as individual buttons or to act as group or checkbox buttons when they are placed onto the screen.

Visual Basic Environment Constants

If you are using any of the new Visual Basic embedded constants, you will need to provide a translation file of codes for anything that you want to move to LotusScript. Or you can simply choose not to use the Visual Basic constants in the first place. For example, you might be using things like this in Visual Basic 4 applications now:

```
If rc = vbYes or rc = vbAbort Then
 ' add processing here.
EndIf
```

You will need to add constant declarations for vbYes and vbAbort in code ported to LotusScript; otherwise the references to vbYes and vbAbort would prove invalid.

Caution If you are not using Option Declare in your LotusScript or Option Explicit in your Visual Basic modules and move the code shown above, it may compile and give you erroneous results. Both vbYEs and vbAbort would be treated as uninitialized variables and both contain a 0. The code

would execute but give incorrect results since you would always compare whatever the contents of rc was to zero.

```
Const vbAbort = 3

Const vbRetry = 4

Const vbIgnore = 5

Const vbYes = 6

Const vbNo = 7
```

Summary

It's probably safe to say that a lot of your Visual Basic calculation code will port across to LotusScript, but that your interface code will not move as easily.

LotusScript is enough like Visual Basic's BASIC language implementation that you will have few problems porting code from one tool to the other — if you recognize these differences before building any code. With a little care, you will be able to easily create routines that can be called from either Visual Basic or LotusScript without change.

The problem for the Visual Basic developer writing code to share with LotusScript will not be what cannot be moved into LotusScript. Rather, it will be in deciding not to use some of the appealing LotusScript extensions that are now available — like class inheritance, the include% features and some of the LotusScript Notes classes.

Chapter 2
The Notes Integrated Development Environment

Introduction

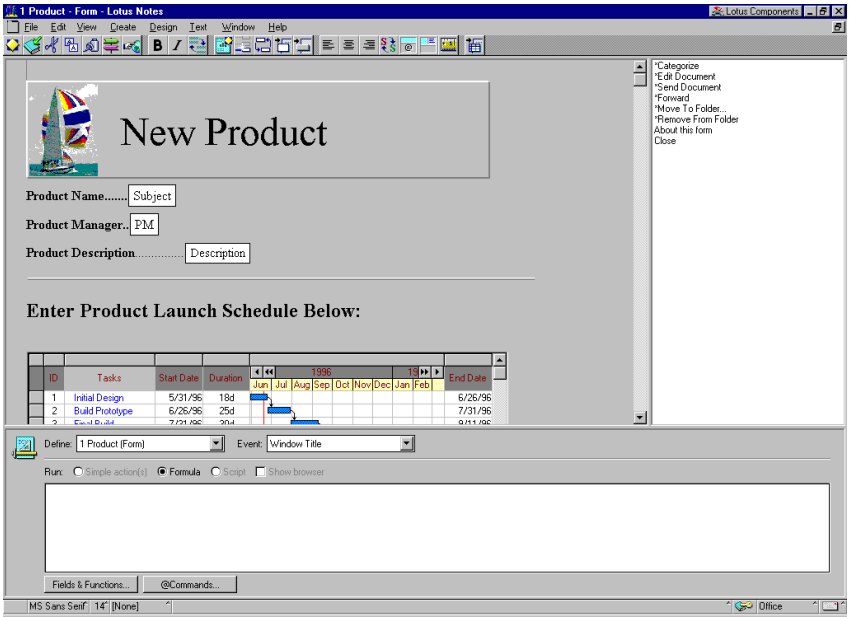
Lotus Notes provides a complete application development environment. A Notes application consists of several or all of the following:

- Forms — which provide the templates through which data in the application is entered and displayed. Unlike a traditional template, forms can also act on the data. For example, when a user inputs information in the form, the form might, depending on the contents, send an e-mail message to another user.
- Views and folders — which provide different ways of looking at all or part of your data, according to specified criteria. A view might be thought of as similar to a report in a traditional database program, except that the view is dynamic and includes links to information in the application.
- Navigators — which provide graphical means of moving between views.
- Agents — which add functionality to the application. For example, you might create an agent that once a day scans the documents in the database, checks the contents of certain fields, and places documents that meet specified criteria into a special folder.

There are, within Notes, Integrated Development Environments (IDESs) for developing each of the above named elements. These IDEs share many common elements. For example, the tools for writing LotusScript are identical among all of the IDEs. We will focus on the forms IDE since it is the most complex, and therefore the most interesting one. Keep in mind, though, that much of the discussion in this chapter applies to the development of all Notes elements, not just forms.

Elements of the Forms Integrated Development Environment

The picture below illustrates the IDE you use when creating and editing forms in Lotus Notes.



The three components of the Integrated Development Environment are represented in the picture.

Main Design Window

This is the large window at the top left of the screen where you visually design your form. Using this window you can add static text, fields, layout regions, and embedded objects to the form.

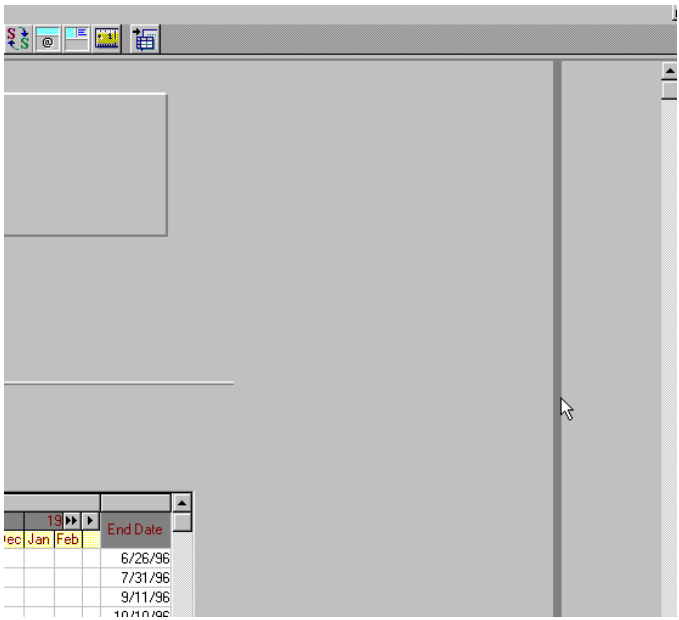
To work in this window you place your cursor at the desired position on the form and type your text, or use the menus or SmartIcons to insert the desired object. When you are working in this window the status bar at the bottom of the screen provides controls you can use to quickly format text.

Action Pane

This is the narrow window at the top right of the screen. The action pane is used to define actions that are associated with the form. An action is a LotusScript procedure, simple action, or macro that performs work when activated. Actions can be invoked from the menu bar, the action bar, or by LotusScript procedures.

The action pane is not visible in the Integrated Development Environment by default. To view the action pane, do one of the following:

1. Choose View → Action Pane, or
2. Drag the right-hand edge of the main design window to the left, as shown in the following figure, until the action pane is the size you want.



To hide the action pane you can either drag its left border all the way to the right, choose View →Action Pane to uncheck the menu option, or click on the appropriate SmartIcon.

Design Pane

The design pane is the window below the main design window and the action pane. This is where all the programming work in Notes, be it using simple actions, the formula language or LotusScript, takes place.

The design pane is, by default, visible when you create a new form or open one for editing. You can also drag the border between the main design window and the design pane to change the size of both, or to hide the design pane altogether. You can also show and hide the design pane by choosing View → Design Pane, just as you do to show and hide the action pane.

The design pane is also used when you are creating or editing a database or view action.

There are a number of components to the design pane. Let's look at each one of them.

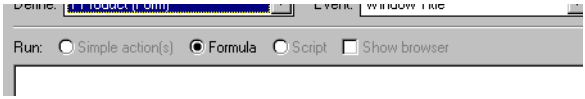
Define Box

The define box is a combo box that shows you all of the objects on your form that can be programmed, along with all of the actions defined for the form.

Event Box

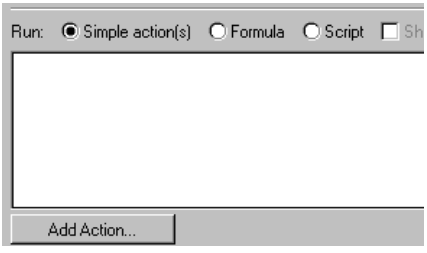
This combo box shows all of the programmable events for the object showing in the define box. Each object has its own set of events, so the contents of this box will change in accordance with the object specified in the define box. There are also some cases where no events are available, in which case the event box is not shown. Also, LotusScript procedures or |subs| which you add to your Notes application will appear in this list.

Run Area

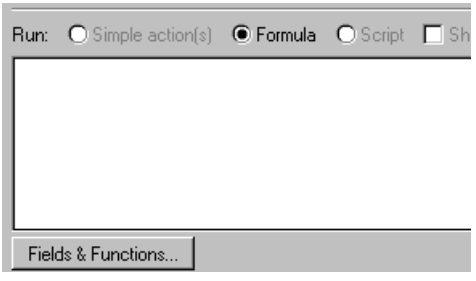


The three radio buttons specify the type of programming that you will apply to the specified object and event. Choose one of the following:

- Simple Action(s): Lets you easily specify one or more actions from a number of pre-defined actions, such as Modify Field, Send Document, Move Document to Folder, etc. When you specify Simple Action(s), an Add Action button (see following figure) appears at the bottom of the design pane. Clicking this button brings up a dialog box which allows you to specify the action you wish to perform. Not all objects on the form support simple actions. If an object which does not support simple actions is selected in the define box, this radio button is disabled. To edit an existing simple action, select it with the mouse and then click the button; the button changes to Edit Action when the action is selected.



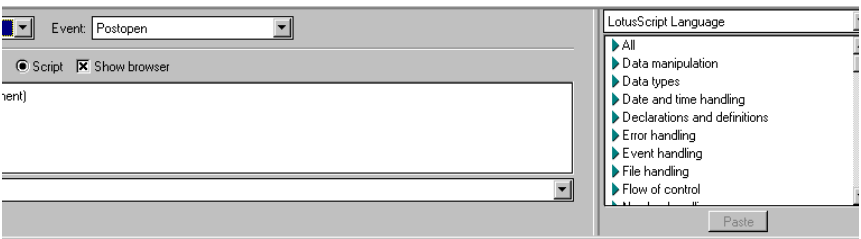
- Formula: Lets you write Notes formula language macros and commands that will run when the specified event occurs for the object. When you specify Formula, a Fields & Functions button (see following figure) appears at the bottom of the design pane. Clicking this button brings up a dialog box which will display all of the fields defined on the form, or all of the functions available in the Notes formula language. Double-clicking one of these fields or functions inserts it into the editor window at the current cursor position.



When certain events are selected an @Commands button will appear next to the Fields & Functions button. Clicking this button brings up a dialog box which will display all of the @Commands available in the Notes formula language. Not all objects on the form support formulas. If an object which does not support formulas is selected in the define box, this radio button is disabled.

- Script: Lets you write LotusScript procedures that will run when the specified event occurs for the object. When this option is selected, the Fields & Functions button and an Error combo box are made available. Not all objects on the form support LotusScript. If an object which does not support LotusScript is selected in the define box, this radio button is disabled.

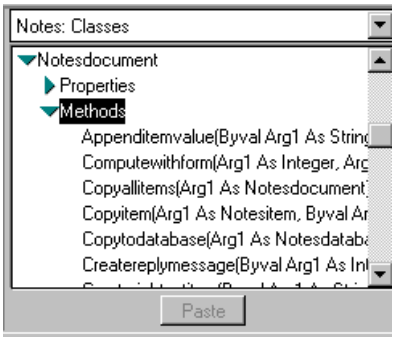
The last control in the run area is the Show Browser check box. This control is only enabled when the Script radio button is selected. When this control is checked, the LotusScript browser is displayed, as shown below:



The browser is a ready reference of the LotusScript language and functions, and of all of the objects, their properties and methods. For example, if you want to quickly determine all of the methods available for the NotesDocument class, show the browser and then:

1. Select Notes Classes from the top combo box. All the Notes object classes are displayed.
2. Scroll the browser's list box until you find the NotesDocument entry.
3. Click the triangular "twistie" icon to expand the listing under NotesDocument. You will see three entries, for Properties, Methods and for Events. The Events entry has no twistie, meaning there are no events defined for this class.

4. Click the Methods twistie, and a list of all of the NotesDocument methods will be listed, as shown below:

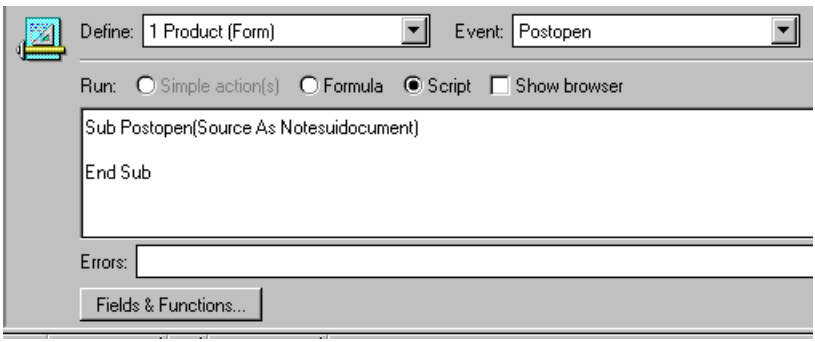


5. Double-click the method or property in the browser to insert the prototype code into your Notes application.

Script Editor and Formula Editor

The larger window below the run area is where you program the actions that Notes will execute. If the Formula radio button is selected, this area is the formula editor. If the Script radio button is selected, this area is the script editor.

When the Script button is selected, the script editor will automatically enter the appropriate Sub and End Sub statements for the specified object and event, as the following picture illustrates:



Error Box

This box (see figure above) is only displayed when the Script button is checked. It lists all of the syntax errors that Notes detected in your LotusScript. If there are multiple errors, clicking the arrow to the right of the error box will expand this combo box so that all syntax errors found are displayed. You can navigate to a specific error in the list, no matter where in the application it is located, by selecting it from the drop-down list.

When a syntax error is corrected, Notes will remove the error indication from the error box. Notes will not allow you to save a form with LotusScript syntax errors, so you will have to fix all errors (which will result in an empty error box) before you will be able to save your form. If you wish to save an application with errors in it, comment out the sections that contain errors, or copy the contents to the clipboard, remove the error, and save the application.

Working with the Script Editor

The script editor functions very much like a text editor. The standard text editor key conventions are used, such as:

- **HOME** places the cursor at the start of the current line.
- **END** places the cursor at the end of the current line.
- **CTRL+HOME** places the cursor at the start of the script.
- **CTRL+RIGHT ARROW** moves the cursor one word to the right.

You can select text in the usual way (using the **SHIFT** and arrow keys, or by dragging the mouse pointer over the text to be selected). The clipboard-related menus and SmartIcons, such as Cut and Paste, are available when you are working in the script editor, as well as the corresponding accelerator keys, such at **CTRL+C** and **CTRL+V**. This means that you can cut and paste scripts, or script fragments, from other objects in Notes.

Special Script Editor Features

One feature already mentioned was the capability of looking up LotusScript functions and objects in the browser. If you click on any entry in the browser, that entry becomes highlighted, and you can click the Paste button at the bottom of the browser to paste that line at the current cursor position in the script editor. You can also double-click the entry to copy it to the script editor.

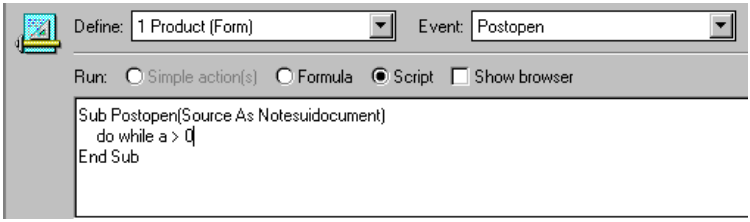
Note In Notes Release 4.5, additional features will be available, such as colorization of identifiers, the ability to export or import all the code, and selection of the font in which to display the script.

Each time you press the **ENTER** key, or move off a line of LotusScript, the script editor checks that line for syntax errors, and also capitalizes the LotusScript reserved words, i.e., the words in the statement that are a component of the LotusScript basic language.

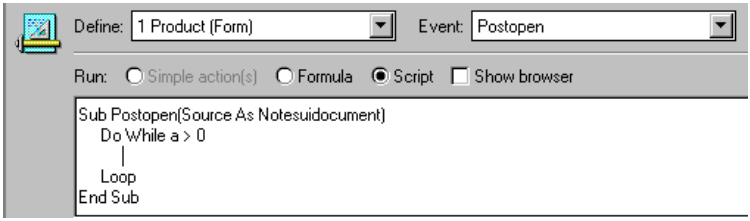
When you enter structured programming statements such as For, While, Do, Select Case, etc., the script editor automatically does the following:

- Inserts the corresponding ending statement (for example, Loop for the Do statement) below the statement you typed.
- Inserts a blank line between the two statements, with the cursor being placed on that line so you can continue to type your code.
- Automatically indents the statements within the construct.

The following figure shows the state of the script editor after you type the opening statement of a Do loop:



When the **ENTER** key is pressed, the script editor window changes as shown in the following figure:

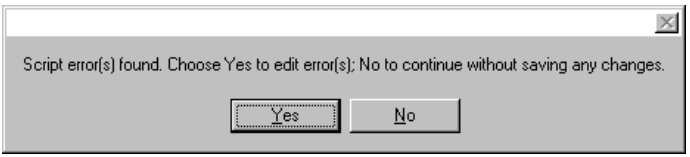


LotusScript is not case sensitive, except for text constants. Thus the constant "Text" is not the same as the constant "text." It does not matter, then, if you sometimes refer to a variable using upper case, and at other times use lower case; LotusScript will consider both to refer to the same variable. However, it is advisable that you develop a consistent naming convention and then stick to it. For example, you may have all constants start with an upper case letter, and all variables with lower case letters. LotusScript allows names (for constants, variables, etc.) to have up to 40 characters, so do not skimp on letters when naming your variables. Although a variable name such as employee_pay_rate, or employeePayRate, is harder to type than p, it is a lot more meaningful when you or someone else is trying to modify the code at a later date. By the way, long names do not slow down execution of LotusScript because all names are transformed by the compilation process into relative memory addresses.

Error Checking

There are two types of error checking performed in the script editor:

1. Each time you move the cursor to a different line, by pressing **ENTER**, using a cursor movement key, or clicking with the mouse, Notes will check the syntax of that line in isolation. This kind of syntax check picks up errors such as unmatched parentheses and incomplete expressions. Notes does not force you to fix these errors right away. You can continue entering code. However, Notes will not allow you to save the form until the error is corrected.
2. Whenever you save the form, Notes will compile the LotusScript code, and may uncover other errors, such as a Select Case statement with no matching End Select statement, and invocations of methods which the object does not support. When such a compile error is found, Notes will display the following message box:



   If you click Yes, the form is not saved and you can go and edit the code to correct the error. If you click No, the form is not saved *and any changes you made since the last successful save are lost.*

Testing the Form

Choosing Design → Test Form will cause the form to be saved, and a new document created based on the form. You can then run this new document through its paces to check on how the code you wrote for the form is performing. When you complete your testing, press **ESC** (Notes will ask you if you want to save the document) to take you back to the form design IDE.

Debugging LotusScript

It is possible that all of your LotusScript statements are syntactically correct, and that your program compiles without error, and yet it will not run correctly. Consider this very simple code fragment:

```
If A + B Then C = 0
```

What the programmer meant to type is

```
If A = B Then C = 0
```

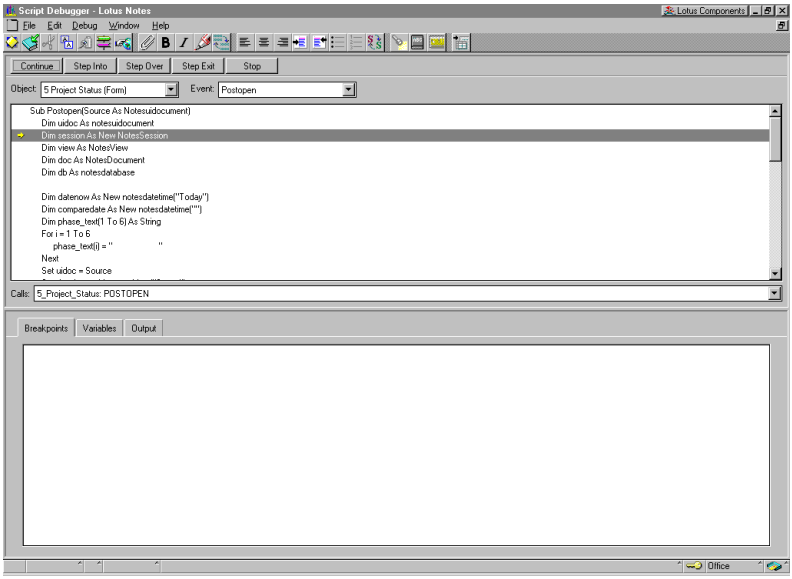
but in haste, the **SHIFT** key was not released in typing the equals sign, and the plus sign was typed instead. Both statements are syntactically correct,

but the results are different. Such errors are called logic errors, meaning that the logic of the program is flawed.

For such errors Notes provides LotusScript debug facilities. To invoke the debugger, choose File → Tools → Debug LotusScript. This menu choice is a toggle, so to turn debugging off, choose the same command.

Note You need to turn the debugger on before loading or creating the document on which you want the debugger to work on.

When LotusScript starts executing, the debugger window, shown in the following figure, will appear:



Notice the following elements of this window:

- A debug menu on the menu bar.
- Action buttons, which control the execution of the next statement of LotusScript (for details on the operation of each of these buttons, consult the Notes documentation).
- Object and Event combo boxes, which correspond to the Define and Event combo boxes in the form design window.
- A script window that shows the current script sub being debugged.
- Calls window, which shows the sequence of calls that led up to this point in the code.

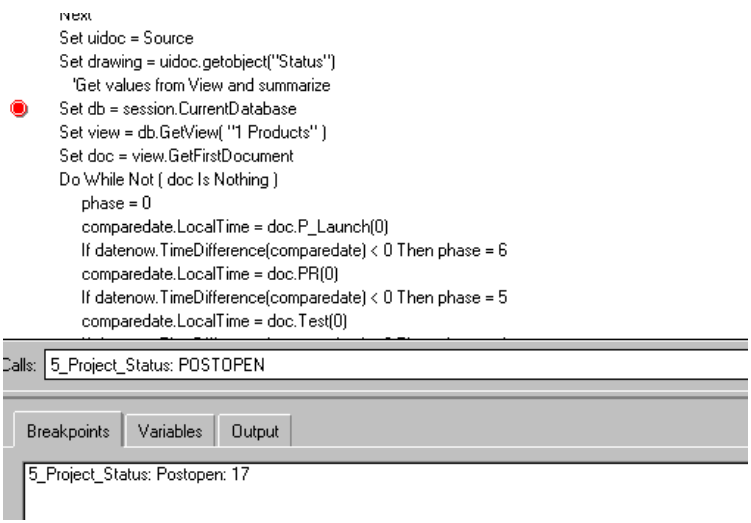
- Information window, where you can see any set breakpoints, the current values of variables in your program, and any output produced by Print statements in your program. The size of the information window and the script window can be changed by dragging their common border.

The statement about to be executed is preceded by a right arrow, and highlighted.

Setting Breakpoints

Setting breakpoints is useful if you want your program to stop at a specific statement, and stepping to the statement is time consuming, or if the program only reaches the statement on occasion. In the script window select the statement where you want to set the breakpoint, and double click it, press **F9**, or choose Debug → Set / Clear Breakpoint. A red stop sign is placed to the left of the statement, and an entry is made in the breakpoint window in the form object:event:line. For example, the following figure shows we have set a breakpoint at the statement

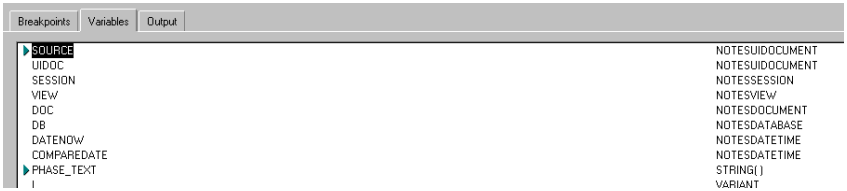
```
set db = session.CurrentDatabase
```



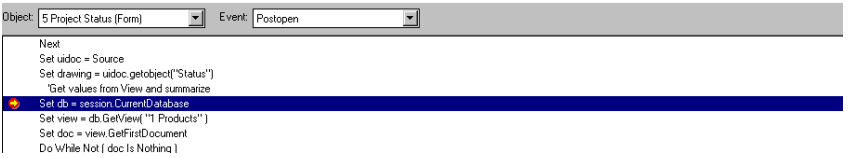
The Breakpoints tab of the information window shows us that:

- The object whose script we are examining is the 5_Project_Status object, in this case a form.
- The event is the Postopen event, which occurs when a document is opened.
- The breakpoint is set at line 17 of this sub.

Before we let the code run to the breakpoint, let's look at the Variables tab. This is illustrated in the following figure:

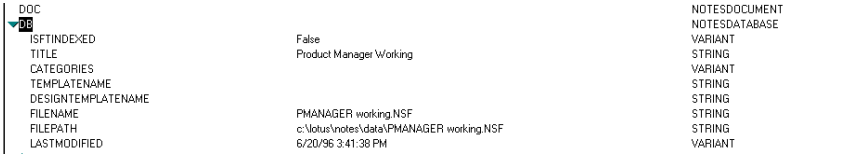


Note, for example, that variable DB (since LotusScript is case insensitive, db and DB are the same variable) is listed, and it is of type NotesDatabase. However, at this time there is no value assigned to this variable. If we let the script run by clicking the Continue button, we see the following:



The program has stopped just prior to executing the Set db statement. We wish to find out what happens when this statement is executed. We click the Step Into button, and the statement is executed, and Notes stops prior to executing the next statement. If we now look at the Variables tab in the information window we see the following:

Note that scalar variables, such as I, display their value. More complex variables, such as DB (which, as a result of the statement just executed, points to a NotesDocument object) have a twistie to their left. Clicking on the twistie displays the components of the variable. Since a NotesDocument is composed of fields, clicking on its twistie displays the fields in the document and their current values, as shown in the following figure:



Thus the debugger is a powerful tool in helping you understand what your program is doing.

Chapter 3
LotusScript Notes Classes

Notes defines LotusScript object classes (called Notes classes in the following) that allow you to access Notes structures at two levels:

The database (back-end) classes allow you to access named databases, views, documents, and other Notes objects. Both workstation and server users can run scripts that access database objects.

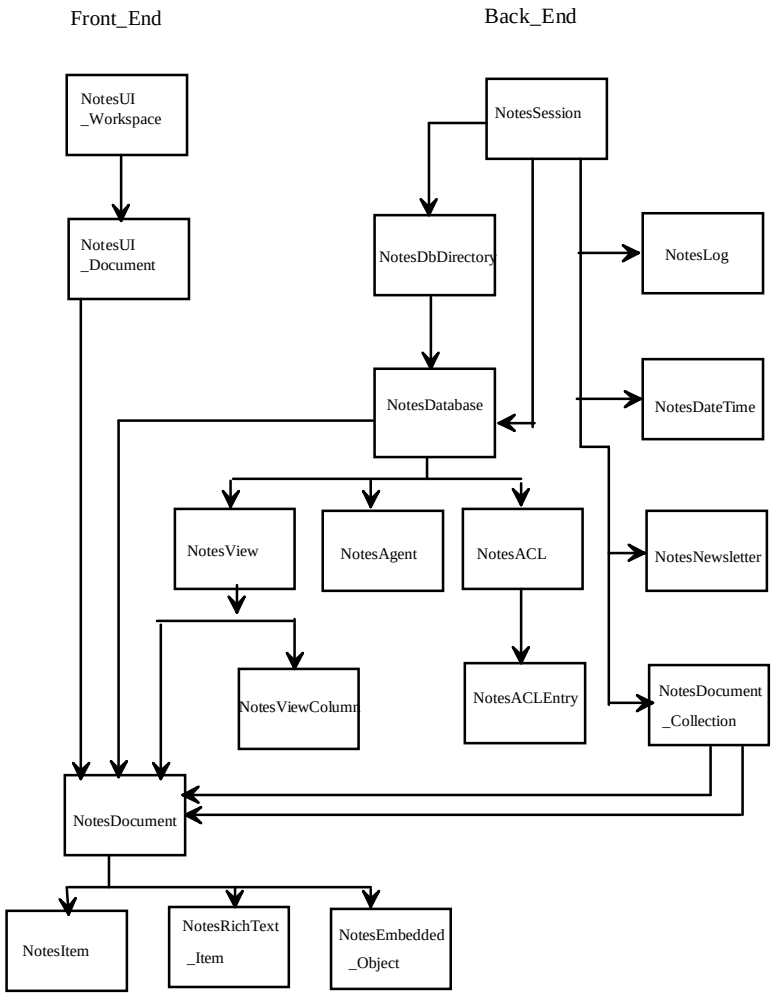
The UI (front-end) classes allow you to access current objects that the user is working on. Only workstation users can run scripts that access UI objects.

There is a hierarchical relationship for object classes. Higher hierarchical classes contain the lower ones. Each class has defined members, properties and methods. Using these members, you can access other objects. The relationship of containment and access means that the higher class has the property or the method to access the lower one. NotesSession is the highest-level class in the hierarchy among the database classes. The NotesUIWorkspace class is the highest-level class in the hierarchy among the UI classes.

In this chapter, we will discuss:

- Database classes and UI classes
- How to access the classes
- The properties and methods of the classes

The following figure shows the hierarchy of Notes classes:



Note This chart includes the 18 classes available with Lotus Notes Release 4.1. There will be an additional six classes available with Notes Release 4.5. These classes are documented in the following, and will be identified as Release 4.5 classes in the text.

The Database (back_end) Classes

There are 16 database (back-end) classes available in Release 4.1. An additional five database classes will be available with Release 4.5. The classes are as follows:

Class	*Description*
NotesACL	Represents a collection of all the access control list entries for a database.
NotesACLEntry	Represents a single entry in an access control list.
NotesAgent	Represents an agent.
NotesDatabase	Represents a Notes database.
NotesDateTime	Provides a means to translate between LotusScript and Notes date/time formatting.
NotesDbDirectory	Represents the database files on a server or the local machine.
NotesDocument	Represents a document in a database.
NotesDocumentCollection	Represents a collection of documents.
NotesEmbeddedObject	Represents embedded objects, links, and file attachments.
NotesItem	Represents a piece of data in a document.
NotesLog	Represents actions and errors that occur during a script's execution.
NotesNewsLetter	A summary document that contains information from, or links to, several other documents.
NotesRichTextItem	Represents items that can contain rich text.
NotesSession	Root of Notes database objects—for global attributes, context, and persistent information.
NotesView	Represents a named view of a database.
NotesViewColumn	Represents a column of a view.
NotesDateRange	Represents a range of Notes date.
NotesForm	Represents a Notes form.
NotesInternational	Represents date-time settings from operating system.
NotesName	Represents a name.
NotesTimer	Represents a timer.

Note The classes listed in ***italics*** will be available in Lotus Notes Release 4.5.

NotesACL

Every NotesDatabase contains a NotesACL object representing that database's access control list. To get NotesACL, use the ACL property in NotesDatabase. NotesACL also contains NotesACLEntry.

