Chapter 17 – Communicating with Other Applications

# Objectives:

* Understanding Microsoft’s Evolving Object Model
* Using Automation
* Using the Shell Command
* Using Sendkeys
* Other Related Commands
* The Environment Class
* The Application Class
* Calling APIs
* The Clipboard

This chapter will examine how to communicate with other applications. Part of what has made Visual Basic such a successful programming language is its ability to harness the functionality that's present in other applications. This chapter will look at working with other application's object models.

In addition, this chapter will cover the various commands related to starting up and executing other programs from within Visual Basic. VB also provides two classes, the Environment class and the Application class, to assist in this interprocess communication.

Finally, the chapter will end with a discussion of how to perform low level operating system (O/S) API calls. The Application Programmer Interface greatly extends the capabilities of VB by allowing a developer access to the functionality that is built into the Windows O/S.

# Microsoft’s Evolving Object Model

When Microsoft Windows 1.0 was released in 1985, very few people cared. In fact, Microsoft honestly looked like it just threw something together to compete with the Macintosh's graphical user interface. At that time, the Mac was sleek and sophisticated – Windows, well, it was slow, clunky and ran off of 5 ¼" floppy disks. Most PCs at the time had a high-resolution mode of 640 x 480 with 2 colors (black and whatever color the monitor was, usually green or amber). You can see why many users were happy with good old DOS and stayed put.

With the advent of Windows version 2.0, Microsoft created a much more solid system that ran on top of DOS, had multiple colors and lived on a user's hard drive. One of the more dramatic additions to Windows was the fact that Microsoft provided the capability for two applications to communicate with one another through DDE, or Dynamic Data Exchange. Essentially one application was set up to be the DDE Server and another application could serve as a DDE Client. The two applications could perform some simply data exchanges that were text based. However, this launched the start of Windows’ cut and copy editing between applications, which up to this point was not a common thing.

Windows 3.0 introduced a more mature technology, OLE, or Object Linking and Embedding, where one application could now host a “document” that had been created by another application. For example, you could embed an Excel spreadsheet into Word, or record a sound and embed the sound recording in Excel. This provided some great functionality for users depending on how one used OLE. It was possible to paste in (embed) a copy of a foreign document into a different application, and if the user wanted to change anything in that copy, all he or she had to do was double-click on it. The original application that created the foreign document would start up, allowing for changes to be made. The other type of application communication in Windows version 3.0 was linking, in which a user would perform a "Paste Special" command to create a link to a foreign document from the host document.

You may be wondering that the difference is between linking and embedding. In linking, there is only one copy of a document that ever exists. Any documents that have a link to the foreign document simply point to that single copy. If the user changes the foreign document in any way, the single copy is altered. Therefore, if the user starts up the application that natively created the foreign document and modifies the document, the next time that a document which has a link to the foreign document is opened, the linked document will automatically show the new, changed version.

Embedding documents makes multiple copies of the same document. If you embed an Excel spreadsheet in Word, essentially another copy of the spreadsheet is made: you have the original spreadsheet and you also have a copy in Word. Unfortunately, the two spreadsheets do not automatically synchronize if you make changes to one or the other. Therefore, if you change the original spreadsheet, you would have to re-embed it in Word to show the updated changes. This is why many people preferred linking over embedding back in the Windows 3.0 days.

Windows 3.1 (but it didn’t really take off until Windows 95) introduced the world to COM, or the Component Object Model (there’s some disagreement as to whether the C in COM stands for Common or Component). COM is a binary standard that defines interfaces to components so that any programming language can access and use COM objects without worrying about how the COM object was created. Microsoft has changed many of their applications including VB, Internet Explorer and the Microsoft Office Suite to be built around large COM models so that the applications and their components are largely reusable by other applications. Microsoft continued along this line of shared object usage with COM+ and DCOM, where DCOM refers to Distributed COM. The DCOM model allows for objects to be accessible anywhere on a network, whereas the original COM model required any objects that were to be used to be loaded on the user’s local computer.

The .NET architecture brought along an entirely different object model than that those that predated .NET technologies. In .NET, COM controls are to a degree passé, while their new counterparts are known as .NET framework components. Here's a diagram that shows how the .NET architecture is comprised:

Visual Basic C++ C# F# …

Common Language Specification (CLS)

ASP.NET Windows Forms

Data and XML

Base Class Library (BCL)

Common Language Runtime (CLR)

Windows API COM+ Services

Now this isn’t to say that you can't use COM components in .NET. Microsoft just doesn't make any guarantees that the COM components will work. From personal experience, I can tell you that many COM components work just fine. There are a couple I've run across though, that do not work right anymore under .NET. We will examine how to create .NET framework components in a couple chapters from now.