The NotesDatabase class has three methods: QueryAccess, GrantAccess and RevokeAccess. You can use these methods to access and modify an ACL without declaring a NotesACL object. All you need to know is the name of the person, server, or group.

Method	*Description*
AddRole	Adds a role to an ACL.
CreateACLEntry	Creates an entry in the ACL with the name and level that you specify. When used with OLE automation, this method allows you to create a NotesACLEntry object without using the New method.
DeleteRole	Deletes a role from an ACL.
GetEntry	Given a name, finds its entry in an ACL.
GetFirstEntry	Returns the first entry in an ACL, usually the -Default- entry.
GetNextEntry	Given an ACL entry, returns the next one.
RenameRole	Changes the name of a role.
Save	Saves the changes you've made to the ACL. If you don't call Save before closing a database, the changes you've made to the ACL are lost.

Property	*Description*
Parent	Read-only. The database that owns an ACL.
Role	Read-only. All the roles defined in an access control list.
UniformAccess	Read-Write. Enables the "uniform access" option for the database.

Note The property listed in ***italics*** will be available in Lotus Notes Release 4.5.

NotesACLEntry

To create a new NotesACLEntry object, we use the New method of NotesACLEntry or the CreateACLEntry method of NotesACL. The New method creates an entry in an ACL with the name and level that you

specify. You must call Save on the ACL if you want the modified ACL to be saved to disk.

NotesACL provides three ways to access an existing NotesACLEntry:

- To access an entry in an ACL when you know its name, use GetEntry.
- To access the first entry in the ACL, use GetFirstEntry.
- To access entries after the first one, use GetNextEntry.

Method	*Description*
DisableRole	Given a role, disables the role for an entry.
EnableRole	Given the name of a role, enables the role for an entry.
IsRoleEnabled	Given a role, indicates if the role is enabled for an entry.
New	Creates a new ACLEntry.
Remove	Removes an entry from an access control list.

Property	*Description*
CanCreateDocuments	Read-Write. For an entry with Author access to a database, indicates if the entry is allowed to create new documents.
CanCreatePersonalAgent	Read-write. Indicates if an entry can create personal agents in a database.
CanCreatePersonalFolder	Read-write. Indicates if an entry can create personal folders in a database.
CanDeleteDocuments	Read-Write. For an entry with Author access or above to a database, indicates if an entry can delete documents.
Level	Read-Write. The access level this entry has for this database.
Name	Read-Write. The name of an entry.
Parent	Read-only. The access control list that contains an entry.
Roles	Read-only. The roles that are enabled for an entry.

NotesAgent

NotesAgent is contained by NotesSession and NotesDatabase. To access the agent that's currently running, use the CurrentAgent property of NotesSession. To access all the agents in a database, use the Agents property of NotesDatabase.

Method	*Description*
Remove	Permanently deletes an agent from a database.
Run	Runs an agent from a database.

Property	*Description*
Comment	Read-only. The comment that describes an agent, as entered by the agent's designer.
CommonOwner	Read-only. The name of the person who last modified and saved an agent (the agent's owner). If the owner has a hierarchical name, only the common name is returned.
IsEnabled	Read-only. Indicates if an agent is able to run or not.
IsPublic	Read-only. Indicates if an agent is public or personal.
LastRun	Read-only. The date that an agent last ran.
Name	Read-only. The name of an agent. Within a database, the name of an agent may not be unique.
Owner	Read-only. The name of the person who last modified and saved an agent (the agent's owner). If the owner has a hierarchical name, the entire name is returned.
Parent	Read-only. The database that contains an agent.
Query	Read-only. The text of the query used by an agent to select documents.
ServerName	Read-only. The name of the server on which an agent runs.

Note The method listed in ***italics*** will be available in Lotus Notes Release 4.5.

NotesDatabase

NotesDatabase is contained by NotesSession and NotesDbDirectory. It contains NotesACL, NotesAgent, NotesDocument, NotesDocumentCollection, and NotesView.

Note In Notes Release 4.5, NotesDatabase will contain NotesForm and will be contained by NotesUIDatabase.

We have many ways to access NotesDatabase:

- To access an existing database when you know its server and file name, use the New method of NotesDatabase, or the GetDatabase method of NotesSession.
- To access the database in which a script is currently running, without indicating a server or file name, use the CurrentDatabase property of NotesSession.
- To access an existing database when you know its server and replica ID, use the OpenByReplicaID method of NotesDatabase.
- To access an existing database when you know its server but not its file name, use the NotesDbDirectory class.
- To access the current user's mail database use the OpenMail method of NotesDatabase.
- To open the default Web Navigator database, use the OpenURLDb method of NotesDatabase.
- To access the available Address books, use the AddressBooks property of NotesSession.
- To test for the existence of a database with a specific server and file name before accessing it, use one of these properties or methods of NotesDatabase: IsOpen, Open, OpenIfModified.
- To create a new database from an existing database, use one of these methods of NotesDatabase: CreateCopy, CreateFromTemplate, CreateReplica.
- To create a new database from scratch, use the Create method of NotesDatabase.
- To access a database when you have a NotesView, NotesDocument, NotesDocumentCollection, NotesACL, or NotesAgent from that database, use the appropriate Parent (or ParentDatabase) property.

Method	*Description*
New	Creates a new NotesDatabase object.
Close	Closes a database.
Compact	Compacts a database.
Create	Creates a new database on disk, using the server and file name that you specify. Because the new database is not based on a template, it is blank and does not contain any forms or views.
CreateCopy	Creates an empty copy of the current database. The copy contains the design elements of the current database, an identical access control list, and an identical title. The copy does not contain any documents and is not a replica.
CreateDocument	Creates a document in a database and returns a NotesDocument object that represents the new document. You must call Save if you want the new document to be saved to disk. When used with OLE automation, this method allows you to create a NotesDocument object without using the New method.
CreateFromTemplate	If the current database is a template, creates a new database from the template. The new database has the design features of the template and does not contain any documents.
CreateReplica	Creates a replica of the current database at a new location.
FTSearch	Conducts a full text search of all the documents in a database.
GetAgent	Gets an agent by agent name in the database.
GetDocumentByID	Finds a document in a database, given the document's NoteID.
GetDocumentByUNID	Finds a document in a database, given the document's universal ID (UNID).
GetDocumentByURL	Instantiates a document in the Web Navigator database and returns a NotesDocument object for it. The database must be an open Web Navigator database.
GetForm	Gets a form by form name in the database.
GetProfileDocument	Gets a profile document of the database.

Continued

Method	*Description*
GetURLHeaderInfo	Gets the specific Hypertext Transfer Protocol (HTTP) header information from the uniform resource locator (URL). A URL is a text string used for identifying and addressing a Web page.
GetView	Finds a view or folder in a database, given the name or alias of the view or folder.
GrantAccess	Modifies a database access control list to provide the specified level of access to a person, group, or server.
Open	Opens a database. A database must be open in order for a script to access its properties and methods.
OpenByReplicaID	Given a server name and a replica ID, opens the specified database, if it exists.
OpenIfModified	Given a date, opens the specified database if it has been modified since that date.
OpenMail	Assigns a database to the current user's mail database and opens the database.
OpenURLDb	Binds the current database object to the default Web Navigator database. If the object is currently bound to another database, that database is closed first.
OpenWithFailover	Finds a replica database on another server if the first choice is not available.
QueryAccess	Returns a person's, group's, or server's current access level to a database.
Remove	Permanently deletes a database from disk.
Replicate	Replicates a database with its replica(s) on a given server.
RevokeAccess	Removes a person, group, or server from a database access control list. This resets the access level for that person, group, or server to the Default setting for the database.
Search	Given selection criteria for a document, returns all documents in a database that meet the criteria.
UnprocessedFTSearch	Given selection criteria for a document, returns documents in a database that the current agent considers to be unprocessed or that match the query.

Continued

Method	*Description*
UnprocessedSearch	Given selection criteria for a document, returns documents in a database that the current agent considers to be unprocessed, meet the criteria, or were created or modified since the cutoff date.
UpdateFTindex	Updates the database's full text index.

Property	*Description*
ACL	Access control list for the database
Agents	Agents in the database
AllDocuments	All the documents in the database
Categories	(Read-write) Categories in the database
Created	Date and time the database was created.
CurrentAccessLevel	User's access level to the database
DelayUpdates	Read-Write. When doing multiple updates to documents on a server, don't do a synchronous write for each update; let the server batch them later.
DesignTemplateName	Database's design template, if any
FileName	Database file name
FilePath	Database path
Forms	Notes form array
IsMultiDbSearch	Read-Write. True if the database contains a multi-database full-text search index.
IsFTIndexed	True if the database is full text indexed.
IsOpen	True if the database is open.
IsPrivateAddressBook	True if the database is a Personal Address Book; only valid through AddressBooks in NotesSession.
IsPublicAddressBook	True if the database is a Public Address Book; only valid through AddressBooks in NotesSession.
LastFTIndexed	Date and time a full text index, if any, was last updated.
LastModified	Date and time the database was last modified.

Continued

Property	*Description*
Managers	Users that have Manager access to the database.
Parent	Current Notes session
PercentUsed	Percent of a database's total size that is occupied by real data.
ReplicaID	Database replica ID in hexadecimal
Server	Name of the server containing the database.
Size	Database size, in bytes
SizeQuota	(Read-write) Database size quota, if any; you must be an administrator to write.
TemplateName	Database template name, if database is a template.
Title	(Read-write) Database title
UnprocessedDocuments	All documents not yet processed in agent or view.
Views	Named views in the database

Note The properties and methods listed in ***italics*** will be available in Lotus Notes Release 4.5. The FTSearch method will change in Release 4.5; two new optional arguments, sort and other, will be available. The ***Close*** method will no longer be available in Release 4.5.

NotesDateRange

The NotesDateRange class will be available in Notes Release 4.5. It contains NotesDateTime.

Property	*Description*
EndDateTime	Read-Write. The end time of NotesDateRange.
StartDateTime	Read-Write. The start time of NotesDateRange.
Text	Read-Write. Text format of the date range.

NotesDateTime

NotesDateTime is contained by NotesSession and NotesDateRange. To create a new NotesDateTime object, you can use the New method of NotesDateTime, or the CreateDateTime method of NotesSession. Given a string that represents the date and time you want, New creates an object that represents that date and time.

Method	*Description*
New	Creates a new DateTime Object.
AdjustDay	Increments a date-time by the number of days you specify.
AdjustHour	Increments a date-time by the number of hours you specify.
AdjustMinute	Increments a date-time by the number of minutes you specify.
AdjustMonth	Increments a date-time by the number of months you specify.
AdjustSecond	Increments a date-time by the number of seconds you specify.
AdjustYear	Increments a date-time by the number of years you specify.
ConvertToZone	Converts the time / date value to the specified time zone.
SetAnyDate	Sets the date component to a wildcard value, which means it will match any date. The time component is unaffected.
SetAnyTime	Sets the time component to a wildcard value, which means it will match any time. The date component is unaffected.
SetNow	Sets the value of a date-time to now (today's date and current time).
TimeDifference	Finds the difference in seconds between one date-time and another.

Property	*Description*
GMTTime	Read-only. A string representing a date-time, converted to Greenwich Mean Time (timezone 0).
IsDST	Read-only. Indicates if the time reflects daylight savings time.
LocalTime	Read-Write. A string representing a date-time, in the local time zone.
LSGMTTime	Read-only. A LotusScript variant representing a date-time, converted to Greenwich Mean Time (timezone 0).
LSLocalTime	Read-Write. A LotusScript variant representing a date-time, in the local time zone.
TimeZone	Read-only. An integer representing the time zone of a date-time. In many cases, but not all, this integer indicates the number of hours which must be added to the time to get Greenwich Mean Time. May be positive or negative.
ZoneTime	Read-Only. Displays the time / date in the zone in which it was originally stored.

Note The properties and methods listed in ***italics*** will be available in Lotus Notes Release 4.5.

NotesDbDirectory

The NotesDbDirectory class is contained by NotesSession and contains NotesDatabase.

You create a new NotesDbDirectory object using the name of the server you want to access. You can use the New method of NotesDbDirectory or the GetDbDirectory method of NotesSession.

Method	*Description*
New	Creates a new NotesDbDirectory object.
GetFirstDatabase	Returns the first database on a server (or local computer), using the file type you specify.
GetNextDatabase	Returns the next database in a directory, using the file type specified in the GetFirstDatabase method.

Property	*Description*
Name	Read-only. The name of the server whose database directory you are searching. This property is set when you create a database directory using New.

Note For GetFirstDatabase, you must use the type number in Visual Basic using Notes Classes through OLE Automation, rather than the type constant used in LotusScript.

Type Constant	*Type Number*
DATABASE	1247
TEMPLATE	1248
REPLICA _CANDIDATE	1245
TEMPLATE_CANDIDATE	1246

NotesDocument

NotesDocument is contained by NotesDatabase, NotesDocumentCollection, and NotesView. It contains NotesEmbeddedObject, NotesItem, and NotesRichTextItem.

To create a new NotesDocument object, we use the New method of NotesDocument or the CreateDocument method of NotesDatabase.

You must call Save if you want the new document to be saved to disk.

Method	*Description*
New	Creates a new NotesDocument object.
AppendItemValue	Creates a new item on a document and sets the item's value.
ComputeWithForm	Validates a document by executing the default value, translation, and validation formulas, if any are defined in the document's form.
CopyAllItems	Copies all items in a document.
CopyItem	Given a destination document, copies all of the items in the current document into the destination document. The item names are unchanged.
CopyToDatabase	Copies a document into the specified database.
CreateReplyMessage	Creates a new document that is formatted as a reply to the current document.

Continued

Method	*Description*
CreateRichTextItem	Creates a new rich text item in a document, using a name you specify, and returns the corresponding NotesRichTextItem object. When used with OLE automation, this method allows you to create a new rich text item and NotesRichTextItem object without using the New method.
Encrypt	Encrypts a document in a database.
GetAttachment	Given the name of a file attachment, returns a NotesEmbeddedObject representing the attachment.
GetFirstItem	Given a name, returns the first item of the specified name belonging to the document.
GetItemValue	Given the name of an item, returns the value of that item in a document.
GetNextItem	Given an item, returns the next item of the same name belonging to a document.
HasItem	Given the name of an item, indicates if that item exists in the document.
MakeResponses	Makes one document a response to another document. The two documents must be in the same database.
PutinFolder	Adds a document to the specified folder. If the folder does not exist in the document's database, it is created.
Remove	Permanently deletes a document from a database.
RemoveFromFolder	Removes a document from the specified folder.
RemoveItem	Given the name of an item, deletes the item from a document.
RenderToRTItem	Creates a picture of a document and places it into a rich text item you specify. The picture is created using both the document and its form; therefore, the form's input translation and validation formulas are executed.
ReplaceItemValue	Replaces all items of the specified name with one new item, which is assigned the specified value. If the document does not contain an item with the specified name, the method creates a new item and adds it to the document.

Continued

Method	*Description*
Save	Saves any changes you have made to a document.
Send	Mails a document to the recipients you specify.
Sign	Signs a document.

Property	*Description*
Authors	Read-only. The names of the people who have saved a document.
ColumnValues	Read-only. An array of values, each element of which corresponds to a column value in the document's parent view.
Created	Read-only. The date a document was created.
EmbeddedObjects	Read-only. The OLE/2 and OLE/1 embedded objects in a document.
EncryptionKeys	Read-Write. The key(s) used to encrypt a document. The Encrypt method uses these keys when it encrypts the document.
EncryptOnSend	Read-Write. Indicates if a document is encrypted when mailed.
FTSearchScore	Read-only. The full text search score of a document, if it was retrieved as part of a full text search.
HasEmbedded	Read-only. Indicates if a document contains one or more embedded objects, object links, or file attachments.
IsNewNote	Read-only. Indicates if a document is new. A document is new if it hasn't been saved.
IsProfile	Read-only. Indicates if a document is a profile document.
IsResponse	Read-only. Indicates if a document is a response to another document.
IsSigned	Read-only. Indicates if a document contains a signature.
IsUIDocOpen	Read-only. True if this document was accessed from a NotesUIDocument.
IsUnread	Read-only. True if this document was unread.

Continued

Property	*Description*
Items	Read-only. All the items in a document. An item is any piece of data stored in a document.
Key	Read-only. Profile key, if this is a profile document.
LastAccessed	Read-only. The date a document was last modified or read.
LastModified	Read-only. The date a document was last modified.
NameOfProfile	Read-only. The name of the profile document, if it is a profile document.
NoteID	Read-only. The NoteID of a document, which is an 8-character combination of letters and numbers that uniquely identifies a document within a particular database.
ParentDatabase	Read-only. The database that contains a document.
ParentDocumentUNID	Read-only. The universal ID of a document's parent, if the document is a response. Returns an empty string ("") if a document doesn't have a parent.
ParentView	Read-only. The view from which a document was retrieved, if any. If the document was retrieved directly from the database or a document collection, returns Nothing.
Responses	Read-only. The immediate responses to a document.
SaveMessageOnSend	Read-Write. Indicates if a document is saved to a database when mailed. Only applies to new documents that have not yet been saved.
SentByAgent	Read-only. Indicates if a document was mailed by a script.
Signer	Read-only. The name of the person who created the signature, if a document is signed.
SignOnSend	Read-Write. Indicates if a document is signed when mailed.
Size	Read-only. The size of a document in bytes, which includes the size of any file attachments in the document.

Continued

Property	*Description*
UniversalID	Read-only. The Universal ID of a document, which is a 32-character combination of letters and numbers that uniquely identifies a document across all replicas of a database.
Verifier	Read-only. The name of the certificate that verified a signature, if a document is signed.

Note The methods and properties listed in ***italics*** will be available in Lotus Notes Release 4.5. The UniversalID property will change to Read-Write in Release 4.5. The CopyAllItems method will have a new optional argument, replace, in Rel. 4.5. The ***GetNextItem*** method will be removed in Release 4.5.

NotesDocumentCollection

NotesDocumentCollection contains NotesDocument.

A NotesDocumentCollection represents a subset of all the documents in a database. The documents in the subset are determined by the NotesDatabase method or property we use to search the database, which can be any of the following:

- AllDocuments property
- UnprocessedDocuments property
- Search method
- UnprocessedSearch method
- FTSearch method
- UnprocessedFTSearch method

The Responses property in NotesDocument also returns a NotesDocumentCollection.

Both NotesDocumentCollection and NotesView provide access to documents within a database. Use the NotesDocumentCollection object if:

- You want to act on a specific set of documents that meet certain criteria.
- There is no view in the database that contains every document you need to search.
- You do not need to navigate the documents' response hierarchies.

Views are a more efficient means of accessing documents because they are already indexed by the database itself. However, they do not necessarily provide access to the documents that you want.

The documents in a collection are ordered when the collection results from a full text search; otherwise, the documents are unordered.

Method	*Description*
FTSearch	Fetches the documents in a query.
GetFirstDocument	Gets the first document in a collection.
GetLastDocument	Gets the last document in a collection.
GetNextDocument	Given a document, finds the document immediately following it in a collection.
GetNthDocument	Given a position number, returns the document at that position in a collection.
GetPrevDocument	Given a document, finds the document immediately preceding it in a collection.
PutAllInFolder	Puts all documents in a folder.
RemoveAll	Removes all the documents in NotesDocumentCollection.
RemoveAllFromFolder	Removes all the documents from a folder.
SaveAll	Saves all the documents in NotesDocumentCollection.
StampAll	Stamps all the documents.
UpdateAll	Updates all the documents.

Property	*Description*
Count	Read-only. The number of documents in a collection.
IsSorted	Read-only. Indicates if the documents in a collection are sorted. A collection is sorted only when it results from a full text search of a database.
Parent	Read-only. The database that contains a document collection.
Query	Read-only. The text of the query that produced a document collection, if the collection results from a full text or other search.

Note The methods listed in ***italics*** will be available in Release 4.5.

NotesEmbeddedObject

The NotesEmbeddedObject class is contained by NotesDocument and NotesRichTextItem. It represents an embedded object, an object link, or a file attachment. Some methods and properties that are available for embedded and linked objects are unavailable for file attachments.

We use the EmbedObject method of NotesRichText to create a NotesEmbeddedObject. There are three ways to access the class:

- To access an object, object link, or attachment when you know its name and the rich text item that contains it, use the GetEmbeddedObject method of NotesRichTextItem.
- To access all the objects, object links, and attachments in a particular rich text item, use the EmbeddedObjects property of NotesRichTextItem.
- To access the objects and object links in a particular document, including those that are not contained within a particular rich text item, use the EmbeddedObjects property of NotesDocument. This property does not return file attachments or objects and object links created in Notes Release 3.

Method	*Description*
Activate	Causes an embedded object or object link to be loaded by OLE.
DoVerb	Given the name of a verb, executes the verb in an embedded object.
ExtractFile	Copies a file attachment to disk.
Remove	Permanently deletes an embedded object, object link, or file attachment.

Property	*Description*
Class	Read-only. The name of the application which created an object.
FileSize	Read-only. The size of an embedded object, object link, or file attachment, in bytes.
Name	Read-only. The name used to reference an embedded object or object link.

Continued

Property	*Description*
Object	Read-only. If an embedded object has been loaded into memory, returns the OLE handle (IUnknown or IDispatch handle). If the OLE object supports OLE Automation, you can invoke the methods and properties of the object using the handle.
Parent	Read-only. The rich text item that holds an object.
Source	Read-only. If the NotesEmbeddedObject is an embedded object or object link, this property returns the internal name that Notes uses to refer to the source document. If the NotesEmbeddedObject is a file attachment, this property returns the file name of the original file.
Type	Read-only. Indicates if a NotesEmbeddedObject is an embedded object, an object link, or a file attachment.
Verbs	Read-only. The verbs that an object supports, if the object is an OLE/2 embedded object.

Note The DoVerb method of NotesEmbeddedObject is used differently depending on the platform and OLE server. For example, you use Embobj.DoVerb "&Open" in Windows 95 and Embobj.DoVerb "Edit" in Windows 3.1. For details on the OLE server, refer to the appropriate OLE server documentation.

NotesForm

The NotesForm class will be available in Release 4.5.

Method	*Description*
Remove	Deletes a form from the database.

Properties	*Description*
Aliases	Read-only. String array of aliases.
Fields	Read-only. String array of field names.
FormUsers	Read-Write. String array of the form user names.
IsSubForm	Read-only. Indicates if the form is a subform.
Name	Read-only. Form name.
ProtectReaders	Read-Write. If true, does not replace $Readers item during replication.
ProtectUsers	Read-Write. If true, does not replace $FormUser item during replication.
Readers	Read-Write. String array of readers.

NotesInternational

The NotesInternational class will be available in Release 4.5. All the properties here are Read-only.

Property	*Description*
AMString	Read-only. String that indicates before noon in 12-hour time.
CurrencyDigits	Integer - number of digits after decimal point.
CurrencySymbol	String - the currency symbol, for example, the dollar sign in US.
DateSep	String - the character used to separate the parts of the date.
DecimalSep	String - the character used to separate the parts of a decimal number.
IsCurrencySpace	True if the currency symbol is separated by a space from the number.
IsCurrencySuffix	True if the currency symbol follows the number.
IsCurrencyZero	True if a decimal number that is a fraction has a zero before the decimal separator.
IsDateDMY	True if the date format is day-month-year.
IsDateMDY	True if the date format is month-day-year.
IsDateYMD	True if the date format is year-month-day.
IsDST	True if daylight savings time is in effect.
IsTime24Hour	True if the time is in 24-hour format.
PMString	String that indicates after noon in 12-hour time.
ThousandsSep	String - the character that separates numbers at the 1000s.
TimeStep	String - the character used to separate the parts of the time.
TimeZone	Integer - the time zone.
Today	String that means today.
Tomorrow	String that means tomorrow.
Yesterday	String that means yesterday.

NotesItem

In the user interface, Notes displays items in a document through fields on a form. When a field on a form and an item in a document have the same name, the field displays the item (for example, the Subject field displays the Subject item). 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.

The NotesItem class is contained by NotesDocument. You can create a new NotesItem object from one that already exists using the CopyItemToDocument method of NotesItem or using the CopyItem or ReplaceItemValue methods of NotesDocument. You can also create one from scratch using the New method of NotesItem or using the AppendItemValue or ReplaceItemValue methods of NotesDocument.

You must call Save on the document if you want the modified document to be saved to disk. The document will not display the new item in the user interface unless there is a field of the same name on the form used to display the document.

Method	*Description*
New	Creates a new NotesItem object.
Abstract	Abbreviates the contents of a text item.
AppendToTextList	For an item that's a text list, adds a new value to the item without erasing any existing values.
Contains	Given a value, checks if the value matches at least one of the item's values exactly.
CopyItemToDocument	Copies an item to a specified document.
Remove	Permanently deletes an item from a document.

Property	*Description*
DateTimeValue	Read-Write. For a date-time item, returns a NotesDateTime object representing the value of the item. For items of other types, returns Nothing.
IsAuthors	Read-only. Indicates whether or not an item is of type Authors. An Authors item contains a list of Notes user names, indicating people who have Author access to a particular document.
IsEncrypted	Read-Write. Indicates if an item is encrypted.

Continued

Property	*Description*
IsNames	Read-only. Indicates if an item is a Names item. A Names item contains a list of Notes user names.
IsProtected	Read-Write. Indicates if a user needs at least Editor access to modify an item.
IsReaders	Read-Write. Indicates whether or not an item is of type Readers. A Readers item contains a list of Notes user names, indicating people who have Reader access to a particular document.
IsSigned	Read-Write. Indicates if an item contains a signature.
IsSummary	Read-Write. Indicates if an item can appear in a view or folder.
LastModified	Read-only. Date of the last modification.
Name	Read-only. The name of an item.
Parent	Read-only. The document that contains an item.
SaveToDisk	Read-Write. If true, then the item is written to disk on document save.
Text	Read-only. A plain text representation of an item's value.
Type	Read-only. The data type of an item.
ValueLength	Read-only. The size of an item's value in bytes.
Values	Read-Write. The value(s) that an item holds.

Note The properties listed in ***italics*** will be available in Release 4.5.

NotesLog

NotesLog enables us 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).

NotesLog is contained by NotesSession. To create a new log, you can use the New method of NotesLog or the CreateLog method of NotesSession. When you create a log using New, you can use one of the following methods to open the log:

- To log to a database, use the OpenNotesLog method.
- To log to a mail memo, use the OpenMailLog method.
- To log to a file (only available to scripts running locally), use the OpenFileLog method.

Notes does not automatically log actions or errors. You must explicitly log each action and error using the following methods:

- To log an action, use LogAction.
- To log an error, use LogError.

Method	*Description*
New	Creates a NotesLog object.
Close	Closes a log.
LogAction	Records an action in a log.
LogError	Records an error in a log.
LogEvent	Sends a Notes event out to the network. Only scripts running on a server can use this method.
OpenAgentLog	Starts logging an agent event.
OpenFileLog	Starts logging to a specified disk file. This method returns an error if you call it on a server.
OpenMailLog	Opens a new mail memo for logging. The memo is mailed when the log's Close method is called, or when the object is deleted.
OpenNotesLog	Opens a specified Notes database for logging.

Property	*Description*
LogActions	Read-Write. Indicates if action logging is enabled or not.
LogErrors	Read-Write. Indicates if error logging is enabled or not.
NumActions	Read-only. The number of actions logged so far.
NumErrors	Read-only. The number of errors logged so far.
OverwriteFile	Read-Write. For a log that records to a file, indicates if the log should write over the existing file or append to it. This property has no effect on logs that record to a mail message or database.
ProgramName	Read-Write. The name that identifies the script whose actions and errors you're logging. The name is the same as the name specified with New or CreateLog.

Note The methods listed in ***italics*** will be available in Release 4.5.

NotesName

The NotesName class will be available in Release 4.5. NotesName is contained by NotesSession. All the properties of NotesName are Read-only.

Method	*Description*
New	Creates a new NotesName object.

Property	*Description*
Abbreviated	String - abbreviated format of hierarchical name
ADMD	Administration management domain name part of name
Canonical	String - canonical format of hierarchical name
Common	String - common name (CN=) component of hierarchical name
Country	String - common name (CN=) component of hierarchical name
Generation	Generation part of a name, for example, “Jr.”
Given	Given name part of a name
Initials	Initials part of name
IsHierarchical	True if the name is hierarchical
Keyword	Country \ organization \ organizational unit 1 \ organizational unit 2 \ organizational unit 3 \ organizational unit 4
Organization	String - organization (O=) component of hierarchical name
OrgUnit1	String - first organizational unit (OU=) component of hierarchical name
OrgUnit2	String - second organizational unit (OU=) component of hierarchical name
OrgUnit3	String - third organizational unit (OU=) component of hierarchical name
OrgUnit4	String - fourth organizational unit (OU=) component of hierarchical name
PRMD	Private management domain name part of name
Surname	Surname part of name

NotesNewsletter

NotesNewsletter is contained by NotesSession and contains NotesDocument.

To create a new NotesNewsletter object, you use NotesDocumentCollection containing the documents you want, or the New method of NotesNewsletter, or the CreateNewsletter method of NotesSession. If you create a newsletter using New, there are two things you can do with it:

- Use the FormatDocument method to create a new document with a rendering (picture) of one of the documents in the collection.
- Use the FormatMsgWithDoclinks method to create a new document with links to each of the documents in the collection.

Method	*Description*
New	Creates a new NotesNewletter object.
FormatDocument	Creates a new document in the given database, containing a rendering (picture) of a specified document in the newsletter's collection.
FormatMsgWithDoclinks	Creates a newsletter document in the given database that contains a link to each document in the newsletter's collection.

Property	*Description*
DoScore	Read-Write. For a newsletter document created using the FormatMsgWithDoclinks method, indicates if the newsletter includes each document's relevance score.
DoSubject	Read-Write. For a newsletter document created using the FormatMsgWithDoclinks method, indicates if the newsletter includes a string describing the subject of each document.
SubjectItemName	Read-Write. For a newsletter document created using the FormatMsgWithDoclinks method, indicates the name of the item on a newsletter's documents which contains the text you want to use as a subject line.

NotesRichTextItem

NotesRichTextItem is a derived class based on NotesItem. It is contained by NotesDocument and contains NotesEmbeddedObject.

To create a new NotesRichTextItem object, use the New method of NotesRichTextItem or the CreateRichTextItem method of NotesDocument. Given a document, New creates a rich text item in the document, with the name you specify.

To access an existing NotesRichTextItem object, use the GetFirstItem and GetNextItem methods of NotesDocument.

Because NotesRichTextItem inherits from NotesItem, all of the NotesItem properties and methods can be used on a NotesRichTextItem, too.

When you change the value of a NotesRichTextItem object, the change is not written to disk until you call the Save method for the parent NotesDocument.

Method	*Description*
New	Creates a new NotesRichTextItem object.
AddNewLine	Appends one or more new lines (carriage returns) to the end of a rich text item.
AddTab	Appends one or more tabs to the end of a rich text item.
AppendDocLink	Given a database, view, or document to link to, adds a link to the end of a rich text item.
AppendRTFile	Appends the contents of a rich text file (composed of compound document records) to the end of a rich text item.
AppendRTItem	Appends the contents of one rich text item to the end of another rich text item.
AppendText	Appends text to the end of a rich text item. The text is rendered with the current style of the item (such as bold, or italicized).
EmbedObject	Given the name of a file or an application, does one of the following: Attaches the file you specify to a rich text item, or embeds an object in a rich text item. The object is created using either the application or the file you specify, or places an object link in a rich text item. The link is created using the file you specify.
GetEmbeddedObject	Given the name of a file attachment, embedded object, or object link in a rich text item, returns the corresponding NotesEmbeddedObject.
GetFormattedText	Returns the contents of a rich text item as plain text.

Property	*Description*
EmbeddedObjects	Read-Write. All the embedded objects, object links, and file attachments contained in a rich text item.

Note When you use the EmbedObject method of NotesRichTextItem in Visual Basic using Notes classes through OLE Automation, the argument type of the method is type number rather than type constant used in LotusScript.

Type Constant	*Type Number*
EMBED_ATTACHMENT	1454
EMBED_OBJECT	1453
EMBED_OBJECTLINK	1452

Tip If Visual Basic cannot recognize a type constant, you can always use **`Message Cstr(TypeConstant)`** in LotusScript to find out the type number.

NotesSession

The NotesSession class represents the Notes environment of the current script, providing access to environment variables, address books, information about the current user, and information about the current Notes platform and release number.

It contains NotesAgent, NotesDatabase, NotesDbDirectory, NotesDateTime, NotesLog, and NotesNewsletter. In Notes Release 4.5, it will contain the NotesInternational and NotesDateRange classes.

Method	*Description*
New	Creates a new NotesSession object.
Close	Closes a session. Any objects contained within the session become invalid once you close it.
CreateDateRange	Given a NotesDateRange object which represents the date range you want.
CreateDateTime	Given a string that represents the date and time you want, creates a new NotesDateTime object that represents that date and time. When used with OLE automation, this method allows you to create a NotesDateTime object without using New.
CreateLog	Creates a new NotesLog object with the name you specify. When used with OLE automation, this method allows you to create a NotesLog object without using New.

Continued

Method	*Description*
CreateNewsletter	Given a NotesDocumentCollection containing the documents you want, creates a new NotesNewsletter. When used with OLE automation, this method allows you to create a NotesNewsletter object without using New.
FreeTimeSearch	Given a NotesDateRange array.
GetDatabase	Creates a NotesDatabase object that represents the database located at the server and file name you specify, and opens the database, if possible. When used with OLE automation, this method allows you to create a NotesDatabase object without using New. It does not create a new database on disk.
GetDBDirectory	Creates a new NotesDbDirectory object using the name of the server you want to access. When used with OLE automation, this method allows you to create a NotesDbDirectory object without using New.
GetEnvironmentString	Given the name of a string environment variable, retrieves its value.
GetEnvironmentValue	Given the name of a numeric environment variable, retrieves its value.
SetEnvironmentVar	Sets the value of a string or numeric environment variable.
UpdateProcessedDoc	Marks a document as processed by an agent.

Property	*Description*
AddressBooks	Read-only. The Address Books, both public and personal, that are known to the current script.
CommonUserName	Read-only. The common name portion of the current user's name.
CurrentAgent	Read-only. The agent that's currently running.
CurrentDatabase	Read-only. The database in which the current script resides. This database may or may not be open.
DocumentContext	Read-only. If the agent is invoked from the Notes API, then this might contain a NotesDocument. Used to pass arguments to an agent.

Continued

Property	*Description*
EffectiveUserName	Read-only. The user name that is in effect for the current script.
IsOnServer	Read-only. Indicates if a script is running on a server.
International	Read-only. Returns a NotesInternational object.
LastExitstatus	Read-only. The exit status code returned by the Agent Manager the last time the current agent ran.
LastRun	Read-only. The date when the current agent was last executed.
NotesVersion	Read-only. The release of Notes in which the current script is running.
Platform	Read-only. The name of the platform in which the current script is running.
SavedData	Read-only. A document that an agent script uses to store information in-between invocations. The script can use the information in this document the next time the script runs.
UserName	Read-only. The current user's name.

Note The properties and methods listed in ***italics*** will be available in Release 4.5. The SetEnvironmentVar method will have a new optional argument, system, in Release 4.5. The ***Close*** method will be removed in Release 4.5.

NotesTimer

The NotesTimer class will be available in Release 4.5.

Method	*Description*
New	Creates a new NotesTimer object.

Property	*Description*
Comment	Read-Write. Any string to identify the timer.
Enabled	Read-Write. True if the timer is enabled.
Interval	Read-Write. Number of seconds in the timer's interval.

NotesView

The NotesView class is contained by NotesDatabase, and contains NotesDocument and NotesViewColumn. It will contain NotesDocumentCollection in Release 4.5.

You access a view or folder through the NotesDatabase object that contains the view or folder. There are two ways:

- To access a view or folder when you know its name or synonym, use the GetView method of NotesDatabase.
- To access all the views and folders in a database, use the Views property of NotesDatabase.

Both options return NotesView objects that represent public view(s) and/or folder(s) in the database. If a script runs on a workstation, the NotesView object may also represent personal views and folders.

To access a view or folder when we have a document that was retrieved from the view or folder, use the ParentView property of NotesDocument.

Both NotesDocumentCollection and NotesView provide access to documents within a database. Use the NotesView object if:

- There is a view or folder in the database that contains all the documents you want to search.
- You need to navigate through the documents' response hierarchies.
- You want to access documents as quickly as possible.
- You want to find a document by its key in a view.
- You want to access documents in sorted order.

Views are the more efficient means of accessing documents because they are already indexed by the database itself.

Method	*Description*
Clear	Clears the full text search filtering on a view.
FTSearch	Conducts a full text search on all documents in a view and filters the view so that it represents only those documents that match the full text query. This method does not find word variants.
GetAllDocumentsByKey	Finds all documents based on its Column value in a view.
GetChild	Given a document in a view, returns the first response to the document.
GetDocumentByKey	Finds a document based on its column values within a view. You create an array of strings (keys), where each key corresponds to a value in a sorted and categorized column in the view. The method returns the first document whose column values match each key in the array.
GetFirstDocument	Returns the first document in a view. This is the same document you see when you scroll to the top of the view in the Notes user interface.
GetLastDocument	Returns the last document in a view. This is the same document you see when you scroll to the bottom of the view in the Notes user interface.
GetNextDocument	Given a document in a view, returns the document immediately following it.
GetNextSibling	Given a document in a view, returns the document immediately following the given document at the same level. If you send the method a main document, the next main document in the view is returned; if you send a response document, the next response document with the same parent is returned.
GetNthDocument(n)	Given a number, returns the document at the given position in a view.
GetParentDocument(doc)	Given a response document in a view, returns its parent document.
GetPrevDocument	Given a document in a view, returns the document immediately preceding.

Continued

Method	*Description*
GetPrevSibling	Given a document in a view, returns the document immediately preceding the given document at the same level.
MarkRead	Marks the document as read.
MarkUnread	Marks the document as unread.
Refresh	Updates a view's contents with any changes that have occurred to the database since the NotesView object was created, or since the last Refresh.
Remove	Permanently deletes a view from a database.

Property	*Description*
Aliases	Read-only.The alternative names for the view.
AutoUpdate	Read-Write. Boolean. If true, the view will update itself automatically.
NotesViewColumnArray	Read-only. All the columns in a view.
Created	Read-only. The date that a view was created.
IsCalendar	Read-only. Indicates whether or not a view is a calendar view.
IsDefaultView	Read-only. Indicates whether or not a view is the default view of the database.
IsFolder	Read-only. Indicates whether a NotesView object represents a folder.
LastModifed	Read-only. The date that a view was last modified.
Name	Read-only. The name of a view.
Parent	Read-only. The database to which a view belongs.
ProtectReaders	Read-Write. If true, does not replace $Readers item during replication.
Readers	Read-Write. Indicates the readers of the view.
UniversalID	Read-only. The Universal ID of a view, which is a 32-character combination of letters and numbers that uniquely identifies a view across all replicas of a database.

Note The methods and properties listed in ***italics*** will be available in Release 4.5. The max argument of FTSearch will be optional in Release 4.5. The GetDocumentByKey method will provide a new optional argument, exact, in Release 4.5.

NotesViewColumn

The NotesViewColumn class is contained by NotesView. You access an existing NotesViewColumn object through the view or folder that contains it. Use the Columns property of NotesView.

Property	*Description*
Formula	Read-only. The @function formula for a column, if it exists.
IsCategory	Indicates if a column is categorized.
IsHidden	Indicates if a column is hidden.
IsRespones	Read-only. Indicates if a column contains only response documents.
IsSorted	Read-only. Indicates if a column is sorted.
ItemName	Read-only. The name of the item whose value is shown in the column. For columns whose values are simple functions or formulas, returns an automatically generated internal name.
Position	Read-only. The position of a column in its view. Columns are numbered from left to right, starting with 1.
Title	Read-only. The title of a column, if any.

Notes UI (front_end) Classes

There are two Notes UI classes.

Class	*Description*
NotesUIWorkspace	Provides access to the current workspace.
NotesUIDocument	Models the behavior of a Notes document window.
NotesUIView	Provides access to the current view.
NotesUIDatabase	Provides access to the current database.

Note The classes listed in ***italics*** will be available in Release 4.5.

NotesUIWorkspace

NotesUIWorkspace contains NotesUIDocument. To create a new NotesUIWorkspace object, we use the New method.

Method	*Description*
New	Creates a new NotesUIWorkspace object.
AddDatabase	Adds a database to the workspace.
CheckAlarms	Checks the alarms.
ComposeDocument	Using a database and form you specify, creates a new document and displays it on the workspace.
DialogBox	Brings up a dialog box that displays the current document (either open or selected in a view) using a form you specify. The user interacts with the form and document as usual, selecting OK or Cancel when finished.
EditDocument	Opens the current document in a mode you specify.
EditProfile	Edits the profile document.
EnableAlarms	Enables the alarms.
OpenDatabase	Opens a database to a view you specify.
URLOpen	Opens a URL.

Property	*Description*
CurrentDocument	Returns a NotesUIDocument object representing the document that's currently open.

Note The methods listed in ***italics*** will be available in Release 4.5. The ComposeDocument and DialogBox, methods will have new arguments. The EditDocument method will have two new optional arguments, document and Readonly. The OpenDatabase method will provide several options in Release 4.5.

NotesUIDocument

NotesUIDocument is contained by NotesUIWorkspace and contains NotesDocument. There are four ways to get the current document:

- You can use the CurrentDocument property of NotesUIWorkspace.
- If you are writing a script that responds to a NotesUIDocument event, use the source parameter from one of the events.

- If you want to create a new document in a database and access it, use the ComposeDocument method of NotesUIWorkspace.
- If you want to open the currently highlighted document and access it, use the EditDocument method of NotesUIWorkspace.

Method	*Description*
Categorize	Given the name of a category, places a document in the category.
Clear	Deletes the current selection from a document. The current selection can be anything in an editable field, such as text or graphics.
Close	Closes a document. The NotesUIDocument is no longer available once you call this method.
CollapseAllSections	Collapses all the sections in a document.
Copy	Copies the current selection in a document to the Clipboard. The current selection can be anything in the document, such as text or graphics.
CreateObject	In a document in edit mode, creates an OLE object in the current rich text field.
Cut	Cuts the current selection from a document and places it on the Clipboard. The current selection can be anything in an editable field, such as text or graphics.
DeleteDocument	Marks the current document for deletion and closes it. The NotesUIDocument object is no longer available once you call this method.
DeselectAll	Deselects any selections in a document.
ExpandAllSections	Expands all the sections in a document.
FieldAppendText	Appends a text value into a field in a document without removing the existing contents of the field.
FieldClear	Clears the contents of a field in a document.
FieldContains	In an open document, checks if a field contains a specific text value.
FieldGetText	In a document in read or edit mode, returns the contents of a field you specify, as a string. If the field is of type numbers or date-time, its contents are converted to a string.
FieldSetText	Sets the value of a field on a document. The existing contents of the field, if any, are written over.

Continued

Method	*Description*
FindFreeTimeDialog	Finds free time dialog.
Forward	Creates a new mail memo with the contents of a document. The user can enter recipients and mail the forwarded document like any other mail memo.
GetObject	Given a name, returns a handle to the OLE object of that name.
GotoBottom	Places the cursor in the last editable field in a document.
GotoField	Given a field name, puts the cursor in the specified field in a document.
GotoNextField	Places the cursor in the next field in a document. The next field is the one below and to the right of the current field.
GotoPrevField	Places the cursor in the first editable field in a document.
GotoTop	Places the cursor in the first editable field in a document.
InsertText	Inserts a text value at the current cursor position in a document.
Paste	Pastes the contents of the Clipboard at the current cursor position in a document.
Print	Prints the current document. If one or more parameters are specified, automatically prints the document. If no parameters are specified, or if the first parameter is omitted, displays the File Print dialog box.
Refresh	Refreshes a document. When you refresh a document, its computed fields are recalculated.
RefreshHideFormulas	Recalculates the hide-when formulas in the current document's form.
Reload	Refreshes the current document with any changes that have been made to the corresponding back-end document. Refreshing the current document updates its representation in memory, and visually on the workspace, to reflect the changes that have been made to the back-end document.
Save	Saves a document.

Continued

Method	*Description*
SaveNewVersion	Saves a copy of a document as a new version.
SelectAll	In a document in edit mode, selects the entire contents of the current field. In a document in read mode, selects the entire contents of the document.
Send	Mails a document.

Property	*Description*
AutoReload	Read-Write. Indicates whether or not the current document should be refreshed whenever the corresponding back-end document changes.
CurrentField	Read-only. The name of the field that the cursor is in.
Document	Read-only. The back-end document that corresponds to the currently open document.
EditMode	Read-Write. Indicates if a document is in edit mode.
FieldHelp	Read-Write. Indicates if field help for a document is displayed.
HiddenChars	Read-Write. Indicates if the hidden characters in a document (such as tabs and carriage returns) are displayed.
HorzscrollBar	Read-Write. Indicates if a document's horizontal scroll bar is displayed.
IsNewDoc	Read-only. Indicates if a document is new. A new document is one that hasn't been saved.
InPreviewPane	True if the document is in the preview pane.
PreviewDocLink	Read-Write. Indicates if a link's preview pane is displayed.
PreviewParentDoc	Read-Write. Indicates if the parent document's preview pane is displayed.
Ruler	Read-Write. Indicates if the ruler is displayed.
WindowTitle	Read-only. The window title of a document.

NotesUIDatabase

The NotesUIDatabase class will be available in Release 4.5. It is contained by NotesUIWorkspace and contains NotesDatabase and NotesDocumentCollection.

Method	*Description*
OpenView(viewname)	Opens a view in the current database by view name.

Property	*Description*
Database	Read-only. Refer to back-end database object of the current database.
Documents	Read-only. Indicates all the document arrays in the current database.

NotesUIView

The NotesUIView class will be available in Release 4.5. It is contained by NotesUIDatabase and contains NotesView and NotesDocumentCollection.

Property	*Description*
CalendarDateTime	Read-only. Indicates the calendar date time.
Documents	Read-only. Indicates the back-end document collection of the current view.
View	Read-only. Indicates the back-end view of the current view.

Chapter 4
Lotus Notes as an OLE 2 Automation Server

You can develop Notes applications in Visual Basic using the Notes classes. This chapter shows you how to do this.

In this chapter, we will discuss how to:

- Use Notes classes in Visual Basic
- Declare Notes classes in Visual Basic
- Use the Notes class hierarchy in Visual Basic
- Handle errors in Visual Basic using Notes classes.

Note The software environment used in this chapter is Windows 95, Lotus Notes 4.1 (32-bit) and Visual Basic 4.0 (32-bit).

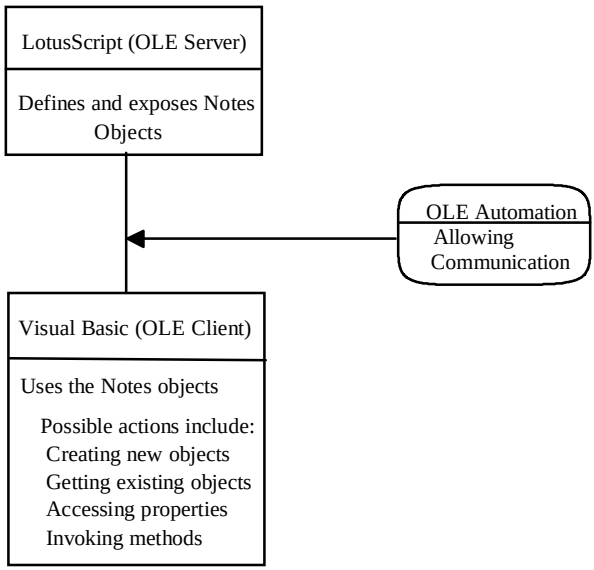
Using Notes Classes in Visual Basic through OLE Automation

We use OLE automation to develop Notes applications in Visual Basic using Notes classes.

OLE automation is a widely accepted object model used by many applications to provide objects in a consistent way to other applications, development tools, and macro languages. OLE servers expose their objects to other applications, whereas client applications use those objects. Objects contain data (properties) and the functions (methods) used to perform an action with the object. You can call upon an OLE automation object's code to perform a variety of tasks that you do not want to, or cannot, perform in your own code.

Lotus Notes Release 4 is an OLE automation server. You can develop your own Notes applications in Visual Basic and manipulate objects in LotusScript.

The following figure shows how it works:



How Visual Basic and LotusScript Work with OLE Automation

You can create a reference to Notes objects, such as NotesDatabase, NotesACL, NotesLog and NotesDocument, in your code. NotesSession, and NotesUIWorkspace are objects that can be created externally. You can set a reference to them using the CreateObject method from outside the LotusScript that created the object.

Those objects are at the highest level of the Notes object hierarchy. Use a method from a higher level object in the hierarchy to get a reference to other objects, such as NotesDatabase, NotesDbDirectory, NotesACL, NotesLog, NotesItem, and NotesUIDocument.

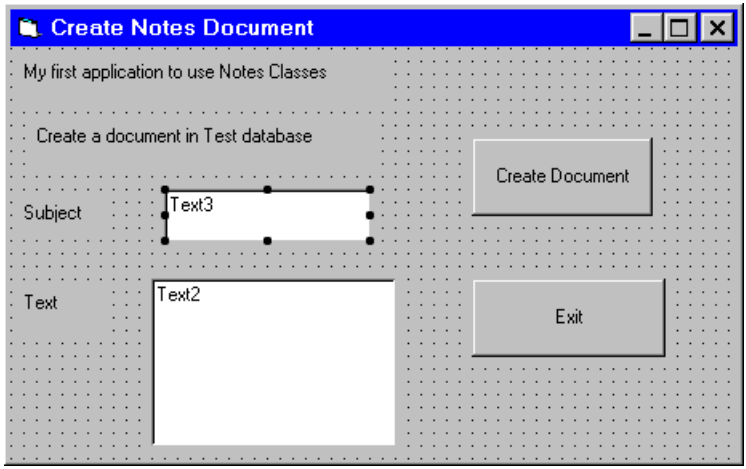
After you have declared a variable that references an OLE automation object, the object can be manipulated in Visual Basic using the object.property syntax to set and return the object's property values, or by using the objec.method syntax to use methods on the object.

When you are finished using a Notes object, clear any variables that reference the Notes objects so the Notes object can be released from memory. To clear a variable, set it to Nothing.

Let's try it.
Let's create our first Visual Basic application using Notes classes. In this example we use a TEST database which was created using the Discussion

(R4) template. The database file - test.nsf - is in your default Notes database directory. For information on how to create the database, refer to Chapter 5.

1. Use Visual Basic to create a form like the following:



2. Put the following code on the Create Document button:

```
Private Sub Command1_Click()
    Dim session As Object
    Dim db As Object
    Dim doc As Object
    Set session = CreateObject("Notes.NotesSession")
    Set db = session.GetDatabase("", "test.nsf")
    Set doc = db.CreateDocument()
    doc.Form = "Main Topic"
    doc.Subject = Form1.Text3.Text
    doc.Body = Form1.Text2.Text
    Call doc.Save(True, False)
End Sub
```

Following is an explanation of the code.

To access the Notes objects, we declare them as OLE objects in Visual Basic.

```
Dim session As Object
Dim db As Object
```

```
Dim doc As Object
```

We create a reference to the Notes object that is created externally.

```
Set session = CreateObject("Notes.NotesSession")
```

We use the Notes object hierarchy to create the object db to access the Notes database TEST on the local server. For the GetDatabase method, the first parameter is the server name, the second parameter is the database file name. If the server is a local server, specify an empty string, as in our example.

```
Set db = session.GetDatabase("", "test.nsf")
```

We use the Notes hierarchy to create the doc object to access a Notes document.

```
Set doc = db.CreateDocument()
```

We use the Main Topic form to create the document.

```
doc.Form = "Main Topic"
```

Fill the Subject and Body text fields with the contents of the text boxes in the Visual Basic application.

```
doc.Subject = Form1.Text3.Text
```

```
doc.Body = Form1.Text2.Text
```

We use the Save method to save the document. We want the document to be saved even if someone else edits and saves the document while the application is running. So we set True in the first argument. Otherwise when someone else edits the document while the application is running, the second argument determines what happens.

```
Call doc.Save(True, False)
```

3. Put the following code on the Exit command button:

```
Private Sub Command2_Click()
```

```
Set db = Nothing
```

```
Set session = Nothing
```

```
Unload Form1
```

```
End Sub
```

Following is an explanation of the code.

We release object db and object session before we exit the application.

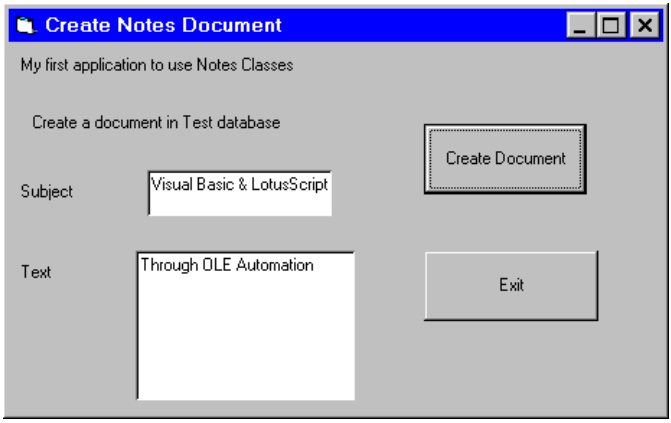
```
Set db = Nothing
```

```
Set session = Nothing
```

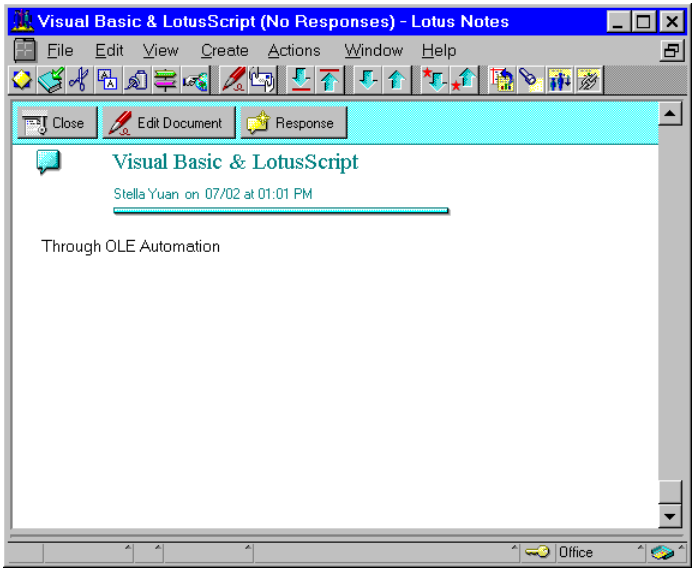
We exit the application and unload form1.

```
Unload Form1
```

4. Let's have a look at how the application works. Choose Run - Start from the menu to run the application.
5. Fill the Subject field and Body field on the form.
6. Click the Create Document button to create a new document in the Test database.



7. Look at your Test database on your Notes workspace. A new document has been created. It looks like this:



Declaring Notes Classes in Visual Basic

When we use LotusScript Notes objects in Visual Basic, the declaration of the Notes objects is different from the declaration in LotusScript. In LotusScript, we declare Notes objects as objects of their specific Notes classes. For example:

```
Dim session As NotesSession

Dim db As NotesDatabase

Dim doc As NotesDocument
```

But in Visual Basic, we declare Notes objects simply as an object. Then we set a reference to the object. For example:

```
'declare Notes objects as Object

Dim session As Object

Dim db As Object

'set a reference to the Notes Objects

set session = CreateObject ("Notes.NotesSession")

set db = session.GetDatabase("","test.nsf")
```

Note By changing the declaration of the Notes objects, you can run the Visual Basic code in LotusScript.

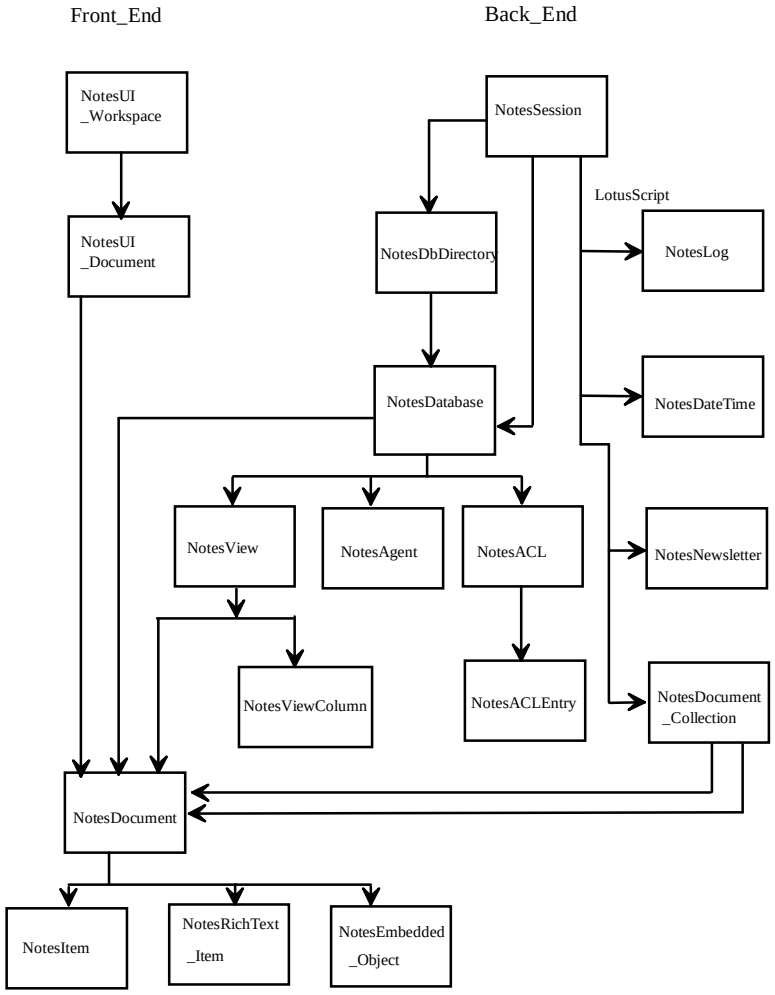
As we declare Notes objects as objects in Visual Basic, we cannot use the New method of the classes. We use the New method to declare a new object in LotusScript. For example:

```
Dim db as New NotesDatabase("","discuss4.ntf")

Dim doc as New NotesDocument
```

Using the Notes Class Hierarchy in Visual Basic

There is a hierarchical relationship for object classes. Higher hierarchical classes contain the lower ones. Each class has defined members, properties and methods. Using these members, you can access other objects. The relationship of containment and access means that the higher class 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) in the workspace 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). The following figure shows the hierarchical relationship for Notes object classes.



Note This chart only includes the 18 classes available in Lotus Notes Release 4.1. There will be an additional six classes available in Release 4.5. Four of them can be used in Visual Basic: NotesInternational, NotesName, NotesDateRange and NotesTimer.

In Visual Basic, we can access the Notes objects in one of two ways:

- Directly, if the object can be created externally.
- Indirectly, if the object is a dependent object. You can get a reference to it from another object higher in the Notes class hierarchy.

NotesSession and NotesUIWorkspace are object classes that can be created externally. We can access these two classes directly from Visual Basic using the CreateObject function. NotesSession and NotesUIWorkspace are at the highest level of the hierarchy. Other object classes are dependent object classes, and can be accessed only by using a method or a property of one of the two classes.

In our previous example, Create Document, we created the externally creatable object session first. Then we used a method of the session type object to access the lower-level class db.

Let's look at some examples:

Example

This example accesses the current document in Notes Workspace.

```
Dim workspace As Object

Dim UIdoc as Object

set workspace = CreateObject ("Notes.NotesUIWorkspace")

set UIdoc = workspace.CurrentDocument()
```

To access the current document (NotesUIDocument object) in Visual Basic, we first need to get a reference from NotesUIWorkspace. Then we use the CurrentDocument property of the NotesUIWorkspace class to get the reference to the NotesUIDocument class. We declare workspace as an object which references NotesUIWorkspace. We declare UIdoc as an object which references NotesUIDocument.

```
Dim workspace As Object

Dim UIdoc As Object
```

Create an externally creatable object.

```
set workspace = CreateObject ("Notes.NotesUIWorkspace")
```

Use the externally creatable object to access a lower-level object doc.

```
set UIdoc = workspace.CurrentDocument()
```

Example

This example accesses the Notes view called “By Author”.

```
Dim session As Object
Dim db As Object
Dim view As Object
Set session = CreateObject ("Notes.NotesSession")
Set db = session.GetDatabase("","test.nsf")
Set view = db.GetView ( "By Author")
```

To access the NotesView class, we will get the reference to NotesView from the NotesDatabase class. Since NotesDatabase is not an externally creatable object, we will get its reference from a higher-level class of the hierarchy, which is NotesSession. We get the reference to NotesDatabase using the GetDatabase method of the NotesSession class. Then we get the reference to NotesView using the GetView method of NotesDatabase.

We declare session as an object which references NotesSession.

```
Dim session As Object
```

We declare db as an object which references NotesDatabase.

```
Dim db As Object
```

We declare view as an object which references NotesView.

```
Dim view As Object
```

We get the reference to NotesSession by using the CreateObject function of Visual Basic, and store it in session.

```
Set session = CreateObject ("Notes.NotesSession")
```

We get the reference to NotesDatabase using the GetDatabase method of NotesSession, and store it in db.

```
Set db = session.GetDatabase("","test.nsf")
```

We get the reference to NotesView and store it in view.

```
Set view = db.GetView ( "By Author")
```

Example

This example shows how we access NotesItem in the current Notes document.

```
Dim workspace As Object
Dim UIdoc As Object
```

```
Dim doc As Object
Dim item As Object
Set workspace = CreateObject ("Notes.NotesUIWorkspace")
Set UIdoc = workspace.CurrentDocument
Set doc = UIdoc.Document
Set item = doc.GetFirstItem ("Subject")
```

The item we are accessing is an item in the current document. So we need to get the document reference from NotesUIDocument. This is not an externally creatable object. Therefore, we first get the reference from NotesUIWorkspace to get the reference to NotesUIDocument and then to the other lower-level classes.

We declare workspace, UIdoc and doc as object.

```
Dim workspace As Object
Dim UIdoc As Object
Dim doc As Object
```

We declare item as an object which references NotesItem.

```
Dim item As Object
```

We get the reference from NotesUIWorkspace and NotesUIDocument.

```
Set workspace = CreateObject ("Notes.NotesUIWorkspace")
Set UIdoc = workspace.CurrentDocument
```

We get a reference to NotesDocument using the Document property of NotesUIDocument, and store it in doc.

```
Set doc = UIdoc.Document
```

We get reference to NotesItem using the GetFirstItem method of NotesDocument to get the first Item named “Subject”.

```
Set item = doc.GetFirstItem ("Subject")
```

Note When using Notes classes through OLE automation in Visual Basic, for some methods, you must use type numbers in arguments rather than type constants which are used in LotusScript. However, you can always get the type number using **`"Messagebox Cstr(TypeConstant)"`** in LotusScript. For more information about type numbers, refer to Chapter 3.

Error Handling

There are three kinds of errors you can encounter:

- Compile errors
- Run-time errors
- Logic errors

Compile Errors

Compile errors result from incorrectly constructed code. If you mistype a keyword, omit some necessary punctuation, or use a Next statement without a corresponding For statement, Visual Basic detects these errors when you compile the application.

Compile errors include errors in syntax. If you have selected the Auto Syntax Check option in the Environment tab on the Options dialog box, Visual Basic will display an error message as soon as you enter a syntax error in the code window.

To set the Auto Syntax check option:

1. From the Tools menu, select Options, and click the Environment tab on the Options dialog box.
2. Select Auto Syntax Check.

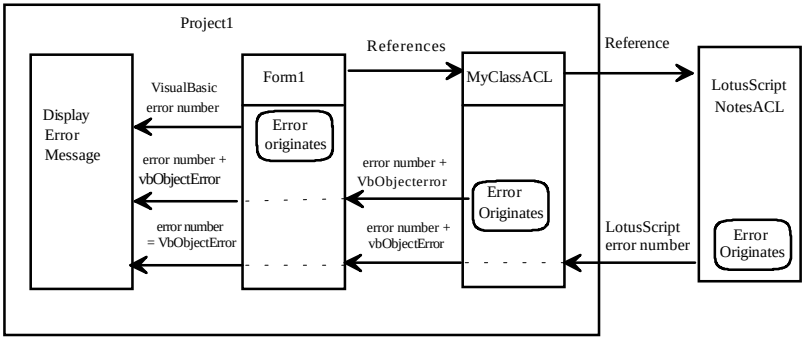
Run-time Errors

Run-time errors occur (and are detected by Visual Basic) when a statement attempts an operation that is impossible to carry out.

When we use Notes object classes in Visual Basic the error-handling code is especially important because code from LotusScript is used remotely from within your Visual Basic application. Where possible, you should include code to handle errors that LotusScript may generate. When there is a run-time error generated in a Notes object, LotusScript handles the errors it can handle, and regenerates the errors which it cannot handle. Visual Basic will automatically remap untapped errors, and the error-handling code in your application is needed to handle the errors.

In applications that use Notes objects and derived classes based on Notes objects, it becomes more difficult to determine where an error occurs, particularly if it occurs in a Notes LotusScript object.

The following figure shows a Visual Basic application that consists of a form module, which references a class module, which in turn references a LotusScript Notes ACL object.



Regenerating Errors between Forms, Classes, and OLE Automation Notes Objects

LotusScript handles the part of the error arising in the Notes object. If LotusScript objects cannot handle a particular error arising in the NotesACL, but regenerate it instead, Visual Basic will pass the error to the referencing object, MyClassACL. Visual Basic automatically remaps untapped errors arising in objects outside of Visual Basic as error code 440.

The MyClassACL object can either handle the error (preferable), or regenerate it. The OLE interface specifies that any object regenerating an error that arises in a referenced object should not simply propagate the error (pass it as error code 440), but should instead remap the error number to something meaningful. When you indicate the error condition, if your handler can determine what the error is, it should map it to an undefined error number. Add the new number to the intrinsic Visual Basic constant vbObject Error to notify other handlers that this error was raised by your object (MyClassACL).

Whenever possible, a class module (MyClassACL module in our case) should try to handle every error that arises in the object. It references errors that are not handled by that object. However, there are some errors that it cannot handle because it cannot anticipate them. There are also cases where it is more appropriate for the referencing object to handle the error, rather than the referenced object.

When an error occurs in the form module, Visual Basic raises one of the predefined Visual Basic error numbers as described in the online Help.

When you regenerate an error, leave the error object's other properties unchanged. If the raised error is not trapped, the Source and Description properties can be displayed to help the user take corrective action.

A class (MyClassALC) module might include the error handler to accommodate any error it might trap, regenerating those it is unable to resolve.

When you are debugging an application that has a reference to an OLE automation object or a class defined in a class module, you may find it confusing to determine which object generates an error. To make this easier, you can select the Break in Class module option on the Advanced tab of the Options dialog box (available from the Tools menu). With this option selected, an error arising in a form or standard module will invoke an error handler, if one is available. An error in a class module or an object in another application that is running in another instance of Visual Basic will cause that class to enter the debugger's break mode, allowing you to analyze the error. An error arising in a compiled object will not display the debug window in break mode; rather, such errors will be handled by the object's error handler, or trapped by the referencing module.

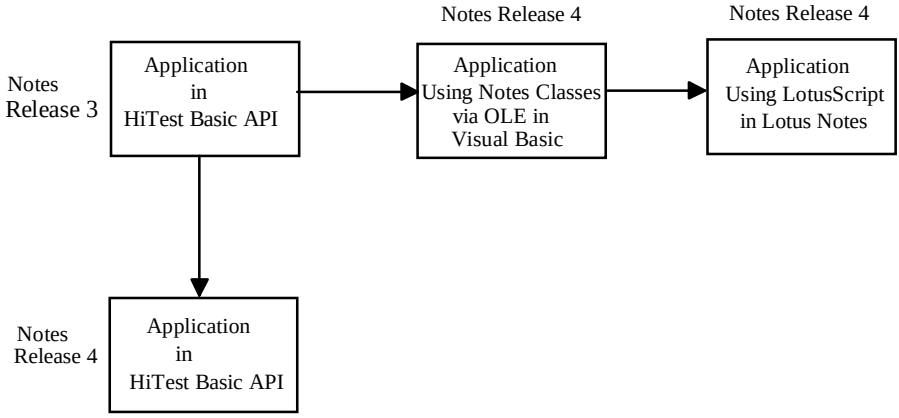
Logic Errors

Logic errors occur when an application doesn't perform the way it was intended. An application can have syntactically valid code, run without performing any invalid operations, and yet produce incorrect results. Only by testing the application and analyzing results can you verify that the application is performing correctly. You can use the Visual Basic Debugging tools to fix logic errors.

Chapter 5
Converting HiTest Applications

HiTest Tools for Visual Basic is a 16-bit product. It can only work with 16-bit Visual Basic and the 16-bit version of Lotus Notes (Lotus Notes Release 3 and Release 4 for Windows 3.1).

You can convert your Notes Release 3 applications writen in HiTest to Notes Release 4 applications in one of three ways, as shown in the following figure:



- You can use the original HiTest code directly in Notes Release 4. However, if you do this, you cannot use any of the Release 4 features, such as folders. Also, since HiTest is a 16-bit product, the HiTest application can only work with the 16-bit version of Lotus Notes Release 4.
- You can convert the HiTest application to an application using Notes classes through OLE automation in Visual Basic. This enables you to use the Release 4 features. However, some of the methods and properties provided with the Notes classes in LotusScript will not be available to you.
- You can convert to LotusScript. This enables you to use all the Notes Release 4 features.

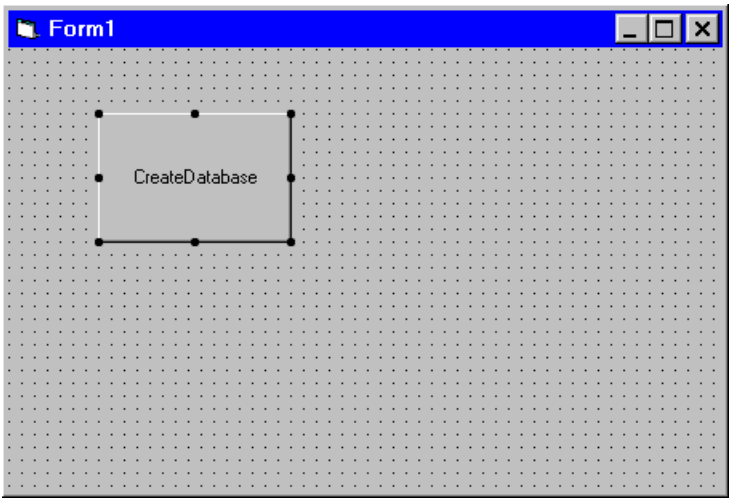
This chapter covers the above topics in more detail.

Creating a Notes Application Using the HiTest Basic API

The following example shows how to create a new database with a file name of TEST.nsf and a title of TEST.

Example

1. Create a new form in Visual Basic.
2. Create a button control for the form and set its caption property to CreateDatabase. It looks like this when you are done:



3. Add the following code to the button's Click Event procedure:

```
Dim hgdatabase As Long

Dim hgstatus As Long

hgstatus = HTInit(0)

hgstatus = HTDatabaseCreate("", "TEST.nsf", "TEST",
"testing", "", "DISCUSS4.NTF", HTDBCLASS_DB, 0&)

If hgstatus = 0 Then

Print "test.nsf has been created on your local server"

hgstatus = HTTerm(0)
```

Following is an explanation of the code.

We declare an hgdatabase variable that references a Notes database. In HiTest, we present a Notes database handle as long.

```
.Dim hgdatabase As Long
```

We declare an hgstatus variable. All the status codes we get later will be put in the variable.

```
Dim hgstatus As Long
```

For every HiTest application, we must initialize the Notes session as a first step. Other functions can only be called after session initialization. If the initialize is unsuccessful, it will return either HTFAIL_HITEST_VERSION (program built with incompatible HiTest API version) or HTFAIL_NOTES_VERSION (the HiTest API requires Notes Release 3.0 or higher) to hgstatus.

```
hgstatus = HTInit(0)
```

We create a standard Notes database on a local server. The database file name is TEST.nsf and its title is TEST. It returns a failure code to hgstatus if the function does not work successfully.

```
hgstatus = HTDatabaseCreate("", "TEST.nsf", "TEST",
"testing", "", "DISCUSS4.NTF", HTDBCLASS_DB, 0&)
```

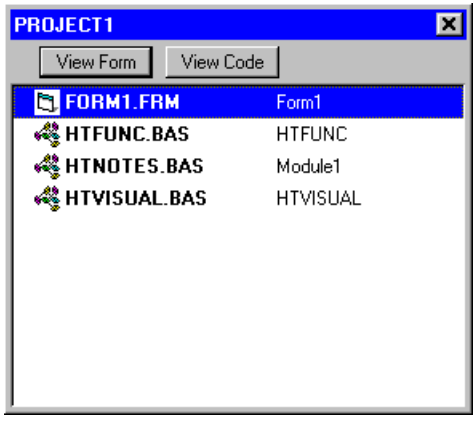
If hgstatus does not contain any failure code, a message is displayed.

```
If hgstatus = 0 Then

Print "test.nsf has been created on your local server"
```

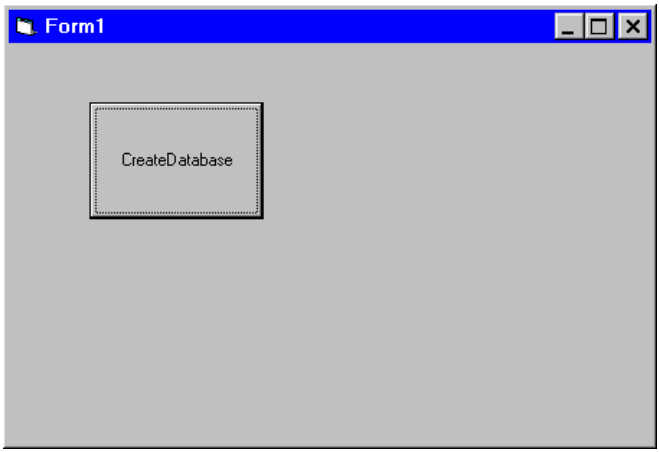
We close the Notes session. Each HTInit function must be closed. If there is a failure, the same failure code will be returned as for HTInit.

```
hgstatus = HTTerm(0)
```

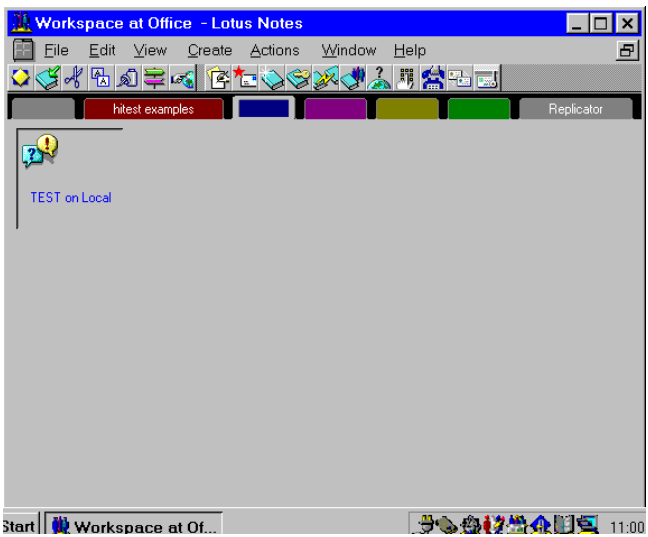
4. From the Visual Basic menu, choose File-Add File to add three modules to your project: HTFUNC.BAS, HTNOTES.BAS, HTVISUAL.BAS. They contain the HiTest object definitions, HiTest constant definitions, and HiTest function definitions.



5. Choose Run-Start to run the application. The following form appears:



6. Click the CreateDatabase button. A message confirming the creation of the new database displays.
7. Open the Notes workspace.
8. Choose File - Database - Open to add the new database called TEST to your workspace.



Converting a HiTest Application Using Notes Classes in Visual Basic through OLE Automation

Lotus Notes Release 4 is an OLE automation server. Converting your HiTest applications to LotusScript Notes classes through OLE automation enables you to develop and use your applications in all the environments supported by Visual Basic, such as Windows, Windows 95 and Windows NT.

To convert applications developed using the HiTest tools, you translate the HiTest Basic API functions to equivalent Lotus Notes functions. There is a number of LotusScript Notes class methods that perform functions equivalent to HiTest functions.

Note For more details, see Appendix 1.

By converting your HiTest applications, you can more easily access the new Release 4 Notes design elements. For example, HiTest has no support for accessing the current Notes document and Notes item. LotusScript Notes classes, on the other hand, include NotesUIWorkspace and NotesUIDocument which enable you to access the current Notes workspace and the current Notes document. You can access the current Notes item using the CurrentField property of NotesUIDocument.

Let's look at an example.

Example

This example shows how you can access the current Notes document in Visual Basic using LotusScript Notes classes through OLE automation. You will need to keep both Lotus Notes and the current document open. Otherwise, although the application will be able to open the Notes workspace, it will return an error message indicating that it cannot find the current document.

```
Dim uispace As Object

Dim uidoc As Object

Dim doc As Object

Set uispace = CreateObject("Notes.NotesUIWorkspace")

Set uidoc = uispace.CurrentDocument

Set doc = uidoc.Document

Print (doc.Created)

Set uispace = Nothing
```

To get the Subject item value of the current document (NotesUIDocument), we need to get the hierarchy from the current workspace

(NotesUIWorkspace). We declare uispace as an object which references the current workspace.

```
Dim uispace As Object
```

We declare uidoc as an object which references the current document.

```
Dim uidoc As Object
```

We declare doc as an object which references the document.

```
Dim doc As Object
```

We use the CreateObject function in Visual Basic to get the reference to NotesUIWorkspace, and store it in uispace.

```
Set uispace = CreateObject("Notes.NotesUIWorkspace")
```

We get the reference to NotesUIDocument using the CurrentDocument property of NotesUIWorkspace.

```
Set uidoc = uispace.CurrentDocument
```

We get the reference to NotesDocument using the Document property of NotesUIDocument.

```
Set doc = uidoc.Document
```

We print the current document's created data.

```
Print (doc.Created)
```

We release the references.

```
Set uispace = Nothing
```

You can use the Notes Release 4 features like folders; however, some methods, such as the New method of NotesDatabase, cannot be used here. You also cannot use the Forall and Foreach statements when using Notes classes through OLE automation in Visual Basic. This is because Visual Basic does not know the concept of a collection, which is a LotusScript way of accessing elements.

Let's take a look now at how to convert the Notes HiTest application created in the previous section to a Visual Basic application using Notes classes through OLE automation in Visual Basic.

1. Use the Visual Basic form created in the previous section and save it using a new name.

2. Put the following code on the CreateDatabase button of form1.

```
Dim session As Object
Dim db1 As Object
Dim db2 As Object
Set session = CreateObject("Notes.NotesSession")
Set db1 = session.GetDatabase("", "test.nsf")
Set db2 = db1.CreateFromTemplate("", "test2.nsf",True)
Print ("test2.nsf is created on local server")
Set db2 = Nothing
Set db1 = Nothing
Set session = Nothing
```

Following is an explanation of the code.

When using Notes classes in Visual Basic through OLE automation, there are two externally creatable objects available: NotesSession and NotesUIWorkspace. You can get references to other objects from these two objects.

We define db1 and db2 as objects that reference NotesSession and NotesDatabase. When using Notes classes in Visual Basic, you cannot directly create a database from a template as you cannot use the New method of the NotesDatabase class. Therefore, you need to get an inheritance from another database, db1, based on the DISCUSS4.NTF template.

```
Dim session As Object
Dim db1 As Object
Dim db2 As Object
```

We get a reference to NotesSession through CreateObject and store it in session. We get a reference to NotesDatabase using the GetDatabase method of NotesSession.

```
Set session = CreateObject("Notes.NotesSession")
Set db1 = session.GetDatabase("", "test.nsf")
```

For the HTDatabaseCreate function of the HiTest Visual Basic API, there is an equivalent function available in LotusScript. It is the CreateFromTemplate method of the NotesDatabase class. We create a new database with a file name of test2.nsf on the local server using this method. The new database inherits the TEST database.

```
Set db2 = db1.CreateFromTemplate("", "test2.nsf",True)
```

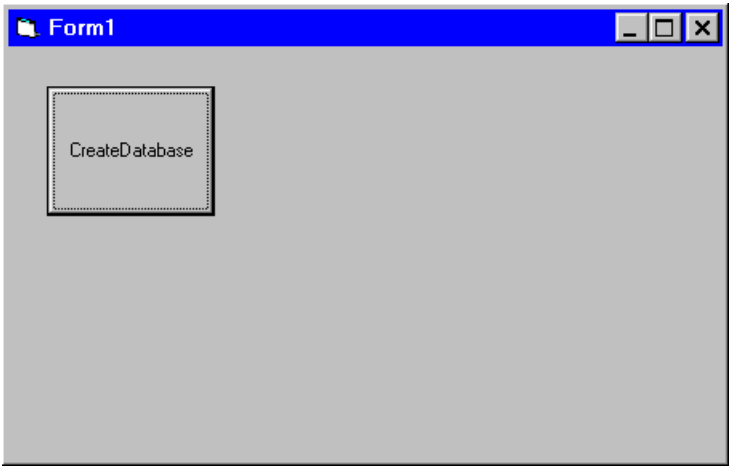
We inform the user that the database is created.

```
Print ("test2.nsf is created on local server")
```

We release the Notes objects.

```
Set db2 = Nothing
Set db1 = Nothing
Set session = Nothing
```

3. Save the new project.
4. Choose Run-Start from the Visual Basic menu. Form1 is displayed. It looks like this:



5. Click the CreateDocument button to run the application. A message displays which informs you that test2.nsf has been created on the local server.
6. Switch to Lotus Notes.
7. Select File - Database - Open and add the new database to your workspace.

Converting a HiTest Application to LotusScript in Notes

LotusScript is part of Lotus Notes Release 4. When using LotusScript in Notes, you do not need an additional compiler for the language. Lotus Notes compiles and runs the code. Converting your application to LotusScript in LotusNotes enables you to distribute your applications across a wider range of platforms, such as Windows, OS/2 and UNIX.

In HiTest Tools for Visual Basic, you first initialize a session object and use the hierarchy of the session object to access the lower level objects.

Using LotusScript Notes classes through OLE automation in Visual Basic you first create one of the two externally creatable objects: NotesUIWorkspace or NotesSession. You access the lower-level objects using the methods and properties of the higher-level objects.

In Lotus Notes, when using LotusScript, you can create most of the Notes objects without using the properties or methods of the higher-level classes. This is an example of how to create a new Notes database:

```
Dim db As New NotesDatabase("","discuss4.ntf")
```

We want to create a new Notes database directory on a server called Mail:

```
Dim dbdir As New NotesDbDirectory("Mail")
```

LotusScript also supports the Forall and Foreach statements which are not supported in Visual Basic using Notes classes through OLE automation.

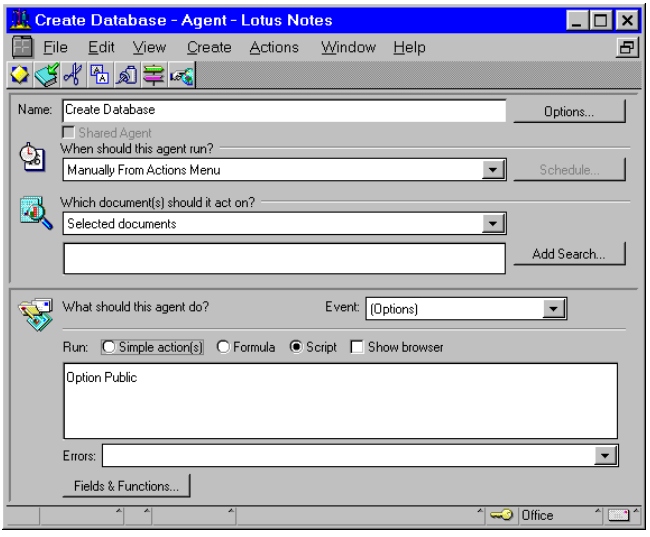
For example, you can access all the views in a database like this:

```
Forall view In db.Views
     Messagebox "View name: " & view.Name
End Forall
```

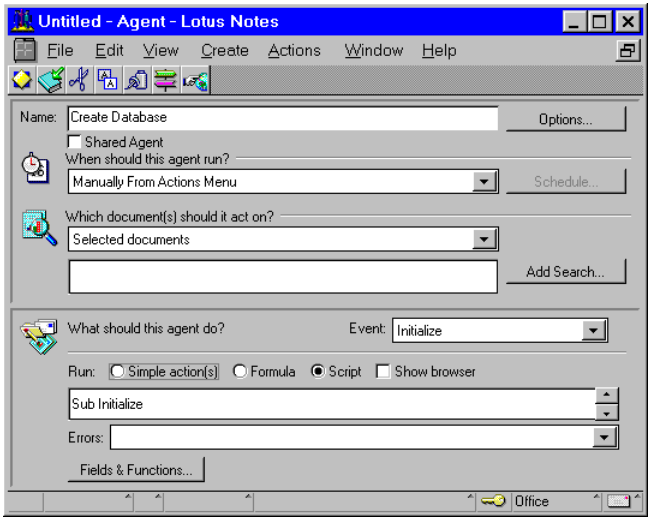
Let's look at how to convert the sample from "Converting a HiTest Application Using Notes Classes in Visual Basic through OLE Automation" into a LotusScript application which is used directly in LotusNotes.

1. Open the TEST database in Lotus Notes.
2. Choose Create - Agent from the menu to create a new agent.
3. Enter the title of the agent in the Name field.

4. Under When should this agent run?, select Manually From Actions Menu.



5. In the Event: box, choose Initialize as the event to drive the script.



6. Enter the following code on the edit pane:

```
Sub Initialize
     Dim session As New NotesSession
     Dim db1 As NotesDatabase
     Dim db2 As NotesDatabase
     Set db1 = session.GetDatabase("", "test.nsf")
     Set db2 = db1.CreateFromTemplate("",
"test3.nsf",True)
     Messagebox ("test3.nsf is created on local server")
End Sub
```

Following is an explanation of the code.

This code is quite similar to the one we used in Visual Basic using OLE automation. However, rather than declaring the Notes objects as objects, we declare them as LotusScript Notes classes. Also, we declare the session as NotesSession.

```
Dim session As New NotesSession
```

We declare db1 as NotesDatabase which references the TEST database.

```
Dim db1 As NotesDatabase
```

We declare db2 as NotesDatabase which will be the new database and is inherited from the TEST database.

```
Dim db2 As NotesDatabase
```

We use the methods of LotusScript Notes classes to create the new database, just as we did in Visual Basic using OLE automation.

```
Set db1 = session.GetDatabase("", "test.nsf")
```

```
Set db2 = db1.CreateFromTemplate("", "test3.nsf",True)
```

We inform the user that database test3.nsf is created.

```
Messagebox ("test3.nsf is created on local server")
```

As we can use the New method of NotesDatabase to create a database from a template, we can create this much simpler program which performs the same function:

```
Sub Initialize
     Dim db As New NotesDatabase("","DISCUSS4.NTF")
     Call db.CreateFromTemplate("", "test3.nsf" ,True)
     Messagebox db.Title
End Sub
```

Following is an explanation of the code:

We declare db as a new database that inherits the DISCUSS4.NTF template.

```
Dim db As New NotesDatabase("","DISCUSS4.NTF")
```

We call the CreateFromTemplate method of NotesDatabase to create and initialize the database.

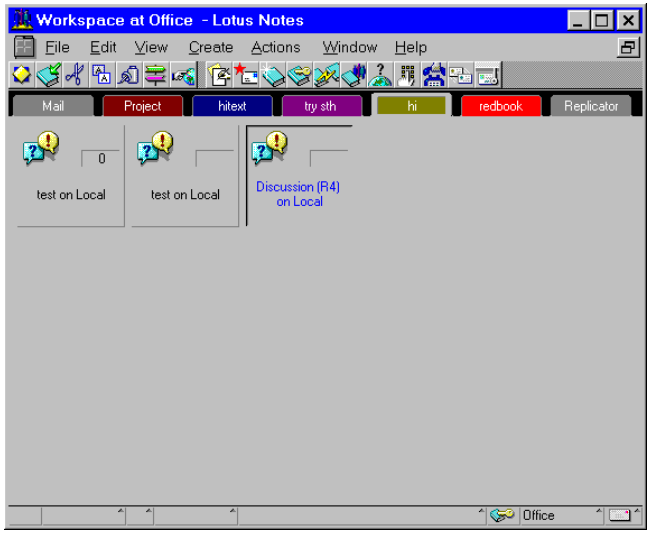
```
Call db.CreateFromTemplate("", "test3.nsf" ,True)
```

We put the title of the database on the message box.

```
Messagebox db.Title
```

7. Save and close the agent by choosing File - Close from the Notes menu.
8. On the Notes workspace, select the TEST database.
9. Choose Agent - Create Database from the Notes menu to run the agent. A message displays informing you that the database is created.

10. From the Notes menu, choose File - Database - Open to add the new database to the workspace.



The following chapters discuss how to access the following major Notes elements: NotesSession, Notes Database, Notes Document, Notes View, and Notes Item. We will look at how to access those elements using the HiTest Basic API, using Notes classes in Visual Basic through OLE automation, and using LotusScript in Notes.

Chapter 6
Accessing Notes Sessions

Notes sessions provide access to environment variables and address books, and they provide information on the current user, Notes platform and Notes release number.

In this chapter, we will discuss how to access Notes sessions:

- In HiTest
- Using Notes classes through OLE automation in Visual Basic
- Using LotusScript in Notes.

Accessing Notes Sessions Using the HiTest Basic API

Session class functions are global within a HiTest API session (one processor task). These functions have no context beneath the implicit session context — that is, no functions in other classes affect their operation — and they are always usable, after calling HTInit.

Every HiTest API program must initialize and terminate the HiTest API. Unless you call the HiTest API initialization function HTInit to establish a session, all other functions will fail. Additionally, every HiTest API program must call the HTTerm function after all HiTest API function calls are complete and before the program terminates. It is crucial to call the termination function to avoid leaving the system in a dangerous state.

The single exception to these restrictions is the HTGetAPILibraryVersion function, which may be used before initialization to check the HiTest API DLL version code and string. The version of the HiTest API with which a program was created can be obtained with the constants HTVERSION_LIBRARY for the version code and HTVERSION_LIBRARY_STRING for the version string.

For detailed information on the Notes session functions, refer to the online help information for the HiTest Tools Visual Basic API.

Let’s look at some of the functions.

Accessing Notes Session Properties

HTGetProperty fetches one of various session-level property values into a supplied buffer. Each property has a data type, and the buffer must be large enough to hold the result.

HTSetProperty assigns a value to a session-level property. The type of the data is indicated by the property. When setting the value of properties which are inherited by newly opened databases, the new value has no effect on currently open databases.

Let's look at an example.

Example

This example retrieves and prints a session property for the current user's name.

```
Dim hgstatus As Long
Dim hgvalue As String

hgvalue = String(HTMaxLen_User_Name, 0)
hgstatus = HTInit(0)
hgstatus = HTGetProperty(HTSession_User_Name, ByVal hgvalue)
hgvalue = Left(hgvalue, InStr(hgvalue, Chr$(0)) - 1)
If hgstatus = 0 Then Print "The current user is " & hgvalue & "."
hgstatus = HTTerm(0)
```

Following is an explanation of the code.

We declare an hgstatus variable. All the status codes we get later will be put in the variable.

```
Dim hgstatus As Long
```

We declare hgvalue as a string which will store the user name of the Notes session.

```
Dim hgvalue As String
```

For every HiTest application, we must initialize the Notes session as a first step. Other functions can only be called after session initialization. If the initialize is unsuccessful, it will return either HTFAIL_HITEST_VERSION (program built with incompatible HiTest API version) or HTFAIL_NOTES_VERSION (the HiTest API requires Notes Release 3.0 or higher) to hgstatus.

```
hgstatus = HTInit(0)
```

We set the string length as the maximum length of the user name in Notes. HTMaxLen_User_Name is a constant defined by HiTest to present the maximum length of the Notes user name.

```
hgvalue = String(HTMaxLen_User_Name, 0)
```

We get the user name and store it in hgvalue.

```
hgstatus = HTGetProperty(HTSession_User_Name, ByVal hgvalue)
```

We set the string to only contain valid characters.

```
hgvalue = Left(hgvalue, InStr(hgvalue, Chr$(0)) - 1)
```

We print out the user name.

```
If hgstatus = 0 Then Print "The current user is " & hgvalue &
"."
```

We close the Notes session. Each HTInit function must be closed. If there is a failure, the same failure code will be returned as for HTInit.

```
hgstatus = HTTerm(0)
```

Accessing Notes Session Environment Values

HTGetEnvironmentString retrieves the value of a Notes environment string variable. Notes stores environment variables in the NOTES.INI file. Use the HTSetEnvironmentString function to modify Notes environment variables. To retrieve import/export formats, which are stored in multiple environment variables, use the functions HTCompositeListImport and HTCompositeListExport.

HTSetEnvironmentString assigns a value to a Notes environment string variable. The calling program controls whether to create a new environment variable. Notes stores environment variables in the NOTES.INI file. Use the HTGetEnvironmentString function to retrieve Notes environment variables. Use care when modifying Notes environment variables, since they affect the Notes client and server programs.

Let's look at an example.

Example

This example sets the value of a Notes environment variable in the NOTES.INI. If the variable does not already exist, it is created.

```
Dim hgstatus As Long


hgstatus = HTInit(0)
hgstatus = HTSetEnvironmentString("TestOfAPI", "It worked",
HTSetEnvF_Create)
If hgstatus = 0 Then Print "This environment variable has
been set in your notes.ini file."
hgstatus = HTTerm(0)
```

Following is an explanation of the code.

We set the environment TestOfAPI to “It worked”. If the TestOfAPI does not already exist in NOTES.INI, we create it. If it fails, an error code is returned: “Value is longer than the maximum environment string length”.

```
hgstatus = HTSetEnvironmentString("TestOfAPI", "It worked",
HTSetEnvF_Create)
```

If the setting is successful, we print out a message to inform the user.

```
If hgstatus = 0 Then Print "This environment variable has
been set in your notes.ini file."
```

Accessing Notes Sessions in Visual Basic Using Notes Classes through OLE Automation

Using Notes classes through OLE Automation in Visual Basic, we present the session as NotesSession. The NotesSession class provides a means for accessing attributes of the user environment and information about agents. It also provides methods for accessing environment variables.

Following is a table which converts the HiTest Basic API functions accessing Notes sessions to the equivalent functions and properties using Notes classes through OLE Automation:

HiTest Functions	*Notes Class methods and Properties*	*Summary*
HTConvert		Converts data between data types.
HTConvertGetLength		Returns the length of data converted as indicated.
HTGetAPILibraryVersion		Obtains the HiTest API DLL version information.
HTGetEnvironmentString	GetEnvironmentString	Obtains the value of a Notes environment string variable.
HTGetFormat		Obtains the default formatting configuration.
HTGetInternational	International	Obtains the international configuration information.
HTGetProperty	UserName, CommonUserName, IsOnServer, NotesVersion, (only in Release 4.5: International, GetAgent)	Obtains a session-level property value.
HTInit	CreateObject	Initializes the HiTest API.
HTSetEnvironmentString	SetEnvironmentVar	Assigns the value of a Notes environment string variable.
HTSetFormat		Assigns the default formatting configuration.
HTSetInternational		Assigns the international configuration information.
HTSetProperty		Assigns the value of a session-level property.
HTStringFetch		Fetches string from within an HT structure.
HTTerm	Close	
HTTranslate		Translates a string between the native character set and LMBCS.
	Platform	Name of the OS Platform
	CurrentDatabase	Object Reference to the database in which the Agent lives
	CurrentAgent	Object reference to the current agent program
	EffectiveUserName	Name of the person who created the agent
	SavedData	Handle to document that stores persistent agent data
	AddressBooks	Returns array containing database objects for known address books.
	UpdateProcessedDoc	Records document processed by agent.

Continued

HiTest Functions	*Notes Class methods and Properties*	*Summary*
	GetDatabase	Equivalent to New Notes Database (see HiTest Database Object)
	CreateDateTime	Equivalent to New NotesDateTime
	CreateLog	Create Log (Equivalent to New NotesLog)
	CreateNewsletter	Equivalent to New NotesNewsletter
	GetDbDirectory	Equivalent to New NotesDbDirectory
	GetEnvironmentValue	Retrieves a numeric environment value.

Note The International method will be available in Lotus Notes Release 4.5.

Accessing Notes Session Properties

Using the object.property syntax, you can access all the session properties.

Let's look at an example:

Example

```
Dim session As Object
Set session = CreateObject ("Notes.NotesSession")
Print ( session.UserName )
Set session = Nothing
```

Following is an explanation of the code.

We declare session as an object which will reference the NotesSession class.

```
Dim session As Object
```

We get a reference to NotesSession using CreateObject.

```
Set session = CreateObject ("Notes.NotesSession")
```

We display the current user name.

```
Print ( session.UserName )
```

We release the NotesSession object.

```
Set session = Nothing
```

Accessing Notes Session Environment Values

Using GetEnvironmentString, SetEnvironmentVar and GetEnvironmentValue, we can access Notes session environment values.

The GetEnvironmentValue, GetEnvironmentString, and SetEnvironmentVar methods retrieve and set environment variables, which are stored in the local NOTES.INI or Preferences file. Use GetEnvironmentValue only for numeric environment variables. Use GetEnvironmentString for strings and numeric values.

SetEnvironmentVar prepends a dollar sign ($) to argument 1, the name of the environment variable. This forces a distinction between user environment variables (name starts with a dollar sign) and system environment variables (name does not start with a dollar sign). GetEnvironmentValue and GetEnvironmentString prepend a dollar sign if the second argument is false or omitted, and do not prepend a dollar sign if the second argument is true. This permits you to examine system environment variables as well as user ones.

Environment variables are useful for saving state and data between Notes sessions on a single server or workstations where no conflicts are possible. They are also useful for obtaining the environment information set by Notes, such as KitType, Directory, Preferences, Domain, Port, and so on.

Let's look at an example.

Example

The following example shows how to set a Notes environment value in NOTES.INI.

```
Dim session As Object

Set session = CreateObject("notes.notessession")

Call session.SetEnvironmentVar("TestOfAPI", "IT WORKS")

Set session = Nothing
```

Following is an explanation of the code.

We use the SetEnvironmentVar method of SessionObject to set the environment variable.

```
Call session.SetEnvironmentVar("TestOfAPI", "IT WORKS")
```

Accessing Notes Sessions in LotusScript

When we use LotusScript in Notes, we have only one Notes session; so successive calls to the New method return the same session handle. You do not need the session handle to access databases or perform other operations in LotusScript. It is only used to access the session properties, methods and Notes session environment values.

You can close a session at any time with the Close method. The Close method is implicit upon program termination.

For detailed information on NotesSession, refer to Chapter 3.

Example

This button example increments the environment variable $SeqNo by one and prints the result. (This script would not be reliable on a server for maintaining a system of sequential numbers.)

```
Sub Click(Source As Button)
     Dim session As New NotesSession
     Dim seqNo As integer
     seqNo = session.GetEnvironmentValue("SeqNo")
     If Isempty(seqNo) Then
          Call session.SetEnvironmentVar("SeqNo", 1)
     Else
          seqNo = seqNo + 1
          Call session.SetEnvironmentVar _
          ("SeqNo", Cint(seqNo))
     End If
     Messagebox session.GetEnvironmentValue("SeqNo")
     Call session.Close
End Sub
```

Following is an explanation of the code:

We declare session as a new Notes session object.

```
Dim session As New NotesSession
```

We declare seqNo as an integer which references the Environment value.

```
Dim seqNo As integer
```

We get the Environment value in NOTES.INI and store it in seqNo.

```
seqNo = session.GetEnvironmentValue("SeqNo")
```

If there is no seqNo in NOTES.INI, we create it and set it to 1, otherwise we increase the Environment value by one.

```
If Isempty(seqNo) Then
          Call session.SetEnvironmentVar("SeqNo", 1)
     Else
          seqNo = seqNo + 1
          Call session.SetEnvironmentVar _
          ("SeqNo", Cint(seqNo))
End If
```

We print out the Environment value.

```
Messagebox session.GetEnvironmentValue("SeqNo")
```

We close the Notes session using the Close method.

```
Call session.Close
```

Chapter 7
Accessing Notes Databases

A database is a file used by Notes to store data in the form of documents. A database also contains metadata in the form of forms, views, agents, and an Access Control List (ACL).

In this chapter, we will discuss how to access a Notes database

- In HiTest
- Using Notes classes through OLE automation in Visual Basic
- Using LotusScript in Notes.

Accessing Notes Databases Using the HiTest Basic API

In the HiTest Visual Basic API, the database object exists in the implicit context of the session and represents Lotus Notes databases as distinct objects. The primary attributes of a database are the server name, database name, database file path (relative to the Notes data directory), and a handle when the database is opened.

A database may be opened, in which case it is represented by a handle. When opened, the contents of a database are accessed and manipulated with this handle. An open database also contains status information pertaining to the database file. Many of the remaining objects exist within the context of a database. Some objects use a database directly and some indirectly through a document, which is only valid in the context of that database, including indices, open documents, metadata, and properties. Multiple databases may be opened in a single session. The HiTest API uses the global constant NULLHANDLE = 0 to represent an invalid database handle.

Accessing Notes Database Properties

`HTDatabaseGetProperty`: Fetches one of various database-level property values into a supplied buffer. Each property has a data type, and the buffer must be large enough to hold the result.

`HTDatabaseSetProperty`: Assigns a value to a database-level property. The type of the data is indicated by the property. When setting the value of properties, which are inherited by open indices, the new value has no effect on currently open indices.

Example

The following example sets the title of the database.

```
Dim hgdatabase As Long

Dim hgstatus As Long


hgstatus = HTInit(0)

hgstatus = HTDatabaseOpen("", "test.nsf", 0&, hgdatabase)

hgstatus = HTDatabaseSetProperty(hgdatabase,
HTDATABASE_TITLE, ByVal "New Title")

hgstatus = HTDatabaseClose(hgdatabase, HTDBCLOSEF_SAVE_DOCS)

hgstatus = HTTerm(0)
```

Following is an explanation of the code:

We declare an hgdatabase variable that references a Notes database. In HiTest, we present a Notes database handle as long.

```
Dim hgdatabase As Long
```

We declare an hgstatus variable. All the status codes we get later will be put in the variable.

```
Dim hgstatus As Long
```

For every HiTest application, we must initialize the Notes session as a first step. Other functions can only be called after session initialization. If the initialize is unsuccessful, it will return either HTFAIL_HITEST_VERSION (program built with incompatible HiTest API version) or HTFAIL_NOTES_VERSION (the HiTest API requires Notes Release 3.0 or higher) to hgstatus.

```
hgstatus = HTInit(0)
```

We open the TEST database, which is already on the local server. This returns a database handle, a long value to the hgdatabase variable. 0& refers to the dbopen flag. If the database cannot open, a failure notification (database does not exist) will be returned to hgstatus.

```
hgstatus = HTDatabaseOpen("", "test.nsf", 0&, hgdatabase)
```

We set the property of hgdatabase. It sets the database title property as "New Title". HTDATABASE_TITLE specifies the database property to be set. If it fails, the function will return a failure code to hgstatus.

```
hgstatus = HTDatabaseSetProperty(hgdatabase,
HTDATABASE_TITLE, ByVal "New Title")
```

We open the database specified by hgdatabase. The flag HTDBCLOSEF_SAVE_DOCS is a constant defined by the HiTest API. It means that the document has changed but is still open and will be closed when the database is closed. If it cannot complete successfully, the function will return a failure code (unvalid database) to hgstatus.

```
hgstatus = HTDatabaseClose(hgdatabase, HTDBCLOSEF_SAVE_DOCS)
```

We close the Notes session. Each HTInit function must be closed. If there is a failure, the same failure code will be returned as for HTInit.

```
hgstatus = HTTerm(0)
```

Opening and Closing a Database

The HTDatabaseOpen function opens a Notes database and returns the handle. The handle represents an active connection to the indicated database file. Multiple databases may be open at one time, including multiple connections to a single database. Access to the database's data and metadata can only occur in the context of an open database.

For more information, refer to the example on how to access Notes database properties further on in this chapter.

Creating a Notes Database

The HTDatabaseCreate function creates a new database. The database may be created from scratch or from an existing database template. The local Notes user will be added to the new database Access Control List (ACL) as the manager.

For more information, refer to the example in Chapter 5 on how to create a new database called TEST in HiTest.

Accessing Notes Databases in Visual Basic Using Notes Classes through OLE Automation

Lotus Notes provides the NotesDatabase and NotesDbDirectory classes to access Notes databases. The NotesDatabase and NotesDbDirectory classes provide a means for locating and opening Notes databases. NotesDatabase provides access to the NotesView, DocumentCollection, and NotesDocument objects. NotesDocument can be accessed directly through NotesDatabase, or by first obtaining a NotesDocumentCollection or NotesView object.

To convert the HiTest Visual Basic API applications, we translate the functions to Notes class methods and properties. Following is a table which converts the HiTest Basic API functions accessing the Database class to the equivalent functions and properties in the Notes classes:

HiTest Name	*LotusScript Notes Class*	*Notes Class Methods or Properties*	*Summary*
HTDatabaseClose	NotesDatabase	Close	Closes an open database.
HTDatabaseCompact	NotesDatabase	Compact	Compacts a database file.
HTDatabaseCopy	NotesDatabase	CreateCopy, CreateReplica	Copies part or all of one database to another database.
HTDatabaseCreate	NotesDatabase	New, Create, CreateFromTemplate	Creates a new database.
HTDatabaseCreateFT_Index	NotesDatabase	UpdateFTIndex	Full text indexes a local database.
HTDatabaseDelete	NotesDatabase	Remove	Deletes a database file.
HTDatabaseDeleteFT_Index			Deletes the full text index for a database.
HTDatabaseGetProperty	NotesDatabase	IsFTIndexed, Created, LastFTIndexed, TemplateName, DesignTemplateName, FileName, FilePath, LastModified, ReplicaID, Server, Size, PercentUsed, Title, Categories	Obtains the value of a database property.

Continued

HiTest Name	*LotusScript Notes Class*	*Notes Class Methods or Properties*	*Summary*
HTDatabaseList	DbDirectory	GetFirstDatabase, GetNextDatabase	Iterates through Notes databases and directories on a given server.
HTDatabaseListCatalog			Iterates through databases in a database catalog.
HTDatabaseLocateBy _Replicaid	NotesDatabase	OpenByReplicaID	Obtains a database filename from a database replica ID.
HTDatabaseLocateBy _Title	NotesSession	GetDatabase	Obtains a database filename from a database title.
HTDatabaseOpen	Session DBDirectory	Open, GetDatabase GetFirstDatabase GetNextDatabase GetLastDatabase GetPreDatabase	Opens a Notes database.
HTDatabaseSetProperty	NotesDatabase	Title, Categories, IsOpen, Managers, Views, Agents, Parent, ACL, FTSearch, Replicate	Assigns the value of a database property.

Note The `New` and `Open` methods can only be used in LotusScript.

Note For the GetFirstDatabase method, you must use the Type number in Visual Basic rather than the Type constant in LotusScript.

Type Constant	*Type Number*
DATABASE	1247
TEMPLATE	1248
REPLICA _CANDIDATE	1245
TEMPLATE_CANDIDATE	1246

Accessing Notes Database Properties

We access Notes Database properties in Visual Basic by using the NotesDatabase class properties. The database must be open when accessing the database properties. Let's look at an example.

Example

The following example shows you how to access the database properties.

```
Dim session As Object

Dim db As Object
```

```
Set session = CreateObject ("Notes.NotesSession")
Set db = session.GetDatabase ("","test.nsf")
db.Title = " New Title"
Set db = Nothing
Set session = Nothing
```

Following is an explanation of the code:

We declare the session as an object which references the Notes Session object. We declare db as an object which references the Notes Database object.

```
Dim session As Object
Dim db As Object
```

We get a reference to NotesSession.

```
Set session = CreateObject ("NotesSession")
```

We open the TEST database using the GetDatabase method of NotesSession.

```
Set db = session.GetDatabase ("","test.nsf")
```

We set the Title property of NotesDatabase to “New Title”.

```
db.Title = " New Title"
```

We release the Notes objects.

```
Set db = Nothing
Set session = Nothing
```

Opening and Closing a Database

To open a database we use the GetFirstDatabase or GetNextDatabase method of NotesDbDirectory.

We use the OpenByReplicaID method to open a database whose server and replica ID are known. We can obtain the replica ID from a database that is already open using the ReplicaID property.

The OpenIfModified method opens a database only if it was modified after a specified date.

The OpenMail method opens the user’s mail database.

The Close method explicitly closes a database. Exiting from Notes implicitly closes all open databases. We open a Notes database using the Open method of NotesDatabase. We close a Notes database using the Close method of NotesDatabase.

All of the methods mentioned above can be used in Visual Basic.

For the HiTest function, HTDatabaseOpen, the GetDatabase method of NotesSession is available, which opens a database specified by the server and the database file.

For the HiTest function, HTDatabaseClose, the Close method of NotesDatabase is available, which closes the particular database.

Let's look at an example.

Example

```
Dim session As Object

Dim db As Object

Set session = CreateObject ("NotesSession")

Set db = session.GetDatabase ("","test.nsf")

If db.Isopen

   print ( "Database TEST is open")

End IF

db.close

Print(" Database TEST is Closed .")

Set db = Nothing

Set session = Nothing
```

Following is an explanation of the code:

We open the test.nsf database on the local server.

```
Set db = session.GetDatabase ("","test.nsf")
```

We check if the database is open. If it is, we print the message "Database TEST is open".

```
If db.Isopen

   print ( "Database TEST is open")

End IF
```

We close the database using the Close method.

```
db.close
```

We print the message that the database is closed.

```
Print(" Database TEST is Closed .")
```

Creating a Notes Database

Create: Creates an uninitialized database. A script can access it, but a user cannot access it from the Notes workspace because it has no forms or views.

CreateFromTemplate: Creates an initialized database based on an existing template. This database has the design of the template and can be accessed from the Notes workspace.

CreateReplica: Creates an uninitialized replica of an existing database. The replica must be initialized either by the application developer using the Replicate method or by the user accessing it from the Notes workspace upon first use.

CreateCopy: Creates a copy of the design of an existing database.

For more details, refer to Chapter 5 which contains an example on how to convert an application using LotusScript Notes classes.

Accessing Notes Databases in LotusScript

The following describes how to access Notes databases using LotusScript in Notes.

Accessing Notes Database Properties

You can access Notes database properties of an open Notes database. Using LotusScript directly in Lotus Notes enables you to use the Forall and Foreach statements, which make it easier to access the properties. This is not possible in Visual Basic using Notes classes through OLE automation, as Visual Basic does not know the concept of a collection, whereas LotusScript does.

Let's look at an example.

Example

This example prints the name of each view in the database. A NotesView object is defined and set to each value in the Views property. The name is a property of the view object. You can put the code on an agent and choose initialize as the event to drive the agent.

```
Sub Initialize
   Dim session As New NotesSession
   Dim db As NotesDatabase
   Set db = session.CurrentDatabase
```

```
    REM view becomes a NotesView object
    Forall view In db.Views
      Messagebox "View name: " & view.Name
    End Forall
End Sub
```

Following is an explanation of the code.

We will process a view whose object class is NotesView. It descends from NotesDatabase, which in turn descends from NotesSession. So in this statement we declare a NotesSession with a property of CurrentDatabase that allows us to access the current database.

```
Dim session As New NotesSession
```

We declare db to reference the current database.

```
Dim db As NotesDatabase
```

We get the reference to the current database using the CurrentDatabase method of session, and store it in the db variable.

```
Set db = session.CurrentDatabase
```

We include a comment to state that the view is a NotesView object in our case. The LotusScript compiler will ignore this statement.

```
REM view becomes a NotesView object
```

We process all the views in the database and display all the views' names one by one in a message box.

```
Forall view In db.Views
  Messagebox "View name: " & view.Name
End Forall
```

Opening and Closing a Database

You can use the New method of NotesDatabase in LotusScript. The New method opens an existing database. The first argument specifies the server and defaults (empty string) to the local Notes directory. The second argument specifies the file name of the database. If you specify an empty string here, you must later use the Open method to open the database.

If you are running on a server and want to open a database on that server, specify the first argument as an empty string. The New method does not create a database.

Let's look at an example:

Example

```
Dim db As New NotesDatabase( "", "" )

Call db.Open( "HongKong", "sales.nsf" )
```

Following is an explanation of the code.

We create a NotesDatabase object which does not reference any object yet, and will open it later on.

```
Dim db As New NotesDatabase( "", "" )
```

We open the sales.nsf database on the HongKong server, and have db reference the sales.nsf database on the HongKong server.

```
Call db.Open( "HongKong", "sales.nsf" )
```

Creating a Notes Database

Use one of the following methods of NotesDatabase to create a database: Create, CreateFromTemplate, CreateReplica, CreateCopy.

Note The New method does not create a database.

Example

The following example creates a local database named saledisc.nsf based on the Notes template discuss4.ntf and changes the title. The database object for the template is discuss and the database object for the new database is db. The third argument, CreateFromTemplate, inherits design changes. Put the code on an agent and choose initialize to drive the script.

```
Sub INITIALIZE

   Dim discuss As New NotesDatabase("", "DISCUSS4.NTF")

   Dim db As NotesDatabase

   Set db = discuss.CreateFromTemplate_

  ("", "SALEDISC", True)

   db.Title = "Sales Discussion Database"

End Sub
```

Following is an explanation of the code.

We declare two databases: db and discuss.

```
Dim discuss As New NotesDatabase("", "DISCUSS4.NTF")

Dim db As NotesDatabase
```

We create a new database on a local server. The file name of the database is SALEDISC and it uses discuss as the database template.

```
Set db = discuss.CreateFromTemplate_
  ("", "SALEDISC", True)
```

We set the new database db's title property to "Sales Discussion Database".

```
db.Title = "Sales Discussion Database"
```

Note In Lotus Notes Release 4.5, a new front-end class will be available, NotesUIDatabase. This class can only be used in LotusScript. In an opened database, you can use it to access the current database using scripts.

Chapter 8
Accessing Notes Views

A view is one of two primary types of metadata. The other is a form. Each database contains one or more views, which describe one representation of some or all of the documents in a database. A view consists of various attributes and one or more columns. In addition, each view defines a collection. A views collection is a hierarchical, sorted collection of documents, categories, and totals.

In this chapter, we will discuss how to:

- Access Notes views in HiTest
- Access Notes views and folders using Notes classes through OLE automation in Visual Basic
- Access Notes views and folders using LotusScript in Notes.

Note Folders are new in Notes Release 4.

Accessing Notes Views Using the HiTest Basic API

Lotus Notes internally represents views as data. The HiTest API view abstraction represents views as metadata, and supports easy access to that metadata. The primary attributes of a view are a name and an ID. The HiTest API uses the global constant NULLID = 0 to represent an invalid view ID.

Accessing Notes View Attributes

HTViewFetch function: Fetches the attributes of a particular view. It returns an hgview type value which contains the attributes.

Let's look at an example.

Example

Locates a view, retrieves the attributes of that view, and prints the name attribute.

```
Dim hgstatus As Long

Dim hgdatabase As Long
```

```
Dim hgviewid As Long
Dim hgview As HTView
Dim temp As String

hgstatus = HTInit(0)
hgstatus = HTDatabaseOpen("", "htvb20.nsf", 0&, hgdatabase)
hgviewid = HTViewLocateByName(hgdatabase, "HiTest API - All
Documents")
hgstatus = HTViewFetch(hgdatabase, hgviewid, hgview)
If hgstatus = 0 Then
    temp = Left(hgview.hgname, InStr(hgview.hgname, Chr$(0))
- 1)
    Print "The view name is '" & temp & "'."
End If
hgstatus = HTDatabaseClose(hgdatabase, 0&)
hgstatus = HTTerm(0)
```

Following is an explanation of the code.

To locate the view, we need to know the view id of the particular view. In HiTest, the Notes view id is presented as long. This statement declares a view id.

```
Dim hgviewid As Long
```

This statement declares a variable which references the Notes view.

```
Dim hgview As HTView
```

We declare a string variable which will contain the view's name later.

```
Dim temp As String
```

We put the Notes view id in hgdatabase in hgviewid. The view is specified by the view name "HiTest API - ALL Documents". If no view exists, it will return NULLID.

```
hgviewid = HTViewLocateByName(hgdatabase, "HiTest API - All
Documents")
```

We fetch a Notes view in hgdatabase. The view is specified by hgviewid. If it fails, a failure code of invalid database or view is returned to hgstatus.

```
hgstatus = HTViewFetch(hgdatabase, hgviewid, hgview)
```

If there is no error code returned by HTViewFetch, we get and print the view's name.

```
If hgstatus = 0 Then

    temp = Left(hgview.hgname, InStr(hgview.hgname, Chr$(0))
- 1)

    Print "The view name is '" & temp & "'."

End If
```

Locating a View

Use the HTViewLocateByName function in HiTest to obtain the view ID of a view through the use of that view's name.

Refer to our previous example of fetching Notes view attributes.

Deleting a View

The HTViewDelete Function deletes a view from a database.

Note A view currently used as the active view in an index cannot be deleted.

Let's look at an example.

Example

This Visual Basic code deletes a view from a database. This is not a working example because a valid view needs to be supplied first.

```
Dim hgstatus As Long

Dim hgdatabase As Long

Dim hgviewid As Long


hgstatus = HTInit(0)

hgstatus = HTDatabaseOpen("", "test.nsf", 0&, hgdatabase)

hgviewid = HTViewLocateByName(hgdatabase, "HiTest API - All
Documents")

hgstatus = HTViewDelete(hgdatabase, hgviewid)

If hgstatus = 0 Then Print "The view has been deleted."

hgstatus = HTDatabaseClose(hgdatabase, 0&)

hgstatus = HTTerm(0)
```

Following is an explanation of the code:

We delete a view in the database specified by hgdatabase. The view is specified by hgviewid. If it cannot run successfully, an error code is returned.

```
hgstatus = HTViewDelete(hgdatabase, hgviewid)
```

If the view is deleted successfully, a message displays saying that the view has been deleted.

```
If hgstatus = 0 Then Print "The view has been deleted."
```

Other Functions in HiTest for Working with Views

HTViewCopy: Copies a view from one database to another.

HTViewList: Iterates through views in a database.

HTViewcellFetch: Converts and retrieves the data for a viewcell into a supplied buffer.

HTViewcellGetLength: Obtains the length of a viewcell as converted to a specified data type.

Accessing Notes Views and Folders in Visual Basic Using Notes Classes through OLE Automation

We translate the HiTest view functions to equivalent methods and properties using Notes classes. The NotesView class lets you locate documents within views and folders, and perform operations on views and folders.

Following is a table which converts the HiTest Basic API functions to the equivalent functions and properties using Notes classes:

HiTest Function Name	*Notes Classes*	*Notes Classes Methods and Properties*	*Summary*
HTViewCopy			Copies a view from one database to another.
HTViewDelete	NotesView	Remove	Deletes a view from a database.
HTViewList			Iterates through views in a database.
HTViewLocateByName	NotesDatabase	GetView, *New*	Obtains a view ID from the view name.

Continued

HiTest Function Name	*Notes Classes*	*Notes Classes Methods and Properties*	*Summary*
HTViewFetch	NotesView	Created	The date the view was created.
	NotesView	LastModified	The date the view was last modified.
	NotesView	UniversalID	The UNID of the View.
	NotesView	Parent	A reference to the database object which owns this view.
	NotesView	IsDefaultView	True if the view is the default view for the database.
	NotesView	IsFolder	True if the object represents a folder.
HTViewcellFetch	NotesView	Columns	Converts and retrieves the data for a viewcell into a supplied buffer.
HTViewcellGetLength			Obtains the length of a viewcell as converted to a specified data type.

Note The `New` method is a new feature in Notes Release 4.5 and can only be used in LotusScript.

Note Folders are new in Notes Release 4. They are not supported by the HiTest Visual Basic API. When you use Notes classes in Visual Basic through OLE automation and in LotusScript in Notes Release 4, you can access folders in your application.

Accessing Notes View Properties

You can access Notes view properties using the NotesView class properties.

Let's look at an example.

Example

The following example is fetching the view's properties:

```
Dim session As Object
Dim db As Object
Dim view As Object

Set session = CreateObject("notes.notessession")
Set db = session.getdatabase("", "test.nsf")
Set view = db.GetView("By Author")
Print (view.Name)
```

Following is an explanation of the code.

We declare a session, db and view which reference the NotesView object.

```
Dim session As Object
Dim db As Object
Dim view As Object
```

We get the reference to NotesView using the GetView method of NotesDatabase.

```
Set view = db.GetView("By Author")
```

We print out the Name of the view .

```
Print (view.Name)
```

Locating a View

If the name of the view or folder is known, specify it to the GetView method of NotesDatabase. The view or folder name must be specified exactly, including cascades; for example, Certificates and ID Information\Cross Certificates. If the name has a synonym, you can specify the name or the synonym. For example, if a view name is By Author and the synonym is AuthorView, you can specify it as

"By Author"

or

"AuthorView"

However, you cannot specify both the name and the alias using the OR operator. For example, By Author | AuthorView is not possible.

If the name of the view or folder is not known, you can compare each view and folder in the Views property of NotesDatabase with one of the NotesView properties; for example, IsDefaultView or UniversalID.

For more details, refer to the previous example.

Deleting a View

If you update a database by adding, deleting, or changing information, the updates are not reflected in a view or folder until they are refreshed. The user can refresh a view or folder by clicking the refresh button or choosing View - Refresh.

If you want updates to automatically appear in a view or folder, call the Refresh method of NotesView after the update is complete. Document updates must first be posted to storage using the Save method of NotesDocument.

If you want to remove a view or a folder, call the Remove method.

Let's look at an example.

Example

The following example shows how to rewrite the equivalent HiTest sample using Notes classes through OLE automation. This example also removes the $ALL view.

```
Dim session As Object
Dim db As Object
Dim view As Object
Set session = CreateObject("Notes.NotesSession")
Set db = session.GetDatabase("","test.nsf")
Set view = db.GetView("$ALL")
Call view.Remove
Set view = Nothing
Set db = Nothing
Set session = Nothing
```

Following is an explanation of the code.

We declare view as an object which references NotesView.

```
Dim view As Object
```

We get the reference to NotesView using the GetView method of NotesDatabase, and store it in view.

```
Set view = db.GetView("$ALL")
```

We delete the $ALL view using the Remove method of NotesView.

```
Call view.Remove
```

We release the NotesView object.

```
Set view = Nothing
```

Accessing Notes Folders

Folders are a new feature in Notes Release. 4. A folder is a special kind of view. Folders let you store and manage related documents without putting them into a category, which requires a Categories field in the form used to create the documents. Folders are also convenient because you can drag documents to them.

The HiTest Visual Basic API does not support folders. If your application needs to access folders, you must use Notes classes through OLE Automation.

Using Notes classes through OLE Automation in Visual Basic, you can access Notes folders the same way you access Notes views. All the methods that work with Notes views also work with Notes folders.

Accessing Notes Views and Folders in LotusScript

Accessing Notes views and folders using LotusScript in Notes works the same way as accessing views and folders in Visual Basic using Notes classes through OLE Automation. However, you declare the variables referring to the view or folder as a Notes class rather than as an object.

In this part of the chapter, we will discuss the New method of NotesView, which is new in Notes Release 4.5.

When you access Notes views and folders in LotusScript, you can use the Forall and Foreach statements, which make your application more simple. Let's look at an example.

Example

This example accesses the view's properties. You can put this sample code on a new agent in the TEST database. Then choose the initialize function to drive the script.

```
Sub Initialized
 Dim session As New NotesSession
 Dim db As NotesDatabase
 Set db = session.CurrentDatabase
 REM view is a NotesView object
 REM cannot be Dim'd - loop reference variable
 Forall view In db.Views
          Messagebox "View name: " & view.Name & Chr(10) _
          & "Parent: " & view.Parent.Title & Chr(10) _
          & "Last modified: " & view.LastModified & Chr(10) _
          & "Created: " & view.Created & Chr(10) _
          & "Universal ID: " & view.UniversalID
          If view.IsDefaultView Then _
```

```
        Messagebox "Default view"
        If view.IsFolder Then Messagebox "Folder"
    End Forall
End Sub
```

Following is an explanation of the code:

We process all the views in the db database. The view's properties, such as parent, LastModified, Created and UniversalID are processed using the syntax object.property. IsFolder is a property of the view, which is defined as a folder.

```
Forall view In db.Views
        Messagebox "View name: " & view.Name & Chr(10) _
        & "Parent: " & view.Parent.Title & Chr(10) _
        & "Last modified: " & view.LastModified & Chr(10) _
        & "Created: " & view.Created & Chr(10) _
        & "Universal ID: " & view.UniversalID
        If view.IsDefaultView Then _
        Messagebox "Default view"
        If view.IsFolder Then Messagebox "Folder"
    End Forall
```

Note In Lotus Notes Release 4.5, a new front-end class will be available, NotesUIView. This class can only be used in LotusScript. In an opened view, you can use this class to access the current view.

Chapter 9
Accessing Notes Documents

Documents are the primary components of databases. A Notes document is a database entry that contains information. A document may range in size from a brief reply to a question to a multi-page market analysis filled with text and graphics. Every document has certain IDs which provide varying levels of uniqueness.

In this chapter, we will discuss how to access documents:

- In HiTest
- Using Notes classes through OLE automation in Visual Basic
- Using LotusScript in Notes.

Accessing Notes Documents Using the HiTest Basic API

In HiTest, every document has certain IDs which provide varying levels of uniqueness. The most commonly used ID is the document ID which is unique within the document's database (type Long). Use this ID to reference individual documents. Sets of documents may be obtained from an index, which is produced by executing a formula.

The primary attributes of a document are its HTDOCID, HTUNID, and HTOID values. To represent a document uniquely between databases, use either the universal ID (HTUNID) or the originator ID (HTOID). The universal ID is unique between databases but is the same for all replicas of the same document. The originator ID is unique among all other documents, including replicas (the universal ID is part of the originator ID).

To access or manipulate the contents of a document, the document must be opened. The HiTest API represents an open document by a document handle. A document handle is valid until closing either the document itself or the database containing the open document. The HiTest API uses the constant NULLHANDLE to represent an invalid document handle. Document decryption is handled automatically by the HiTest API when encrypted information is requested.

Accessing the Document Properties

HTDocumentGetProperty function: Fetches one of various document-level property values into a supplied buffer. Each property has a data type, and the buffer must be large enough to hold the result.

HTDocumentSetProperty function: Assigns a value to a document-level property. The type of the data is indicated by the property.

Example

This example sets the strict binding of all documents created after hgbegin_datetime to off.

```
Dim hgdatabase As Long
Dim hgdocid As Long
Dim hgindex As Long
Dim hgbegin_datetime As HTDATETIME
Dim hgindent As Long
Dim hgdocument As Long
Dim hgstatus As Long

hgstatus = HTInit(0)
hgstatus = HTDatabaseOpen("", "test.nsf", 0, hgdatabase)
hgstatus = HTIndexOpen(hgdatabase, hgindex)
hgstatus = HTIndexSearch(hgindex, "Select @All", 0&,
hgbegin_datetime)
hgstatus = HTIndexNavigate(hgindex, HTNAV_END, 0&, hgdocid,
hgindent)
Do Until hgstatus = HTFAIL_END_OF_DATA
hgstatus = HTDocumentOpen(hgdatabase, hgdocid, 0&,
hgdocument)
hgstatus = HTDocumentSetProperty(hgdocument,
HTDOCUMENT_STRICT_BIND, False)
hgstatus = HTDocumentClose(hgdocument, 0&)
hgstatus = HTIndexNavigate(hgindex,HTNAV_NEXT, 0&, hgdocid,
hgindent)
loop
hgstatus = HTDatabaseClose(hgdatabase, 0&)
hgstatus = HTTerm(0)
```

Following is an explanation of the code.

In HiTest, we present the index as long. We declare an hgindex variable which references the Notes index.

```
Dim hgindex As Long
```

We declare hgbegin_datetime as a Notes datetime item.

```
Dim hgbegin_datetime As HTDATETIME
```

We declare hgindent which references a buffer to receive the viewcell indentation of the new entry. In HiTest, we present it as long.

```
Dim hgindent As Long
```

This statement opens an index in the database specified by hgdatabase and returns the index value to the hgindex variable. If the open is unsuccessful, it will return an error code to hgstatus.

```
hgstatus = HTIndexOpen(hgdatabase, hgindex)
```

We produce hgindex, and have it contain only the documents created after hgbegin_datetime. If the function is unsuccessful, it will return an error code to hgstatus.

```
hgstatus = HTIndexSearch(hgindex, "Select @All", 0&,
hgbegin_datetime)
```

We navigate through the index specified by hgindex using a navigation style (direction and flags). HTNAV_END presents the navigation direction. The document id is returned to hgdocid and viewcell is returned to hgindent.

```
hgstatus = HTIndexNavigate(hgindex, HTNAV_END, 0&, hgdocid,
hgindent)
```

We want to process all the documents in the index. Therefore, we will use a Do_Until-loop statement.

```
Do Until hgstatus = HTFAIL_END_OF_DATA
```

```
Loop
```

We open the document specified by hgdocid in hgdatabase and return the handle of the document to hgdocument. If the open is unsuccessful, it will return an error code to hgstatus.

```
hgstatus = HTDocumentOpen(hgdatabase, hgdocid, 0&,
hgdocument)
```

We set the strict binding of the document to off. The document is specified by hgdocment. In case of failure, an error code is returned to hgstatus.

```
hgstatus = HTDocumentSetProperty(hgdocument,
HTDOCUMENT_STRICT_BIND, False)
```

Then we close each document. If the document cannot be closed successfully, an error code is returned to hgstatus.

```
hgstatus = HTDocumentClose(hgdocument, 0&)
```

Creating a Document

FTDocumentCreate function: Creates and opens a new document in the database. Once created, the document behaves like any other document. The new document will not have a valid document ID until written to disk with HTDocumentSave or HTDocumentClose. If the DELAY_COMMIT property is used when closing, a valid document ID will not be assigned until the containing database is closed.

Example

Creates a new empty document.

```
Dim hgdatabase As Long

Dim FormID As Long

Dim hgdocument As Long

Dim hgstatus As Long


hgstatus = HTInit(0)

hgstatus = HTDatabaseOpen("", "test.nsf", 0, hgdatabase)

FormID = HTFormLocateByName(hgdatabase, "Main Topic")

hgstatus = HTDocumentCreate(hgdatabase, FormID, hgdocument)

hgstatus = HTDocumentClose(hgdocument, 0&)

hgstatus = HTDatabaseClose(hgdatabase, HTDBCLOSEF_SAVE_DOCS)

hgstatus = HTTerm(0)
```

Following is an explanation of the code.

We need to specify a form when we create a new document in the database. The form can be specified by the form ID. In HiTest, we present the form ID as long.

```
Dim FormID As Long
```

After opening a database, we specify the form ID by the form name. Here we use the Main Topic form of the TEST database.

```
FormID = HTFormLocateByName(hgdatabase, "Main Topic")
```

We create and open a new document using FormID and hgdatabase. The document handle is returned to hgdocument. If this is unsuccessful, an error code is returned to hgstatus referring to an invalid database or invalid form.

```
hgstatus = HTDocumentCreate(hgdatabase, FormID, hgdocument)
```

Removing a Document

HTDocumentDelete function: Removes an entire document.

It works this way:

Declare Function HTDocumentDelete Lib “W3HTAPI.DLL” (ByVal hgdatabase As Long, ByVal hgdocid As Long, ByVal hgdocdelete_flags As Long) As Long

’hgdatabase Input

HTDocumentDelete (hgdatabase, hgdocid, hgdocdelete_flags)

Functions	*Type*	*Descriptions*
hgdatabase	input	The database containing the document
hgdocid	input	The document id of the document to be deleted
hgdocdelete_flags	input	Flags affect the operation of this function HTDOCDELETEF_NO_STUB: Do not save a deletion stub HTDOCDELETEF_DELAY_COMMIT: Override database delay commit property on HTDOCDELETEF_FORCE_COMMIT: Override database delay commit property off

If the function fails, an error code is returned to hgstatus. Error codes include:

HTFAIL_INVALID_DOCUMENT (document does not exist)

HTFAIL_DESIGN_DOCUMENT (non-data document without docopen_design)

Copying a Document

HTDocumentCopy function: Creates an entire hierarchy copy of the source document from a view-based index in the destination database.

The function is defined as HTDocumentCopy(hgsrc_database, src-docid,dest_database, dest_docid)

When the function is unsuccessful, it returns an error code to hgstatus. Failure codes include:

HTFAIL_INVALID_DATABASE (invalid destination database)

HTFAIL_INVALID_DOCUMENT (source document does not exist)

Other Functions to Work with Documents

HTDocumentClose: Closes an open document.

HTDocumentEncrypt: Encrypts items within a document.

HTDocumentExecute: Executes a formula against a document.

HTDocumentLocateByClass: Locates a unique document given its document class.

HTDocumentLocateByUNID: Locates a unique document given its universal ID.

HTDocumentOpen: Opens a document for data access or modification.

HTDocumentSave: Saves changes to an open document.

HTDocumentValidate: Validates an open document.

Accessing Notes Documents in Visual Basic Using Notes Classes through OLE Automation

We use the NotesDocument class to access Notes documents when we are using Notes Classes through OLE automation in Visual Basic. The NotesDocument class lets you examine and manipulate document properties and contents. You gain access to a NotesDocument object through methods provided by the NotesDatabase, NotesView, and NotesDocumentCollection classes.

Following is a table comparing equivalent functions in HiTest and in LotusScript.

HiTest Function Name	*Notes Classes*	*Notes Methods and Properties*	*Summary*
HTDocumentClose			Closes an open document.
HTDocumentCopy	NotesDocument	CopyToDatabase	Copies a document between databases.
HTDocumentCreate	NotesDatabase	CreateDocument	Creates a new, empty document.

Continued

HiTest Function Name	*Notes Classes*	*Notes Methods and Properties*	*Summary*
HTDocumentDelete	NotesDocument	Remove	Deletes a document.
HTDocumentEncrypt	NotesDocument	Encrypt	Encrypts items within a document.
HTDocumentExecute			Executes a formula against a document.
HTDocumentGetProperty	NotesDocument	Created, LastModified, LastAccessed, IsResponse, ParentDocumentUNID, NoteID, UniversalID, EncryptOnSend, SignOnSend, HasEmbedded, IsUnread (only for read only in Release. 4.5) , ISUIDocOpen (only in Release. 4.5)	Obtains a document-level property value.
HTDocumentLocateByClass			Locates a unique document given its document class.
HTDocumentLocateByUnid			Locates a document given its universal ID.
HTDocumentOpen			Opens a document for data access or modification.
HTDocumentSave	NotesDocument	Save	Saves changes to an open document.
HTDocumentSetProperty	NotesDocument	EncryptOnSend, SignOnSend, MakeResponse, Verifier, FTSearchScore, ParentView, IsNewNote, Responses, etc.	Assigns the value of a document-level property.
HTDocumentValidate			Validates an open document.
	NotesDocument	MarkRead	Marks document as read.
	NotesDocument	MarkUnread	Marks document as unread.

Accessing the Document Properties

Notes classes provide different NotesDocument properties than provided by HiTest. Only EncryptOnSend, EncryptionKeys, SaveMessageOnSend, and SignOnSend are read-write properties. The other properties are read-only properties.

For more detailed information on NotesDocument, refer to Chapter 3.

Let's look at an example.

Example

This agent example displays the properties of the first document in the TEST database.

```
Dim session As Object
Dim db As Object
Dim dc As Object
Dim doc As Object
Set session = CreateObject ("Notes.NotesSession")
Set db = session.GetDatabase ("","test.nsf")
Set dc = db.AllDocuments
Set doc = dc.GetFirstDocument(1247)
Print ( "Created: " & doc.Created)
Print ( "Last modified: " & doc.LastModified)
Print ("Note ID: " & doc.NoteID)
```

Following is an explanation of the code.

We declare dc as an object which will reference the document collection in the database.

```
Dim dc As Object
```

We declare doc as an object which will reference the Notes document. It descends from NotesDocumentCollection.

```
Dim doc As Object
```

We get a reference to NotesDocument Collection from the AllDocument method of NotesDatabase.

```
Set dc = db.AllDocuments
```

We get a reference to the first document using the GetFirstDocument method of the NotesDocument Collection object and store it in doc. 1247 refers to the database type.

```
Set doc = dc.GetFirstDocument(1247)
```

We get the date when the document was created using the CreateData property of NotesDocument.

```
Print ( "Created: " & doc.Created)
```

We get the document ID and last modified date using the NotesID and LastModified properties of NotesDocument and display them.

```
Print ( "Last modified: " & doc.LastModified)
```

```
Print ( "Note ID: " & doc.NoteID)
```

Creating a Document

We use the CreateDocument method of NotesDatabase to create a Notes document. For an example, refer to Chapter 4.

Removing a Document

The Remove method of NotesDocument removes a document from a database. An argument permits you to force the removal (True) or make it dependent on the document not being modified by someone else since the program started (False).

Let's look at an example.

Example

```
Dim session As Object
Dim db As Object
Dim dc As Object
Dim doc As Object
Set session = CreateObject("Notes.NotesSession")
Set db = session.GetDatabase("", "test.nsf")
Set dc = db.AllDocuments
Set doc = dc.GetFirstDocument(1247)
Call doc.Remove(True)
```

Following is an explanation of the code.

We get the first document in the document list. 1247 refers to the database type.

```
Set doc = dc.GetFirstDocument(1247)
```

We remove the document using the Remove method of NotesDocument.

```
Call doc.Remove(True)
```

Copying a Document

Using Notes classes through OLE automation, the CopyToDatabase method copies a document to a specified database (which can be the same database in which the document exists).

Let's look at an example.

Example

This example copies the first document in test.nsf to the test1.nsf database.

```
Dim session As Object
Dim db As Object
Dim db1 As Object
Dim dc As Object
Dim doc As Object
Set session = CreateObject("Notes.NotesSession")
Set db = session.GetDatabase("", "test.nsf")
Set dc = db.AllDocuments
Set doc = dc.GetFirstDocument(1247)
Set db1 = session.GetDatabase ("","test1.nsf")
Call doc.CopyToDatabase(db1)
```

Following is an explanation of the code:

We get the reference to test.nsf.

```
Set db = session.GetDatabase("", "test.nsf")
```

We get the reference to test1.nsf.

```
Set db1 = session.GetDatabase ("","test1.nsf")
```

We copy the first document to test1.nsf.

```
Call doc.CopyToDatabase(db1)
```

Accessing the Current Document

We can access the document that is currently open through the NotesUIWorkspace and NotesUIDocument classes. The NotesUIWorkspace class has one property, CurrentDocument, which is a reference to the current document, and has the following methods:

NotesUIDocument = ComposeDocument(file, form) creates a new document and makes it the current document.

NotesUIDocument = EditDocument([bool]) puts the current document in edit mode.

For more information about the NotesCurrentDocument class, refer to Chapter 3.

Let's look at an example.

Example

This example executes when the user enters a field in edit mode. The script displays information about the field and the document. You can try this example on a form you created.

```
    Dim workspace As Object
    Dim uidoc As Object
    Set workspace = CreateObject("Notes.NotesUIWorkspace")
    Set uidoc = workspace.CurrentDocument
    If uidoc.EditMode Then
        If uidoc.IsNewDoc Then
            Print_
             ("This is the " & _
             uidoc.CurrentField & " field")
        Else
             Print _
             ("This is the " & uidoc.CurrentField & _
             " field of " & uidoc.WindowTitle)
        End If
    End If
End Sub
```

Folllowing is an explanation of the code.

Current document is a unique object class in Notes. We declare uidoc as an object to reference the current document.

```
Dim uidoc As Object
```

We get a reference to the current document using the CurrentDocument property of the NotesUISession class and store it in uidoc.

```
Set uidoc = workspace.CurrentDocument
```

If the current document is in edit mode, we look at the document. If the document is a new document we display the current field. If not, we display the current field and specify the current document's window title.

```
If uidoc.EditMode Then
        If uidoc.IsNewDoc Then
             Print _
             ("This is the " & _
             uidoc.CurrentField & " field")
        Else
             Print _
             ("This is the " & uidoc.CurrentField & _
             " field of " & uidoc.WindowTitle)
        End If
    End If
```

Accessing Notes Documents in LotusScript

When using LotusScript in Notes, you can use some methods that are not available in Visual Basic. This makes the code more simple and more powerful.

Accessing the Document Properties

The Authors property of NotesDocument is a string array which contains all the names of the authors. Using the Forall function provides you with easy access.

Let's look at an example.

Example

This agent example displays the properties of the first document in the TEST database in a message box. The example is similar to the one in Visual Basic, however, using the Forall function, you will notice how easily you can get the author names.

```
Sub Initialize
     Dim db As New NotesDatabase ("", "test.nsf"))
     Dim dc As NotesDocumentCollection
     Dim doc As NotesDocument
```

```
    Set dc = db.AllDocuments
    Set doc = dc.GetFirstDocument()
    Forall docAuthor In doc.Authors
        Messagebox docAuthor
    End Forall
    Messagebox "Created: " & doc.Created
    Messagebox "Last modified: " & doc.LastModified
    Messagebox "Note ID: " & doc.NoteID
End Sub
```

Following is an explanation of the code.

We declare dc as a Notes document collection object which descends from NotesDatabase.

```
Dim dc As NotesDocumentCollection
```

We declare doc as a Notes document object which descends from Notes Document Collection.

```
Dim doc As NotesDocument
```

We get a reference to NotesDocument Collection from the AllDocument method of NotesDatabase.

```
Set dc = db.AllDocuments
```

We get a reference to the first document using the GetFirstDocument method of the NotesDocument Collection object and store it in doc.

```
Set doc = dc.GetFirstDocument()
```

We get all the author names of the first document and put them in the message box.

```
Forall docAuthor In doc.Authors
        Messagebox docAuthor
End Forall
```

We get the date when the document was created using the CreateData property of NotesDocument.

```
Messagebox "Created: " & doc.Created
```

We get the document ID and last modified date using the NotesID and LastModified properties of NotesDocument and display them in message boxes.

```
Messagebox "Last modified: " & doc.LastModified
Messagebox "Note ID: " & doc.NoteID
```

Creating a Document

In LotusScript we can use the New method of NotesDocument to create a new document in a database.

Example

This example creates a new document in a database and adds items to the Subject, Categories and Body fields of the new document. To run this example, use the TEST database created earlier.

```
Sub Initialize
     Dim session As New NotesSession
     Dim db As NotesDatabase
     Dim view As NotesView
     Dim doc As NotesDocument
     Set db = session.CurrentDatabase
     Set doc = New NotesDocument(db)
     doc.Form = "Main Topic"
     doc.Subject = "LotusScript & HiTest VB API"
     doc.Categories = "Eastern"
     doc.Body = "OK"
     Call doc.Save(True, False)
End Sub
```

Following is an explanation of the code:

We create a new document using the New method of NotesDocument.

```
Set doc = New NotesDocument(db)
```

We specify the form of the document using the Form property of the document.

```
doc.Form = "Main Topic"
```

We enter text in the Subject, Categories and Body fields of the “Main Topic” document.

```
doc.Subject = "LotusScript & HiTest VB API"
```

```
doc.Categories = "Eastern"
```

```
doc.Body = "OK"
```

We save the new document. The document will be saved even if somebody tries to access it while the script is running.

```
Call doc.Save(True, False)
```

Note Removing a document, copying a document, and accessing the current document works just the same as when using Notes classes through OLE automation in Visual Basic. Refer to the appropriate sections in this chapter.

Chapter 10
Accessing Notes Items

In the user interface, Notes displays items in a document through fields on a form. When a field on a form and an item in a document have the same name, the field displays the item; for example, the Subject field displays the Subject item.

In this chapter, we will discuss how to access Notes items (fields):

- In HiTest
- Using Notes classes through OLE automation in Visual Basic
- Using LotusScript in Notes.

Accessing Notes Items (Fields) Using the HiTest Basic API

There are five classes in HiTest associated with Notes items: Item, Field, Attachment, CDRecord, and Composite.

Items

Notes stores individual data values in items, which in turn make up documents. The HiTest API strengthens the sometimes vague distinction between the terms item and field by always using item for document items and field for form fields. The primary attributes of an item are an item name, a data type, and a data value. With the HiTest API, items can be accessed either from open documents, or by binding items to the results in an index. While Lotus Notes itself imposes no restrictions on items within a document, the HiTest API adds a minimal set of qualifications which support basic access and facilitate simpler item access.

Fields

A field is the component of a form which describes a single data item within a document. Each form contains one or more fields. The primary attributes of a field are a name and data type. When strict binding is in effect, all items within a document must have a corresponding field within the document's form.

Attachments

Internally, Lotus Notes stores file attachments as items of a particular data type. The HiTest API handles files differently since they may have different actions performed on them, specifically attaching and extracting. Notes stores files attached to a document within items named $FILE, and these are the items accessed by the file functions. The primary attribute of a file is a file name (which does not include any path information). When attachments are created with the HiTest API, they are rendered through the Notes user interface at the end of their document.

CDRecord

CDRecord is a composite data record. One or more ordered CDRecords make up a composite data object. The primary attributes of a CDRecord are a CDRecord type, an index within the composite, and usually a compound value (some CDRecords have no data beyond their type and index). The HiTest API does not support all CDRecord types available within Lotus Notes, although the common types are supported. Each CDRecord type which contains data is manipulated with a CDRecord structure prefixed with HTCD.

Composites

Composite data is synonymous with rich text or compound text. Composite objects are a special free-form compound data type within Lotus Notes. Each composite consists of one or more subcomponents called CDRecords. Internally, Notes stores a composite object as one or more items within a document. When using multiple items, the name is the same for all the items, and HiTest handles and presents them as a single item. The primary attributes of a composite are an item name, a handle, and a value. HiTest uses the constant NULLHANDLE to represent an invalid composite handle.

Let's have a look at some of the item functions:

Copying an Item

HTItemCopy Function copies an item from one document to another. If an item of the same name exists in the destination document, that item is replaced.

Example

This example copies an item from one document to another and then deletes the item from the original document. In effect, it moves the item from one document to another.

```
Dim hgstatus As Long
```

```
Dim hgdatabase As Long
Dim hgindex As Long
Dim hgbegin_datetime As HTDATETIME
Dim hgdocid As Long
Dim hgindent As Long
Dim hgsrc_document As Long
Dim hgdest_document As Long
Dim hgitemname As String

hgstatus = HTInit(0)
hgstatus = HTDatabaseOpen("", "test.nsf", 0&, hgdatabase)
hgstatus = HTIndexOpen(hgdatabase, hgindex)
hgstatus = HTIndexSearch(hgindex, "", 0&, hgbegin_datetime)
hgstatus = HTIndexNavigate(hgindex, HTNav_Next, 0&, hgdocid,
hgindent)
hgstatus = HTDocumentOpen(hgdatabase, hgdocid, 0&,
hgsrc_document)
hgstatus = HTIndexNavigate(hgindex, HTNav_Next, 0&, hgdocid,
hgindent)
hgstatus = HTDocumentOpen(hgdatabase, hgdocid, 0&,
hgdest_document)
hgitemname = "Subject"
hgstatus = HTItemCopy(hgsrc_document, hgitemname,
hgdest_document)
If hgstatus = 0 Then
        Print hgitemname & " has been copied to the
destination document."
End If
hgstatus = HTItemDelete(hgsrc_document, hgitemname)
If hgstatus = 0 Then
        Print hgitemname & " has been deleted from the source
document."
End If
hgstatus = HTDocumentClose(hgsrc_document, 0&)
```

```
hgstatus = HTDocumentClose(hgdest_document, 0&)

hgstatus = HTIndexClose(hgindex)

hgstatus = HTDatabaseClose(hgdatabase, 0&)

hgstatus = HTTerm(0)
```

Following is an explanation of the code.

We declare hgsrc_document and hgdest_docment to reference the source and destination documents.

```
Dim hgsrc_document As Long

Dim hgdest_document As Long
```

We declare hgitemname as the item name. In HiTest, a Notes item is specified by an item name and presented as long.

```
Dim hgitemname As String
```

After we get the reference of the source and destination document and store the handles in hgsrc_document and hgdest_document, we set the item name as “Subject”.

```
hgitemname = "Subject"
```

We copy the field of the source document to the destination document. The field is specified by hgitemname. If copying is unsuccessful, an error code will return to hgstatus.

```
hgstatus = HTItemCopy(hgsrc_document, hgitemname,
hgdest_document)
```

If the copying is successful, we print out a message to inform the user that the item has been copied.

```
If hgstatus = 0 Then

        Print hgitemname & " has been copied to the
destination document."
```

We delete the item in the source document. If the action is unsuccessful, an error code will return to hgstatus.

```
hgstatus = HTItemDelete(hgsrc_document, hgitemname)
```

If the action is successful, we also print a message to inform the user.

```
If hgstatus = 0 Then

        Print hgitemname & " has been deleted from the source
document."
```

Deleting an Item

The HTItemDelete function deletes an item from a document. This function does not work on file attachments (use the HTAttachment functions to manipulate or delete file attachments). Deleting a composite item deletes all items of that name from the document (the HiTest API considers all composite items of the same name as a single item). Deleting an open composite item (that is, represented by a valid composite handle) invalidates that composite handle.

For more information, refer to the example on copying an item using HiTest.

Accessing Notes Items (Fields) in Visual Basic Using Notes Classes through OLE Automation

When we use Notes classes through OLE automation in Visual Basic and LotusScript in Lotus Notes, there are three classes to work with Notes items: NotesItem, NotesEmbeddedObject and NotesRichTextItem. You gain access to these objects using various methods of the NotesDocument and NotesRichTextItem classes.

The NotesRichTextItem class inherits from NotesItem, meaning that NotesRichTextItem objects can use all the properties and methods of NotesItem.

NotesEmbeddedObject represents any one of the following: An embedded object, an object link and a file attachment.

Note When we use the EmbedObject method of NotesRichTextItem in Visual Basic, the argument type of the method is type number rather than type constant which we can use in LotusScript.

Type Constant	*Type Number*
EMBED_ATTACHMENT	1454
EMBED_OBJECT	1453
EMBED_OBJECTLINK	1452

Tip When using Notes classes through OLE automation, if Visual Basic cannot recognize a type, you can always use **`Message Cstr(TypeConstant)`** in LotusScript to find out the type number.

Note The DoVerb method of NotesEmbeddedObject is used differently depending on the platform and the OLE server. For example, you use Embobj.DoVerb “&Open” in Windows 95, and Embobj.DoVerb “Edit” in

Windows 3.1. For details on the OLE server, refer to the appropriate OLE server documentation.

Following is a table which translates the HiTest functions to the equivalent Notes classes, methods and properties.

HiTest Functions	*Notes Class*	*Notes Class Properties and Methods*	*Summary*
HTItemAppendText_List	NotesDocument	AppendToTextList	Appends a text entry to a text list within a document.
HTItemCopy	NotesDocument NotesItem	CopyItem CopyItemToDocument	Copies an item from one document to another.
HTItemDelete	NotesDocument	RemoveItem	Deletes an item from a document.
HTItemDelete	NotesItem	Remove	Deletes an item from a document.
HTItemFetch	NotesDocument	GetItemValue, HasItem, GetItemValue, Signer, IsSigned, Authors, GetNextItem in Release 4.5	Converts and retrieves the data for an item into a supplied buffer.
	NotesItem	Values, DateTimeValue	Converts and retrieves the data for an item into a supplied buffer.
HTItemGetCount			Obtains the number of items in a document.
HTItemGetInfo			Obtains information about a document item.
HTItemGetLength			Obtains the length of an item as converted to a specified data type.
HTItemList	NotesDocument	GetFirstItem, GetNextItem, Items	Iterates through items in a document.
	NotesItem	Name, ValueLength, Type, IsNames, IsReaders, IsAuthors, IsSigned, IsEncrypted, IsSummary, IsProtected	
HTItemPut	NotesDocument	ReplaceItem, AppendItemValue, Sign, PutInFolder, RemoveFromFolder	Writes an item to a document, overwriting any existing item of the same name.

Copying an Item

Use the CopyItemToDocument method of the NotesItem class to copy the current item to another document. Use the CopyAllItems method of NotesDocument to copy all items in the current document to another document.

After copying an item, you must call the Save method for the NotesDocument object containing the new item. Otherwise the update is lost when the program exits.

Let's look at an example:

Example

In the previous section on how to access Notes items in HiTest, we discussed an example showing how to copy a Subject item from the first document to the second document, and how to delete the original item. Following is the equivalent code using Notes classes through OLE automation in Visual Basic.

```
Dim session As Object
Dim db As Object
Dim view As Object
Dim doc_src As Object
Dim doc_dest As Object
Dim item As Object
Set session = CreateObject("notes.notessession")
Set db = session.GetDatabase("", "test.nsf")
Set view = db.GetView("By Author")
Set doc_src = view.GetFirstDocument()
Set doc_dest = view.GetNextDocument(doc_src)
Set item = doc_src.GetFirstItem("Subject")
Call item.CopyItemToDocument(doc_dest, "Subject")
Call item.Remove
Call doc_src.Save(True, False)
Call doc_dest.Save(True, False)
Set session = Nothing
```

Following is an explanation of the code:

We declare doc_src and doc_dest which will reference the source (first) document and destination (second) document in the By Author View.

```
Dim doc_src As Object
```

```
Dim doc_dest As Object
```

We declare item as an object which will reference the Subject item in the source document.

```
Dim item As Object
```

We get the reference to the source and destination documents using the view's methods:

```
Set doc_src = view.GetFirstDocument()
```

```
Set doc_dest = view.GetNextDocument(doc_src)
```

We get the Subject item using the GetFirstItem method of NotesDocument.

```
Set item = doc_src.GetFirstItem("Subject")
```

We copy the Subject item of the source document to the destination document.

```
Call item.CopyItemToDocument(doc_dest, "Subject")
```

We remove the Subject item in the source document using the Remove method of NotesItem.

```
Call item.Remove
```

Deleting an Item

Use either the Remove method of the NotesItem class or the RemoveItem method of NotesDocument to remove an item from a document. The Remove method removes only the current object; other items with the same name remain in existence. The RemoveItem method removes all items with the specified name.

After removing an item, you must call the Save method for the NotesDocument object that contained the item or the update is lost when the program exits.

For an example, refer to the section on how to copy an item using Notes classes through OLE automation.

Accessing Notes Items (Fields) in LotusScript

By changing the declaration of the variables in Visual Basic using Notes classes through OLE automation, we can run the code as LotusScript code. Refer to the previous examples on how to copy or delete an item in Visual Basic using Notes classes through OLE automation.

We can use the New method of NotesItem and NotesRichTextItem in LotusScript. We can also use the Forall and Foreach statements to enhance our code's performance.

LotusScript also provides strong functions to access OLE objects. Let's look at an example.

Example

The following example shows how to access OLE objects using the EmbeddedObjects property of NotesDocument. The script accesses the last document in the All Documents view of HILL.NSF on a server called SanFrancisco.

The document contains the following:

- A Word Pro embedded object called "Word Pro Document" in the Body item.
- A Microsoft Excel object link called "MS® Excel Worksheet" in the Body item.
- An Ami Pro embedded object, created in Notes Release 4, called "Ami Pro Document" in the Body item.
- A file attachment called "CASTLE.BMP" in the Body item.
- A Freelance Graphics embedded object called "Freelance Presentation" in the Description item.
- A 1-2-3 embedded object called "123 Worksheet" that is embedded in the form used to create the document, and has been subsequently activated and edited by a user.

This script uses the EmbeddedObjects property of NotesDocument, and displays the following:

- "Word Pro Document"
- "MS Excel Worksheet"
- "Freelance Presentation"
- "123 Worksheet"

```
Dim db As NotesDatabase
Dim view As NotesView
Dim doc As NotesDocument
Set db = New NotesDatabase( "SanFrancisco", "hill.nsf" )
Set view = db.GetView( "All Documents" )
Set doc = view.GetLastDocument
Forall o In doc.EmbeddedObjects
  Messagebox( o.Name )
End Forall
```

Following is an explanation of the code:

We get a reference to the hill.nsf database on server SanFrancisco.

```
Set db = New NotesDatabase( "SanFrancisco", "hill.nsf" )
```

We get the All Document view.

```
Set view = db.GetView( "All Documents" )
```

We get a reference to the last document in the All Document view and store it in doc.

```
Set doc = view.GetLastDocument
```

We go through all the embedded objects in the last document and display their names.

```
Forall o In doc.EmbeddedObjects
  Messagebox( o.Name )
End Forall
```

For more information on LotusScript OLE objects, refer to Chapters 11 and 12.

Chapter 11
Using Notes as an OLE 2 Automation Client: Creating Objects

This chapter describes how to create embedded and linked objects in a Notes document manually and using LotusScript.

You can include data from other applications by adding embedded or linked objects to Notes documents. You can:

- Embed part of a file in a Notes document.
- Embed an entire file in a Notes document.
- Embed new data in a Notes document.
- Link a file to a Notes document.

Creating an Embedded Object

In this section you are going to create an embedded object in a Notes document.

Embedding Part of a File in a Document

You can embed data from another application as an object in a Notes document. This way, you can use an OLE server application to enter and edit data in Notes. This combines the features of Notes that facilitate sharing the data with the custom features of applications that can create data. Notes becomes the "container" for the application's data, and you can still use the original application to edit, modify, or add data.

You can paste data from OLE server applications, and you can also drag and drop data from OLE 2 server applications.

The following example shows how to embed part of a Lotus 1-2-3 file into a Notes document.

Creating a Notes Database and a Form

Create a form for the OLE 2 database, and then use this form to embed or link a particular object in a Notes document. In our example, you will embed or link an object in a Notes rich text field.

Perform the following steps to create a form in Notes:

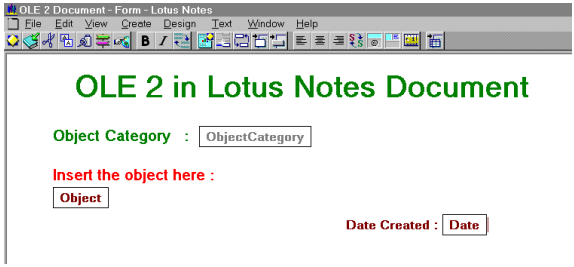
1. Create a new database and save it with OLE 2 as the name.
2. Create a form in the OLE 2 database and save it with OLE 2 Document as the name.
3. Click the Design Form Properties SmartIcon to display an InfoBox and click on the arrow next to Versioning to display the list of available choices.
4. Select Prior versions become responses.

   Tip You can also click the right mouse button and select Form Properties to display the InfoBox.
5. In the upper part of the Form Builder window, type OLE 2 in Lotus Notes Document and change the font size of the title line to 24 and the style to bold.

Next, create an editable Keywords field.

6. On the next line, type Object Category:, which is static text.
7. Click the Create Field SmartIcon to create the new field next to the static text and type ObjectCategory in the Name field.
8. Under Type, drop down the list of available choices and select Keywords.
9. In the list box under Choices, type two keywords: Embedding and Linking.
10. Click the Interface tab, which is the second tab from the left on the InfoBox, and select Radio button.
11. Move the cursor to the next line, and under Object Category, type Insert the object here:.
12. Change the font size of the text to 12 and the style to bold.
13. On the next line, create an editable rich text field and call it Object. This field will be used to store an embedded or a linked object.
14. Move to the next line, tab a little to the right and type Date Created:.
15. Create an editable Time field and call it Date.
16. On the programming pane, select Formula radio button and type @Created.

17. Press ESC and save the form. The form you just created should look like this:



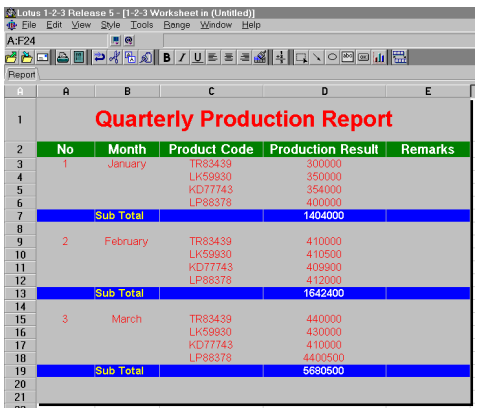
Pasting Data from an OLE Server Application into a Notes Document

Now that you have created a Notes form, use this form to create a Notes document and paste data from a Lotus 1-2-3 document into the Notes document.

Note You must leave Notes up and running. Minimize it, but do not close it.

Perform the following steps to paste Lotus 1-2-3 data into a Notes document.

1. First create a spreadsheet in Lotus 1-2-3. If you need help, refer to the Lotus 1-2-3 User's Guide. Here is an example of a spreadsheet:

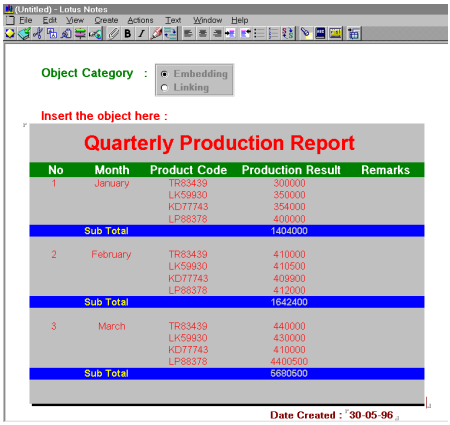


	No	Month	Product Code	Production Result	Remarks
3	1	January	TR83439	300000	
4			LK59930	350000	
5			KD77743	354000	
6			LP88378	400000	
7		Sub Total		1404000	
8					
9	2	February	TR83439	410000	
10			LK59930	410500	
11			KD77743	409900	
12			LP88378	412000	
13		Sub Total		1642400	
14					
15	3	March	TR83439	440000	
16			LK59930	430000	
17			KD77743	410000	
18			LP88378	4400500	
19		Sub Total		5680500	
20					
21					

2. Select the data in the spreadsheet you want to embed and choose Edit - Copy to copy it to the clipboard.
3. Switch to the Notes window and create a document based on the OLE 2 Document form you created earlier.
4. Click the Embedding radio button and click in the Object field.
5. In the Notes document, choose Edit - Paste Special and select the 1-2-3 Worksheet in the As: box.

Tip You can select Display as icon to display an icon rather than the embedded data. Notes displays the server application's icon by default. To display a different icon, click Change Icon, select a different icon, and click OK.

6. Click OK and save the document. The Notes document should look like this:

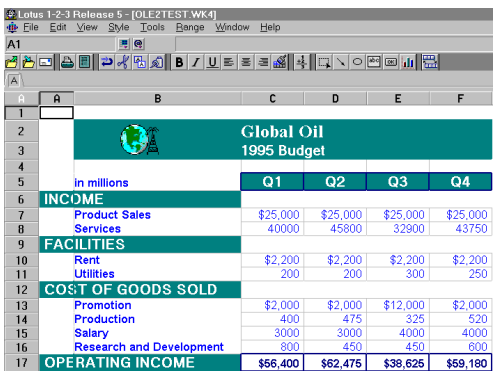


Embedding an Entire File in a Document

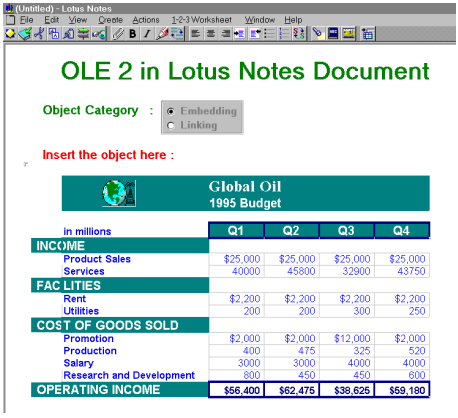
You can embed an entire file created with another application as an object in a Notes document. This way, you can use an OLE server application to enter and edit data in Notes.

The following example shows you how to embed a Lotus 1-2-3 file into a Notes document.

1. Create a new Lotus 1-2-3 spreadsheet or use an existing spreadsheet and save it as OLE2TEST.WK4 in your Notes path, for example, c:\notes\data\123. Our spreadsheet looks like this:



2. Switch to the Notes window and create a document based on the OLE 2 Document form you created earlier on.
3. Click the Embedding radio button and click in the Object field.
4. Choose Create - Object and click the Create an object from a file radio button in the Create Object dialog box.
5. In the File box, type c:\notes\data\123\ole2test.wk4.
6. Click OK and save the document. The Notes document should look like this:



Embedding a New Object in a Document

You can embed a new object in a Notes document. This way, you can use an OLE server application to enter and edit data in Notes.

When you create a new object, Notes opens a blank work file in the application you select so you can enter data. When you save the data, it is saved as an object in Notes (instead of as a separate file).

Note If you select an OLE 2 server application, Notes opens the blank work file directly in Notes.

In this example you embed data from Word Pro 96 into a Notes document. Since Word Pro 96 is an OLE 2 server application, you can edit a Word Pro 96 file in a Notes document directly (without launching Word Pro 96) but you must have an installed copy of Word Pro 96.

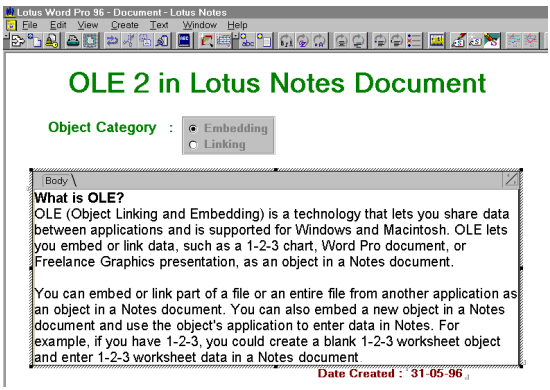
Note The user who receives this document from you will be able to view it but not edit the Word Pro 96 document.

To do so, follow these steps:

1. Create a Notes document based on the OLE 2 Document form you created earlier.
2. Click the Embedding radio button and then click in the Object field.

3. Choose Create - Object from the menu bar.
4. Select Lotus Word Pro 96 document, in the Object type box, then click OK.
5. Type some phrases in Word Pro 96. When you edit the Word Pro document, in a Notes document, you are actually in a Word Pro environment. You can see the Word Pro icons at the bottom of the screen. They look like this :

6. To leave the Word Pro environment editor, click anywhere in the Notes document outside the new object or press **ESC** to leave the Word Pro environment editor.
7. When you are done, save this Notes document. It should look like this:



Creating a Linked Object

You can link data from another application as an object in a Notes document. This way, you can display the latest data from an OLE server application in Notes. A linked object is a pointer to data in another file; changes made to the original file are reflected in the Notes document. Notes asks whether to update (refresh) a linked object each time the document containing the object is opened.

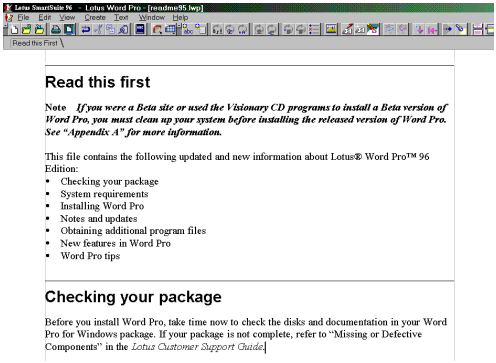
However, since the data is physically contained in a file on the file system of the PC in its server application format, you will “break” the link if you use Notes to e-mail the document or rely on this link in a Notes application. This is a major difference between linking an object and embedding it in the Notes object.

The following example shows you how to link a Word Pro 96 file into a Notes document.

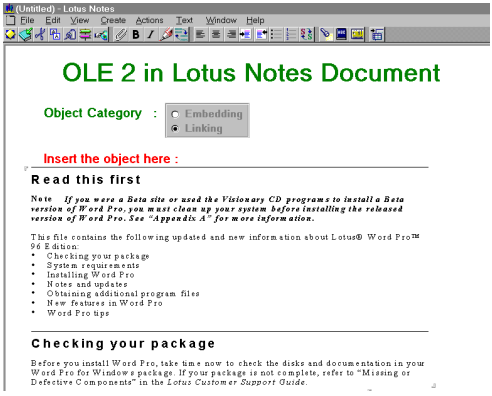
Linking Data to a Notes Document

To link a Word Pro 96 document to a Notes document, follow these steps:

1. Create a Word Pro 96 document and save it in \notes\data\wordpro, for example. In our example, we called the document OLE2TEST.LPW. It looks like this:



2. Open the Word Pro document, select the data you want to link and copy it to the clipboard.
3. Switch to the Notes window and create a document based on the OLE 2 document form you created earlier on.
4. Click the Linking radio button and then click in the Object field.
5. Choose Edit - Paste Special from the menu bar.
6. Click the Paste link to source radio button and select Word Pro 96 document in the As: box.
7. Click OK and save the document. The Notes document should look like this:



Creating an Embedded Object Using LotusScript

In this section you are going to create an embedded object in a Notes document using LotusScript.

Using OLE automation in Notes allows you to combine the services Notes provides with the services of the OLE application. Notes can serve as the container for the application, providing replication, security, and access control as well as summarizing the data in views and making it possible to search for information using Notes' full-text indexing capabilities. The OLE application can provide services that would require extensive effort to develop in Notes, such as the cell engine from Excel or 1-2-3 or the sophisticated text handling capabilities of Word or Word Pro.

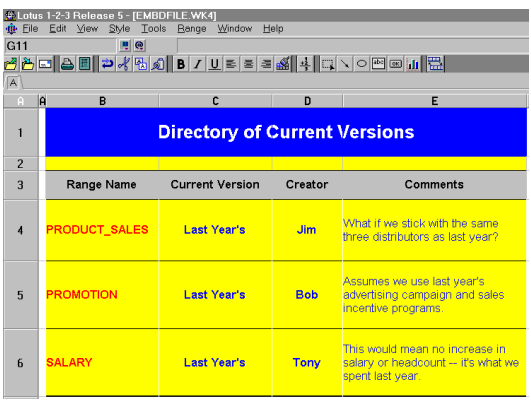
Notes provides the very best container for these objects, as it frees users from having to learn the hierarchical file system (all Notes documents, even those with objects in them, can have a word-based title). In addition, Notes' development capabilities and integrated messaging services make using OLE applications as part of a workflow application fast and easy.

Embedding an Entire File in a Document

You can embed an entire file from another application as an object in a Notes document using LotusScript.

The following example shows how to embed data from a Lotus 1-2-3 spreadsheet file into a Notes document.

1. First, create a new Lotus 1-2-3 spreadsheet or use an existing spreadsheet. In our example, we are using a new spreadsheet. It looks like this:



2. Save this file as EMDBFILE.WK4 in your Notes path, for example, d:\notes\data\123.

3. Open the OLE 2 Notes database which you created earlier. The view window displays.
4. On the navigator pane, select Design, then Forms and then double-click the OLE 2 Document form. The Form Design window displays.
5. Move to the end of the form and choose Create - Hotspot - Button from the menu bar.
6. Type Embed Object from a File in the Button label: box.
7. Next to Properties for: Button, drop down the list of available choices and select Text. Notice how the new button on the form now displays its title.

   Tip With the button properties still displayed, you can also highlight the new button by dragging the cursor from left to right. This automatically changes the button properties to text properties.
8. Click the fourth tab from the left and then select the Opened for reading check box. The Printed check box is selected automatically.
9. On the programming pane, select the Script radio button.
10. Click on the space between Sub Click(Source As Button) and End Sub and type the following LotusScript code:

```
%INCLUDE "LSCONST.LSS"

Sub Click(Source As Button)

   Dim workspace As New NotesUIWorkspace

   Dim uidoc As NotesUIDocument

   Dim fileName, msg As String

   Dim boxType As Long


   Set uidoc = workspace.CurrentDocument

   Call uidoc.FieldSetText ("ObjectCategory",
   "Embedding")

   Call uidoc.GotoField ("Object")

   Call uidoc.CreateObject ("1-2-3 Report", "",

   "D:\NOTES\DATA\123\EMBDFILE.WK4")

   Call uidoc.Save

   boxType = MB_OK + MB_ICONINFORMATION

   msg$ = |File has been embedded !

   Click OK to close this document|
```

```
        Messagebox msg$, boxType, "Successful"
        Call uidoc.Close
    End Sub
```

11. Save the form and close it.

Following is a description of the LotusScript code:

Note In the following, when we explain our code samples, we will explain pieces of code only once. If the same piece of code reappears in a later sample, we will not explain it again.

Declare workspace as an object reference variable of the NotesUIWorkspace class and create a new NotesUIWorkspace object.

```
Dim workspace As New NotesUIWorkspace
```

Declare uidoc as an object reference variable of the NotesUIDocument class.

```
Dim uidoc As NotesUIDocument
```

Declare fileName and msg as string variables and boxType as a Long variable.

```
Dim fileName, msg As String
Dim boxType As Long
```

Assign uidoc as a reference variable of a current document object.

```
Set uidoc = workspace.CurrentDocument
```

Set the ObjectCategory item value with an Embedding string.

```
Call uidoc.FieldSetText ("ObjectCategory", "Embedding")
```

Move the cursor to the Object item.

```
Call uidoc.GotoField ("Object")
```

Create a new embedded object in the Object item. This embedded object is created from the EMBDFILE.WK4 file located in the D:\NOTES\DATA\123 subdirectory.

Note You will receive an error message if the file does not exist in the subdirectory or if it is being used by another process.

```
Call uidoc.CreateObject ("1-2-3 Report", "",
"D:\NOTES\DATA\123\EMBDFILE.WK4")
```

Save the Notes document.

```
Call uidoc.Save
```

Assign the button and icon constant name to the boxType variable.

```
boxType = MB_OK + MB_ICONINFORMATION
```

Assign the string to the msg variable.

```
msg$ = |File has been embedded !
Click OK to close this document|
```

Display a message, from the msg variable, in a message box labeled "Successful" and containing a Yes and No button.

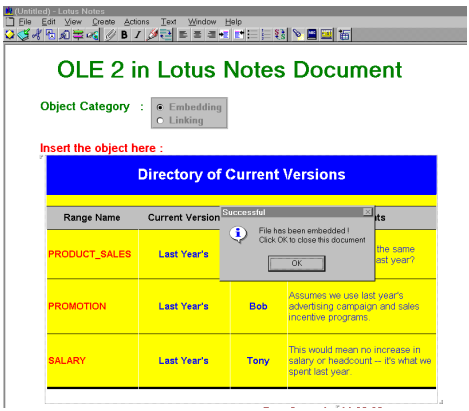
```
Messagebox msg$, boxType, "Successful"
```

Close the Notes document.

```
Call uidoc.Close
```

Next, compose a document using this form.

12. Choose Create - OLE 2 Document from the menu bar.
13. Click the Embed Object from a File button. The spreadsheet is embedded in your document. It looks like this:



14. Save and close the document.

Note You can also embed part of a file in a Notes document using LotusScript. To do so, you need to use the object's native LotusScript function to access the object's data in the object's application. This can be a complex process. Therefore, it might be a better choice to manually embed part of a file in a Notes document.

Embedding a New Object in a Document

In this example we are using the OLE 2 Document form to embed an OLE object in a Notes document. First, follow these steps to modify the form by adding LotusScript code:

1. Open the OLE 2 Notes database. The view window displays.
2. On the navigator pane, select Design, then Forms and open the OLE 2 Document form. The Form Design window displays.
3. Move to the end of the form and click the cursor under the Embed Object from a File button.
4. Choose Create - Hotspot - Button from the menu bar and type Embed New Data Object in the Button label: box.
5. Next to Properties for: Button, drop down the list of available choices and select Text. Notice how the new button on the form now displays its title.
6. Click the fourth tab from the left and select the Opened for reading check box. The Printed check box is selected automatically.
7. On the programming pane, select the Script radio button.
8. Click on the space between Sub Click(Source As Button) and End Sub and type the following LotusScript code:

```
Sub Click(Source As Button)

   Dim workspace As New NotesUIWorkspace

   Dim uidoc As NotesUIDocument

   Dim msg As String


   Set uidoc = workspace.CurrentDocument

   Call uidoc.FieldSetText ("ObjectCategory", _
   "Embedding")

   Call uidoc.GotoField ("Object")

   Call uidoc.CreateObject ("Turbo 123",
   "123Worksheet")

   msg$ = |Switch to 1-2-3 Application to create
   spreadsheet now or

   click OK to leave this 1-2-3 object empty ! |

   Messagebox msg$, MB_OK + MB_ICONINFORMATION, _

   "Information"
```

```
    Call uidoc.Close
End Sub
```

9. Save the form and close it.

Following is a description of the LotusScript code:

Create a new blank embedded object in the Object item. This embedded object is a Lotus 1-2-3 blank spreadsheet. The class name of this object is 123Worksheet.

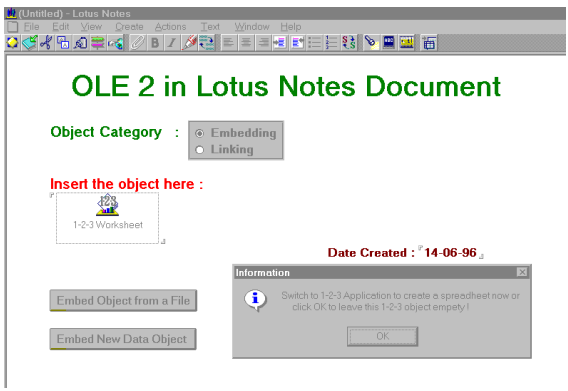
```
Call uidoc.CreateObject ("Turbo 123", "123Worksheet")
```

Display a message, from the msg variable, in a message box labeled "Information" and containing an OK button.

```
Messagebox msg$, MB_OK + MB_ICONINFORMATION, _
"Information"
```

Next, compose a document using this form.

10. Choose Create - OLE 2 Document from the menu bar.
11. Click the Embed New Data Object button. You should see something like this:



Note Clicking the Embed New Data Object button automatically launches Lotus 1-2-3 in the background.

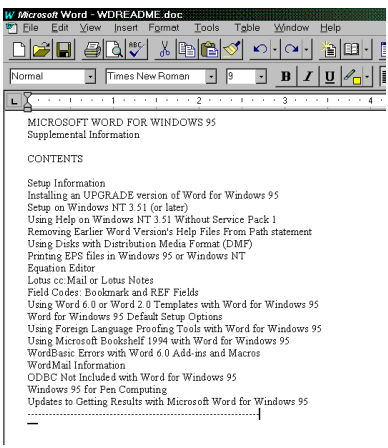
12. If you want to create the spreadsheet straight away, switch to Lotus 1-2-3 and work with it. Do *not* click the OK button on the message window.
13. When you are done, choose File - Close & Return to Lotus Notes from the 1-2-3 menu bar.
14. Click Yes to confirm that you want to update the object.

Note If you do not want to create the spreadsheet straight away, click OK on the message window. A message prompts you whether or not to save the document. If you click Yes, the 1-2-3 object will be saved as a blank object. You can update it later on.

Creating a Linked Object Using LotusScript

In this section you will link data from another application as an object in a Notes document using LotusScript. The following example shows how to link an object from a Microsoft Word document into a Notes document. As we noted before, links are dependent on the directory structure of the file system because they do not store the object's data in the Notes data store. So, links can be broken if the document is mailed to a user who does not have the object's data in the same physical location on their machine.

1. First, create a new MS Word document or use an existing one. If you need help to create this document, refer to the appropriate product manual. We called our document WDREADME.DOC. It looks like this:



2. Open the OLE 2 Notes database. The view window displays.
3. On the navigator pane, select Design, then Forms and open the OLE 2 Document form. The Form Design window displays.
4. Move to the end of the form and click the cursor under the Embed New Data Object button.
5. Choose Create - Hotspot - Button from the menu bar and type Link an Object from a File in the Button label: box.

6. Next to Properties for: Button, drop down the list of available choices and select Text. Notice how the new button on the form now displays its title.
7. Click the fourth tab from the left and select the Opened for reading check box. The Printed check box is selected automatically.
8. On the programming pane, select the Script radio button.
9. Click on the space between Sub Click(Source As Button) and End Sub and type the following LotusScript code:

```
Sub Click(Source As Button)
   Dim session As New NotesSession
   Dim db As NotesDatabase
   Dim doc As NotesDocument
   Dim rtitem As NotesRichTextItem
   Dim object As NotesEmbeddedObject
   Dim workspace As New NotesUIWorkspace
   Dim uidoc As NotesUIDocument
   Dim boxtype As Long
   Dim answer As Integer

   Set db = session.CurrentDatabase
   Set doc = New NotesDocument (db)
   Set rtitem = New NotesRichTextItem (doc,
   "Object")
   doc.Form = "OLE 2 Document"
   doc.ObjectCategory = "Linking"
   doc.Date = Date$

   Set object = rtitem.EmbedObject _
   (EMBED_OBJECTLINK, "", _
   "D:\NOTES\DATA\MSWORD\WDREADME.DOC", _
   "MS Word 6")

   boxtype = MB_YESNO + MB_ICONQUESTION
```

```
        answer% = Messagebox ("Object has beed linked" _
        & Chr(10) & "Save this document ?", _
        boxtype, "Save ?")

        boxtype = MB_OK + MB_ICONINFORMATION
        If answer% = 6 Then
          Call doc.Save (True, False)
          Messagebox "Document has been saved", _
          boxtype, "Saving Information"
        Else
          Messagebox "Document not saved", _
          boxtype, "Saving Information"
        End If
        Set uidoc = workspace.CurrentDocument
        Call uidoc.Close
    End Sub
```

10. Save the form and close it.

Following is a description of the LotusScript code:

Assign db as a reference variable to the current database object.

```
Set db = session.CurrentDatabase
```

Create a new document in the current database.

```
Set doc = New NotesDocument (db)
```

Create a new rich text item in the current document.

```
Set rtitem = New NotesRichTextItem (doc, "Object")
```

Set the Form item with a string of OLE 2 document.

```
doc.Form = "OLE 2 Document"
```

Set the ObjectCategory item with a string of Linking.

```
doc.ObjectCategory = "Linking"
```

Create a new linked object in the Object rich text item. This object is linked from the WDREADME.DOC file located in the D:\NOTES\DATA\MSWORD subdirectory.

```
Set object = rtitem.EmbedObject _
(EMBED_OBJECTLINK, "", _
"D:\NOTES\DATA\MSWORD\WDREADME.DOC", _
"MS Word 6")
```

Display the message "Object has been linked. Save this document?" from the msg variable in a message box labeled "Save" containing a Yes and No button. This message box function returns a value when the button is clicked.

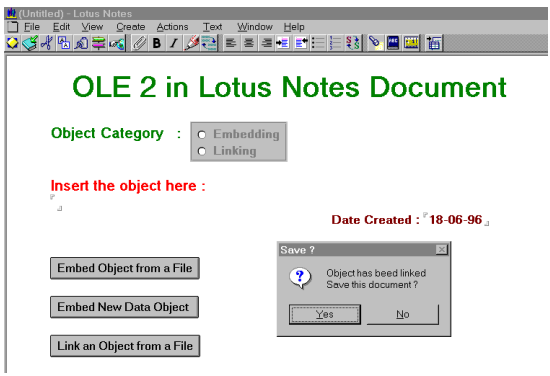
```
answer% = Messagebox ("Object has beed linked" _
& Chr(10) & "Save this document?", _
boxtype, "Save ?")
```

Save this Notes document without creating a response document.

```
Call doc.Save (True, False)
```

Next, compose a document using this form.

11. Choose Create - OLE 2 Document from the menu bar.
12. Click the Link an Object from a File button. A message is displayed. Your Notes document should look like this:



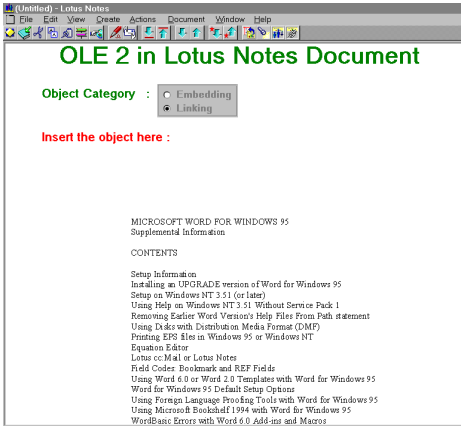
Note You will receive an error message if the object file does not exist or if it is being used by another process.

13. Click Yes to save the document.
14. Click OK to close the document.

Next, open the document.

15. Display the Access OLE 2 view and open the document listed in the Linking category.

16. Click Yes to confirm that you want to refresh the link. The document should look like this:



Chapter 12
Using Notes as an OLE 2 Automation Client: Managing Objects

This chapter describes how to edit and update an embedded or linked object in a Notes document manually and using LotusScript.

You can do the following in a Notes document that contains an OLE object:

- Edit an embedded object. You can edit an embedded OLE object's data by launching the object's server application from Notes. You can edit an embedded OLE 2 object's data "in-place" by using the server application's commands directly in Notes.
- Edit a linked object. You can edit a linked object's data by launching the object's server application from Notes.

Editing an Embedded Object

The following describes how to edit an embedded object manually.

Creating a View

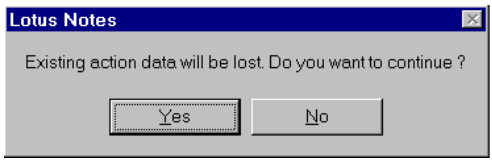
In our example here, before you edit Notes documents that contain embedded or linked objects, you first create a view in Notes to make it easier to access each document.

Follow these steps to create a view in Notes:

1. Open the OLE 2 database.
2. On the navigator pane, click Design, then Views, and then double-click on the *(untitled) view. The View design window displays.
3. Choose Design - View Properties from the menu bar and type Access OLE 2 in the Name box.

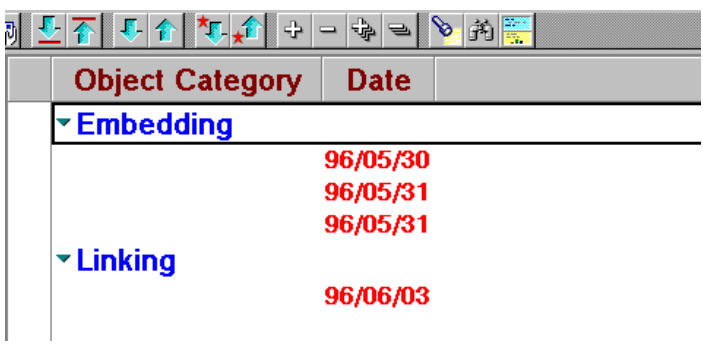
   Tip You can also click the Design View Properties SmartIcon.
4. Click the Options tab and select the Show in View menu check box.
5. Double-click on the "#" column title.
6. Type Object Category in the Title box; this is the title of your first column.

7. Select the Show twistie when row expandable check box.
8. Click the Sorting Tab.
9. Next to Sort, click the Ascending radio button, then, next to Type, click the Categorized radio button.
10. Click the next tab on the InfoBox and select 12 in the Size box and Bold in the Style box.
11. Click the Title tab and select 12 in the Size box and Bold in the Style box.
12. On the programming pane, click the Field radio button and select the ObjectCategory field. Click Yes if the following message displayed:



Next, create the second column.

13. Double-click on the right of the Object Category column and type Date in the Title box, then select 5 in the Width box.
14. Click the Sorting tab and, next to Sort, click the Ascending radio button.
15. Click the next tab on the InfoBox, then select 10 in the Size box and Bold in the Style box.
16. Click the Title tab, then select 12 in the Size box and Bold in the Style box.
17. On the programming pane, select the Field radio button and click Yes if the message appears again, then select the Date field.
18. Save this view, and on the navigator pane, select the Access OLE 2 folder. The view should look like this:

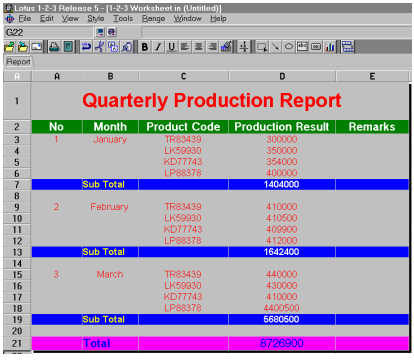


Editing the Object

In this example you will edit the Notes document that contains the embedded object you created in the previous example.

To edit the embedded OLE object in the Notes document, do the following:

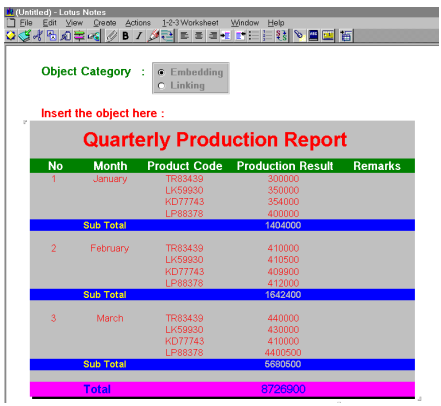
1. Open the Access OLE 2 view and highlight the first document listed in the Embedding category.
2. Press **Ctrl-E** to put the document into edit mode.
3. Double-click the Lotus 1-2-3 object and edit the spreadsheet data using Lotus 1-2-3. For example, insert a row to calculate the total figure for the production results of the quarter. When you are done, the spreadsheet would look like this:



	A	B	C	D	E
1	Quarterly Production Report				
2	No	Month	Product Code	Production Result	Remarks
3	1	January	TR83439	300000	
4			LK59930	350000	
5			KD77743	354000	
6			LP88378	400000	
7		Sub Total		1404000	
8					
9	2	February	TR83439	410000	
10			LK59930	410500	
11			KD77743	409900	
12			LP88378	412000	
13		Sub Total		1642400	
14					
15	3	March	TR83439	440000	
16			LK59930	430000	
17			KD77743	410000	
18			LP88378	4400500	
19		Sub Total		5680500	
20					
21		Total		8726900	

In our example, notice the Total row inserted at the bottom of the spreadsheet.

4. In Lotus 1-2-3, choose File - Update Lotus Notes.
5. Choose File - Exit & Return to leave Lotus 1-2-3 and to return to Lotus Notes. Your Notes document now looks like this:



Object Category : Embedding / Linking

Insert the object here :

Quarterly Production Report				
No	Month	Product Code	Production Result	Remarks
1	January	TR83439	300000	
		LK59930	350000	
		KD77743	354000	
		LP88378	400000	
	Sub Total		1404000	
2	February	TR83439	410000	
		LK59930	410500	
		KD77743	409900	
		LP88378	412000	
	Sub Total		1642400	
3	March	TR83439	440000	
		LK59930	430000	
		KD77743	410000	
		LP88378	4400500	
	Sub Total		5680500	
	Total		8726900	

6. Choose File - Save to save the document.

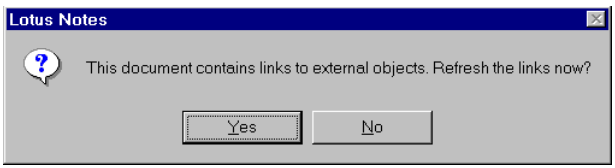
Note If you edit an embedded OLE 2 object in a Notes document, you can also edit the object directly without launching the object's application. However, you must have a copy of the object's application installed. For more details, see the section "Embedding a New Object in a Document" in the previous chapter.

Editing a Linked Object

You can edit a linked object's data by launching the object's server application from Notes.

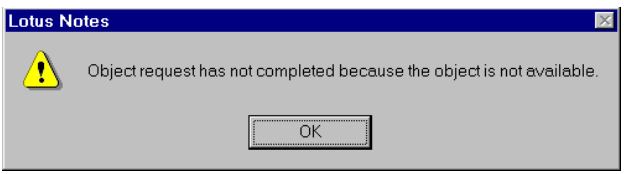
In the following example you will edit a Notes document that contains a linked Word Pro 96 object. To edit this linked object, do the following:

1. Open the Access OLE 2 view and highlight the first document listed in the Linking category.
2. Press **Ctrl-E** to put the document into edit mode.
3. Click Yes when a message displays prompting you to refresh the object link between Notes and Word Pro 96:

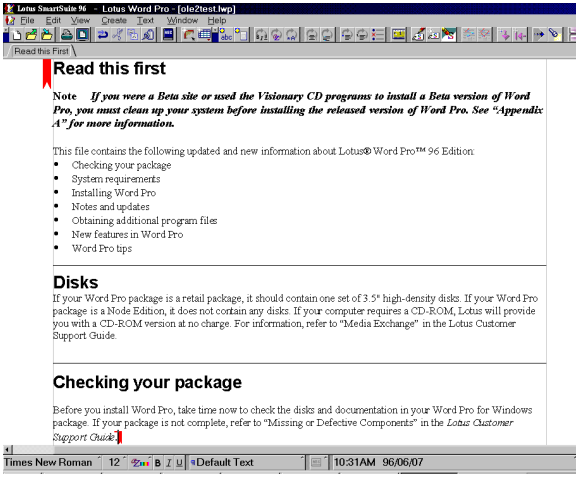


4. Double-click the Word Pro object.

   Note To edit a linked object, you need access to the linked file. If you removed or renamed the linked file, the following warning message is shown:



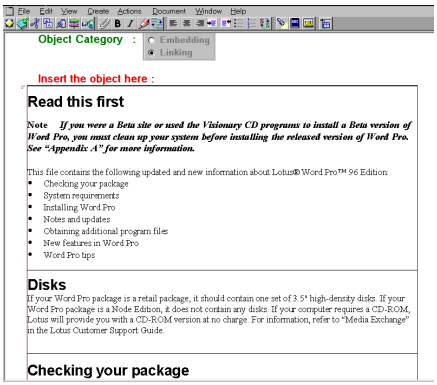
5. When Word Pro 96 is launched, edit the document. Our modified Word Pro document looks like this:



The data you change in the Word Pro 96 document is updated immediately in the Notes document unless the object's update type is Manual.

6. Choose File - Save to save the changes.

7. Choose File - Exit to return to Notes. The Notes document now looks like this:



Note If the object's update type is Manual, press **F9** to refresh the modified Word Pro 96 data in the Notes document.

8. Save the document.

Editing an Embedded or a Linked Object Using LotusScript

You can edit an embedded or a linked object in a Notes document using LotusScript. The following example shows you how to do this.

1. Open the OLE 2 Notes database. The view window displays.
2. On the navigator pane, select Design, then Forms and open the OLE 2 Document form. The Form Design window displays.
3. Move to the end of the form and click the cursor under the Edit Embedded or Linked Object button.
4. Choose Create - Hotspot - Button from the menu bar. An InfoBox displays. It lets you set the properties of the new button.
5. Type Edit Embedded or Linked Object in the Button label: box.
6. Next to Properties for: Button, drop down the list of available choices and select Text. Notice how the new button on the form now displays its title.

   Tip With the button properties still displayed, you can also highlight the new button by dragging the cursor from left to right. This automatically changes the button properties to text properties.
7. Click the fourth tab from the left and select the Opened for reading check box. The Printed check box is selected automatically.
8. On the programming pane, select the Script radio button.
9. Click on the space between Sub Click(Source As Button) and End Sub and type the following LotusScript code:

```
Sub Click(Source As Button)

     Dim session As New NotesSession

     Dim db As NotesDatabase

     Set db = session.CurrentDatabase


     Dim collection As NotesDocumentCollection

     Set collection = db.AllDocuments


     Dim doc As NotesDocument

     Dim object As NotesEmbeddedObject

     Dim rtitem, handle As Variant

     Set doc = collection.GetNthDocument (7)
```

```
        Set rtitem = doc.GetFirstItem ("Object")


        If rtitem.Type = RICHTEXT Then
             Forall o In rtitem.EmbeddedObjects
                  Set object = rtitem.GetEmbeddedObject
                  (o.Name)
                  Set handle = object.Activate (True)
             End Forall
        End If
   End Sub
```

10. Save this form and close it.

Following is a description of the LotusScript code:

Declare collection as an object reference variable of the NotesDocumentCollection class.

```
Dim collection As NotesDocumentCollection
```

Assign collection as a reference variable to all documents in the current database.

```
Set collection = db.AllDocuments
```

Declare object as an object reference variable of class NotesEmbeddedObject.

```
Dim object As NotesEmbeddedObject
```

Assign doc as a reference variable to the seventh document in the collection of documents.

```
Set doc = collection.GetNthDocument (7)
```

Assign rtitem as a reference variable to the object item in the current document.

```
Set rtitem = doc.GetFirstItem ("Object")
```

The following statement tests the rtitem variable to see if it is rich text or not.

```
If rtitem.Type = RICHTEXT
```

This statement assigns object as a reference variable to an object name in the object rich text item.

```
Set object = rtitem.GetEmbeddedObject (o.Name)
```

The object's application is launched according to the object type in the object rich text item.

```
Set handle = object.Activate (True)
```

Next, compose a document using this form.

11. Choose Create - OLE 2 Document from the menu bar.
12. Click the Edit Embedded or Linked Object button.

When you use this LotusScript code to edit an embedded object, OLE 2 launches the object's application and displays the object's data in this application for a while. To display the data permanently and to manipulate the object data in its native application, you need to modify the above LotusScript code by adding some native LotusScript function for the object.

For example, if you edit a Word Pro document, you can add some Word Pro native function in your LotusScript code to edit or manipulate the Word Pro object's data. For more information on the functions that can be accessed by OLE 2 and Notes, refer to the product manuals provided with the object's application.

Deleting an Embedded or a Linked Object Using LotusScript

You can delete an embedded or a linked object in a Notes document using LotusScript. In the following example you will delete an embedded or linked object based on the object name that you enter.

Follow these steps to create LotusScript code for deleting an embedded or a linked object in a Notes document:

1. Open the OLE 2 Notes database. The view window displays.
2. On the navigator pane, select Design, then Forms, and open the OLE 2 Document form. The Form Design window displays.
3. Move to the end of the form and click the cursor under the Edit Embedded or Linked Object button.
4. Choose Create - Hotspot - Button from the menu bar and then type Delete Object in the Button label: box.
5. Next to Properties for: Button, drop down the list of available choices and select Text. Notice how the new button on the form now displays its title.

   Tip With the button properties still displayed, you can also highlight the new button by dragging the cursor from left to right. This automatically changes the button properties to text properties.

6. Click the fourth tab from the left, which allows you to specify security settings.
7. Select the Opened for reading check box. The Printed check box is selected automatically.
8. On the programming pane, select the Script radio button.
9. Click on the space between Sub Click(Source As Button) and End Sub and type the following LotusScript code:

```
Sub Click(Source As Button)
     Dim session As New NotesSession
     Dim db As NotesDatabase
     Set db = session.CurrentDatabase

     Dim view As NotesView
     Dim doc1, doc2 As NotesDocument
     Set view = db.GetView ("Access OLE 2")
     Set doc1 = view.GetDocumentByKey ("Embedding")
     Set doc2 = view.GetNextDocument (doc1)

     Dim object As NotesEmbeddedObject
     Dim rtitem As Variant
     Set rtitem = doc2.GetFirstItem ("Object")

     If rtitem.Type = RICHTEXT Then
          Forall o In rtitem.EmbeddedObjects
               Set object = rtitem.GetEmbeddedObject
               (o.Name)
               Call object.Remove
               Call doc2.Save (True, True)
          End Forall
     End If
End Sub
```

10. Save this form and close it.

Following is a description of the LotusScript code:

Declare view as an object reference variable of the NotesView class.

```
Dim view As NotesView
```

Assign view as a reference variable to the Access OLE 2 view.

```
Set view = db.GetView ("Access OLE 2")
```

Assign doc1 as a reference variable to the column value (Embedding) within the current view (Access OLE 2).

```
Set doc1 = view.GetDocumentByKey ("Embedding")
```

Assign doc2 as a reference variable to the document following doc1 in the current view.

```
Set doc2 = view.GetNextDocument (doc1)
```

Assign rtitem as a reference variable to the object item in the current document.

```
Set rtitem = doc2.GetFirstItem ("Object")
```

This method removes the object in the current rich text item.

```
Call object.Remove
```

Save this changed document without creating the response document.

```
Call doc2.Save (True, True)
```

Next, compose a document using this form.

11. Choose Create - OLE 2 Document from the menu bar.
12. Click the Delete Object button. The embedded object is removed from the document.

More Examples

In this example you will activate a linked object in a Notes document based on the object's name. The user enters the object name and the LotusScript code will search the object in all Notes documents in the current database. When the object is found, LotusScript will examine the object type. If the object is a linked object, LotusScript will display a prompt for activating the object's application. If the object is an embedded object, LotusScript will display a message indicating that this object cannot be activated using this LotusScript code.

Add a button to the OLE 2 Document form and type the following LotusScript code for this button:

```
Sub Click(Source As Button)
  Dim session As New NotesSession
  Dim db As NotesDatabase
  Set db = session.CurrentDatabase

  Dim collection As NotesDocumentCollection
  Set collection = db.AllDocuments

  Dim object As NotesEmbeddedObject
  Dim doc As NotesDocument
  Dim rtitem, handle As Variant
  Dim objName, className, _
  msg, msg1, msg2, msg3 As String
  Dim found, answer As Integer
  Dim boxType As Long

  objName = ""
  objName = Ucase (Trim (Inputbox ( _
  "Enter the object name : ", "Object Name", , 30, 25)))

  If Trim (objName) <> "" Then
     Set doc = collection.GetFirstDocument
     found = False

     Do
       Set rtitem = doc.GetFirstItem ("Object")
       If rtitem.Type = RICHTEXT Then
          Set object = rtitem.GetEmbeddedObject (objName)

          If Not (object Is Nothing) Then
```

```
                    Select Case object.Type

                    Case EMBED_OBJECTLINK
                      className = object.Class
                      msg1 = "This is a LINKED object " & Chr (10)
                      msg2 = "The object class name is " & _
                      className & Chr (10)
                      msg3 = "Click OK to activate this object"
                      boxType = MB_OKCANCEL + _
                      MB_ICONINFORMATION

                      answer = Messagebox (msg1 & msg2 & msg3, _
                      boxType, "Object Information")

                      If answer = 1 Then
                         handle = object.Activate (True)
                      Else
                         Exit Sub
                      End If
                      found = True

                    Case EMBED_OBJECT
                      className = object.Class
                      msg1 = "This is an EMBEDDED object " & Chr
(10)
                      msg2 = "The object class name is " & _
                      className & Chr (10)
                      msg3 = "You cannot edit this object" & _
                      " using this LotusScript code"
                      boxType = MB_OK + MB_ICONINFORMATION

                      Messagebox msg1 & msg2 & msg3, _
```

```
                boxType, "Object Information"
                found = True
             End Select
          End If   ' not object is nothing
       End If    ' rtitem = RICHTEXT
       Set doc = collection.GetNextDocument (doc)
    Loop  Until doc Is Nothing ' Or found

    If Not found Then
       boxType = MB_OK + MB_ICONSTOP
       msg = "Object " & objName & " not found !"
       Messagebox msg, boxType, "Object not found"
    End If
  Else
    Exit Sub
  End If   ' objName is not Null
End Sub
```

Following is a description of the LotusScript code:

Enter the object's name using the Input box function. This function will display "Enter the object name: " in the Inputbox window labeled Object name in the screen's coordinate 30, 25. The object name is converted into uppercase characters after removing the leading and trailing spaces.

```
objName = Ucase (Trim (Inputbox ( _
"Enter the object name : ", "Object Name", , 30, 25)))
```

Verify if the objName variable is empty by removing the leading and trailing spaces.

```
If Trim (objName) <> "" Then
```

Assign the doc as a reference variable to the first document in the collection of documents.

```
Set doc = collection.GetFirstDocument
```

Assign the rtitem as a reference variable to the Object item in the first document.

```
Set rtitem = doc.GetFirstItem ("Object")
```

This statement assigns object as a reference variable to the object name in the object rich text item.

```
Set object = rtitem.GetEmbeddedObject (objName)
```

Verify if the object name in the current document is equal to the object name entered by the user.

```
If Not object Is Nothing Then
```

Verify if the object type is a linked object.

```
Case EMBED_OBJECTLINK
```

Set the className variable with the object's class name.

```
className = object.Class
```

Set the boxType variable with OKCANCEL button and icon information values.

```
boxType = MB_OKCANCEL + _
MB_ICONINFORMATION
```

Display the messages from the in msg1, msg2, msg3 variables in a message box labeled "Object Information" containing an OK and a Cancel button. This message box function returns a value when the button is clicked.

```
answer = Messagebox (msg1 & msg2 & msg3, _
boxType, "Object Information")
```

Verify if the clicked button is the OK button.

```
If answer = 1 Then
```

Load the object, through OLE, in the Notes document by displaying the user interface of the server application.

```
handle = object.Activate (True)
```

Assign the doc as a reference variable to the next document in the collection of documents.

```
Set doc = collection.GetNextDocument (doc)
```

Verify the value of doc and the found variables. If one of the variable values is true, the loop is terminated.

```
Until (doc Is Nothing ) Or found
```

Exit from this click procedure.

```
Exit Sub
```

Appendix A
HiTest and LotusScript Notes Classes: Comparing Functions

The following table lists equivalent functions in HiTest and LotusScript. Not all the properties and methods of LotusScript Notes classes are listed here. For more detailed information on LotusScript Notes classes, refer to Chapter 3.

The LotusScript Notes classes, properties and methods shown in *italics* will be available in Lotus Notes Release 4.5.

Note that not all the LotusScript Notes classes, methods and properties can be used in Visual Basic through OLE automation.

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
ACL			
	HTACLAdd	NotesACLEntry	Name, CanCreateDocuments, CanDeleteDocuments, CanCreatePersonalAgent, CanCreatePersonalFolder
	HTACLDelete	NotesACLEntry	Remove
	HTACLGetRole	NotesACLEntry	Roles, IsRoleEnabled
	HTACLGetRoleName	NotesACLEntry	Roles
	HTACLList	NotesACLEntry	GetFirstEntry, GetNextEntry
	HTACLLookup	NotesACLEntry	GetEntry
	HTACLReset		
	HTACLSetRole	NotesACLEntry	AddRole, DeleteRole, EnableRole, DisableRole
	HTACLSetRoleName	NotesACLEntry	RenameRole
	HTACLUpdate	NotesACLEntry	Save

Continued

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
Agent			
	HTAgentCopy		
	HTAgentCreate		
	HTAgentDelete	NotesAgent	Remove
	HTAgentExecute	NotesAgent	Run
	HTAgentFetch	NotesAgent	Name, Owner, ServerName, Query, Comment
	HTAgentList	NotesAgent	Name, IsPublic
	HTAgentLocateByName	NotesDatabase	***GetAgent***
Attachment			
	HTAttachmentAttach		
	HTAttachmentDelete		
	HTAttachmentExtract	NotesDocument	GetAttachment
	HTAttachmentList	RichTextItem	FindEmbeddedObject
CDRecord			
	HTCDRecordDelete		
	HTCDRecordFetchXxxx	RichTextItem	GetFormattedText
	HTCDRecordGetCount		
	HTCDRecordGetInfo		
	HTCDRecordInsertXxxx	RichTextItem	AppendText, AddTab, AddNewline, AppendDocLink, EmbedObject, AppendRTItem
	HTCDRecordList	RichTextItem	FindEmbeddedObject
	HTCDRecordMakeDoclink	RichTextItem	AppendDocLink
	HTCDRecordUpdateXxxx		
Column			
	HTColumnFetch	NotesViewColumn	Title, ItemName, Formula, Position, IsSorted, IsCategory, IsHidden, IsResponse
	HTColumnGetCount	NotesViewColumn	
	HTColumnList	NotesViewColumn	Title, ItemName, Formula, Position, IsSorted, IsCategory, IsHidden, IsResponse

Continued

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
Composite			
	HTCompositeClose		
	HTCompositeConvertDocument	NotesDocument	RenderToRTItem
	HTCompositeCopy		
	HTCompositeCopySubset		
	HTCompositeCreate	NotesDocument	New, AppendRTFFile
		NotesDocument	CreateRichTextItem
	HTCompositeExport		
	HTCompositeGetFont		
	HTCompositeGetProperty		
	HTCompositeImport		
	HTCompositeListExport		
	HTCompositeListImport		
	HTCompositeListText		
	HTCompositeMerge		
	HTCompositeOpen		
	HTCompositePutFont		
Database			
	HTDatabaseClose	NotesDatabase	Close
	HTDatabaseCompact	NotesDatabase	Compact
	HTDatabaseCopy	NotesDatabase	CreateCopy, CreateReplica
	HTDatabaseCreate	NotesDatabase	New, Create, CreateFromTemplate
	HTDatabaseCreateFTIndex	NotesDatabase	UpdateFTIndex
	HTDatabaseDelete	NotesDatabase	Remove
	HTDatabaseDeleteFTIndex		
	HTDatabaseGetProperty	NotesDatabase	IsFTIndexed, Created, LastFTIndexed, TemplateName, DesignTemplateName, FileName, FilePath, LastModified, ReplicaID, Server, Size, PercentUsed, Title, Categories
	HTDatabaseList	DbDirectory	GetFirstDatabase, GetNextDatabase
	HTDatabaseListCatalog		
	HTDatabaseLocateByReplicaid	NotesDatabase	OpenByReplicaID

Continued

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
	HTDatabaseLocateByTitle		
	HTDatabaseOpen	NotesDatabase	Open
	HTDatabaseSetProperty	NotesDatabase	Title, Categories, IsOpen, Managers, Views, Agents, Parent, IsPublicAddressBook , IsPrivateAddressBook, ACL , CurrentAccessLevel, AllDocuments, Size, UnprocessedDocuments
Datetime			
	HTDatetimeAdjust	NotesDateTime	Adjust Second, AdjustMinute, Adjust Hour, AdjustDay, Adjust Month, Adjust Year
	HTDatetimeCompare	NotesDateTime	***TimeDifference***
	HTDatetimeConstant	NotesDateTime	SetAnyTime, Set Any Date
	HTDatetimeCreate	NotesDateTime	New
	HTDatetimeFetch	NotesDateTime	TimeZone, IsDT
	HTDatetimeGetCurrent	NotesDateTime	SetNow
	HTDatetimeGetDiff	NotesDateTime	TimeDifference
	HTDatetimeGetProperty		
Document			
	HTDocumentClose		
	HTDocumentCopy	NotesDocument	CopyToDatabase
	HTDocumentCreate	NotesDocument	New
	HTDocumentDelete	NotesDocument	Remove
	HTDocumentEncrypt	NotesDocument	Encrypt
	HTDocumentExecute		
	HTDocumentGetProperty	NotesDocument	Created, LastModified, LastAccessed, IsResponse,ParentDocumentUNID, NoteID, UniversalID, EncryptOnSend, SignOnSend, HasEmbedded
	HTDocumentLocateByClass		
	HTDocumentLocateByUnid		
	HTDocumentOpen		
	HTDocumentSave	NotesDocument	Save

Continued

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
	HTDocumentSetProperty	NotesDocument	EncryptOnSend, SignOnSend, MakeResponse
	HTDocumentValidate		
Error			
	HTErrorFetch		
	HTErrorGetInfo		
Field			
	HTFieldFetch		
	HTFieldGetCount		
	HTFieldList		
Fontstyle			
	Description		
Form			
	HTFormCopy		
	HTFormCreate		
	HTFormDelete	***NotesForm***	***Remove***
	HTFormFetch	***NotesForm***	***Aliases, Field, FormUsers, IsSubform, Name, ProtectReaders, Readers, ProtectUsers***
	HTFormList		
	HTFormLocateByName	NotesDatabase	***GetForm***
Format			
	Description		
Index			
	HTIndexClose		
	HTIndexCopyHierarchy		
	HTIndexDeleteHierarchy		
	HTIndexFTSearch	NotesView	FTSearch, Clear
	HTIndexGetCount	NotesDocumentCollection	Count
	HTIndexGetPosition		
	HTIndexGetProperty	NotesDocument	FTSearchScore
		NotesDocumentCollection	IsSorted

Continued

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
	HTIndexGetViewPosition		
	HTIndexNavigate	NotesDocumentCollection	GetLastDocument, GetNextDocument, GetPrevDocument, GetNextSibling, GetPrevSibling, GetChild, GetParent
	HTIndexOpen		
	HTIndexOpenView		
	HTIndexRefresh	NotesDocumentCollection	Refresh
	HTIndexSearch		
	HTIndexSetPosition	NotesDocumentCollection	GetFirstDocument, GetNthDocument
		NotesDocumentCollection	GetFirstDocument, GetNthDocument
	HTIndexSetProperty		
	HTIndexSetViewPosition		
		NotesDocumentCollection	GetDocumentByKey
Item			
	HTItemAppendTextList	NotesDocument	AppendToTextList
	HTItemCopy	NotesDocument	CopyItem
		NotesItem	CopyItemToDocument
	HTItemDelete	NotesItem	RemoveItem
	HTItemDelete	NotesItem	Remove
	HTItemFetch	NotesItem	GetItemValue, HasItem, GetItemValue, Signer, IsSigned, Authors
		NotesItem	Values, DateTimeValue
	HTItemGetCount		
	HTItemGetInfo		
	HTItemGetLength		
	HTItemList	NotesItem	GetFirstItem, GetNextItem, Items
		NotesItem	Name, ValueLength, Type, IsNames, IsReaders, IsAuthors, IsSigned, IsEncrypted, IsSummary, IsProtected
	HTItemPut	NotesItem	ReplaceItem, AppendItemValue, Sign, PutInFolder, RemoveFromFolder
		NotesItem	New
		NotesItem	Contains
		NotesItem	Abstract

Continued

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
Mail			
	HTMailCreate	NotesDocument	CreateReplyMessage, SaveMessageOnSend,
	HTMailSend	NotesDocument	Send
Server			
	HTServerExecute		
	HTServerGetProperty		
	HTServerList		
	HTServerReplicate		
	HTServerSetPort		
Session			
	HTConvert		
	HTConvertGetLength		
	HTGetAPILibraryVersion		
	HTGetEnvironmentString	NotesSession	GetEnvironmentString
	HTGetFormat		
	HTGetInternational		
	HTGetProperty	NotesSession	UserName, CommonUserName, IsOnServer, NotesVersion
	HTInit	NotesSession	New
	HTSetEnvironmentString	NotesSession	SetEnvironmentVar
	HTSetFormat		
	HTSetInternational		
	HTSetProperty		
	HTStringFetch		
	HTTerm	NotesSession	Close
	HTTranslate		
TextList			
	HTTextListFetch		
	HTTextListGetCount		
	HTTextListGetLength		

Continued

HiTest Class	*HiTest Functions*	*LotusScript Notes Classes*	*LotusScript Notes Classes: Properties and Methods*
User			
	HTUserRegisterCertifier		
	HTUserRegisterServer		
	HTUserRegisterWorkstation		
	HTUserSetCertifier		
View			
	HTViewCopy		
	HTViewDelete	NotesView	Remove
	HTViewFetch	NotesView	Name
	HTViewList		
	HTViewLocateByName	NotesDatabase	GetView
Viewcell			
	HTViewcellFetch	NotesView	Columns
	HTViewcellGetLength		

Appendix B
Special Notices

This publication is intended to help Visual Basic programmers get acquainted with LotusScript to enable them to migrate their HiTest applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by Lotus Notes.

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, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. 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 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.

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:

IBM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

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

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

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Other trademarks are trademarks of their respective companies.

The following are trademarks of Lotus Development Corporation in the United States and/or other countries.

DataLens
InterNotes
InterNotes Web Publisher
Lotus
Lotus Notes Reporter
Lotus Notes
Lotus Notes ViP
Lotus @SQL
LotusScript
Notes
Notes HiTest
Notes ViP
Notes Mail
NotesPump
NotesSQL
Notes/FX
Phone Notes
Phone Notes Mobile Mail
SmartIcons
Video Notes
Word Pro

Appendix C
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”.

- *Developing Applications with Lotus Notes Release 4*, SG24-4618, Lotus Part No. 12394

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com/redbooks.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- PUBORDER - to order hardcopies in the United States.
- GOPHER link to the Internet - type gopher.wtscpok.itso.ibm.com.
- Tools disks

  To get LIST3820s of redbooks, type one of the following commands:

  tools sendto ehone4 tools2 redprint get sg24xxxx package

  tools sendto canvm2 tools redprint get sg24xxxx package (Canadian users only)

Note The current redbook *LotusScript for Visual Basic Programmers* is not available as a LIST3820 or in BookManager format.

To get lists of redbooks:

tools sendto wtscpok tools redbooks get redbooks catalog

tools sendto usdist mkttools mkttools get itsocat txt

tools sendto usdist mkttools mkttools get listserv package

To register for information on workshops, residencies, and redbooks:

tools sendto wtscpok tools zdisk get itsoregi 1996

For a list of product area specialists in the ITSO:

tools sendto wtscpok tools zdisk get orgcard package

- Redbooks Home Page on the World Wide Web

  http://w3.itso.ibm.com/redbooks/redboooks.html

Note The current redbook *LotusScript for Visual Basic Programmers* is also available in HTML format and in Adobe Acrobat format on the World Wide Web. The URL is http://www.lotus.com/devtools. In addition, the code samples provided throughout the book are available for your use on http://www.lotus.com/redbook.

- IBM Direct Publications Catalog on the World Wide Web

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.
- ITSO4USA category on INEWS
- Online - send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL.
- Internet Listserver

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBM Mail	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	bookshop at dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** - send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29554	Calgary, Alberta T2P 3N5	DK-3450 Allerod
Raleigh, NC 27626-0570	Canada	Denmark

- **Fax** - send orders to:

United States (toll free)	1-800-445-9269
Canada (toll free)	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:
  Index #4421 Abstracts of new redbooks
  Index #4422 IBM redbooks
  Index #4420 Redbooks for last six months

- **Direct Services** — send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://ww.redbooks.ibm.com/redbooks
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

  The current redbook *LotusScript for Visual Basic Programmers* is also available in HTML format and in Adobe Acrobat format on the World Wide Web. The URL is http://www.lotus.com/devtools. In addition, the code samples provided throughout the book are available for your use on http://www.lotus.com/redbook.

- **Internet Listserver**
  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service send an e-mail note to annouce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Index

T

U

V

W