# Using Automation

As we already mentioned, COM (DCOM and COM+) is Microsoft’s preferred way of providing application and some component interoperability. As of this writing of these notes, this still hasn't changed – Microsoft Office 365 still has a COM object model available for use by other applications.

The access to the COM models that make up applications and the ability to programmatically manipulate them is known as automation in Visual Basic. Many modern applications have two things that VB programmers like:

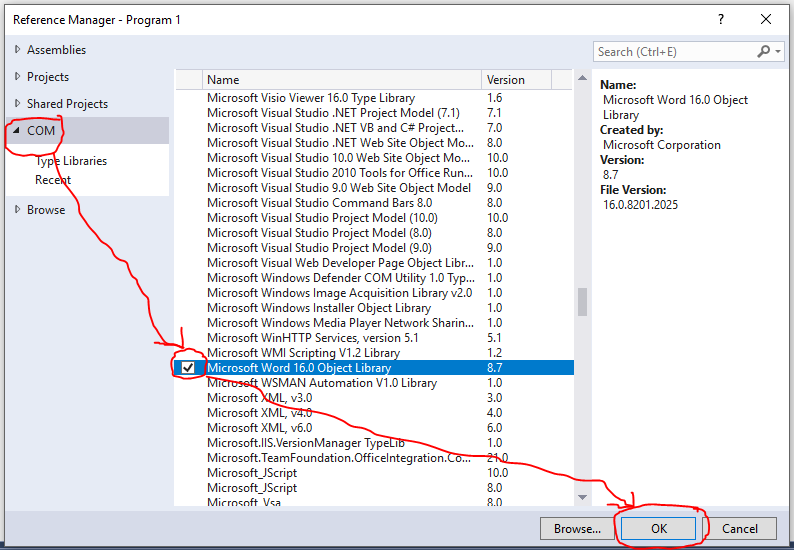
1. Most Microsoft applications have the ability allowing users to create macros or write programs in VBA (Visual Basic for Applications). This allows a developer to leverage their VB skills to add additional functionality to existing applications.
2. Most widely-used applications have large COM models built into them. This allows the application to be initialized and manipulated through code in Visual Basic. For example, instead of writing code in a VB application to generate mailing labels, Microsoft Word could be called to handle the work instead.

Some examples of applications that we can access via COM automation include Word, Excel, Access, PowerPoint, Outlook, Internet Explorer, Visio, Adobe Acrobat and Photoshop, just to name a few. You'll notice that the list is not just Microsoft exclusive – many companies have bought into the COM view because they developed their products using Microsoft’s development languages and have exported the object models of their applications for developers to tie into.

Now that we've gone through the mechanics of what automation is, let’s see an example of it. We will write a VB application that starts up Microsoft Word and writes some text to a new Word document. Obviously to execute this application, you will need to have Microsoft Word available on the computer where you are writing and testing your VB code – in fact, you will need Microsoft Office loaded to execute most of the applications in this chapter.

To get started with automation, all that we need to do to use Word from VB is to provide a reference to the Microsoft Word Object Library. In case you forgot how to do that, we add references through the Project🡪Add Reference menu selection. Next click on the COM tab and scroll down in the listbox until you find the Microsoft Word Object Library and then check the checkmark out in front of the library name. If you have multiple Microsoft Word Object Libraries, then select the newest version you have available to you.

Here’s a screenshot of the COM libraries I have available on my system. Note that I am on the COM tab, checked the Microsoft Word Object Library and then click the OK button to add the reference to my project:



Our application has a single Command button placed on it (using the default name). Here’s the code for the application:

'Chapter 17 - Program 1

'Even though we added a reference to the DLL, we want

'to be able to work with the high level pieces without

'writing out the class names in long form

Imports Microsoft.Office.Interop

Public Class Form1

'Create a variable that will eventually point to Word

Dim aWordDoc As Word.Application

Private Sub Button1\_Click(sender As Object, e As EventArgs)

Handles Button1.Click

'Create a new instance of Word from within our application

'This will actually start Word up.

aWordDoc = New Word.Application()

'Set the Visible property to true if you want to see Word

'running

aWordDoc.Visible = True

'Add a new document

aWordDoc.Documents.Add()

'Set the font size to 20

aWordDoc.Selection.Font.Size = 20

'Set the font style to italic

aWordDoc.Selection.Font.Italic = True

'Type some information to the document

aWordDoc.Selection.TypeText("Hello -- this data is" \_

& " coming to you live from VB")

'We could save the Word document with this command:

'aWordDoc.Documents(1).SaveAs(“c:\mydoc.doc”)

'We should then quit the Word application, but if I quit,

'you wouldn’t see anything, so I don't stop Word here.

'Here's the way we would make Word terminate:

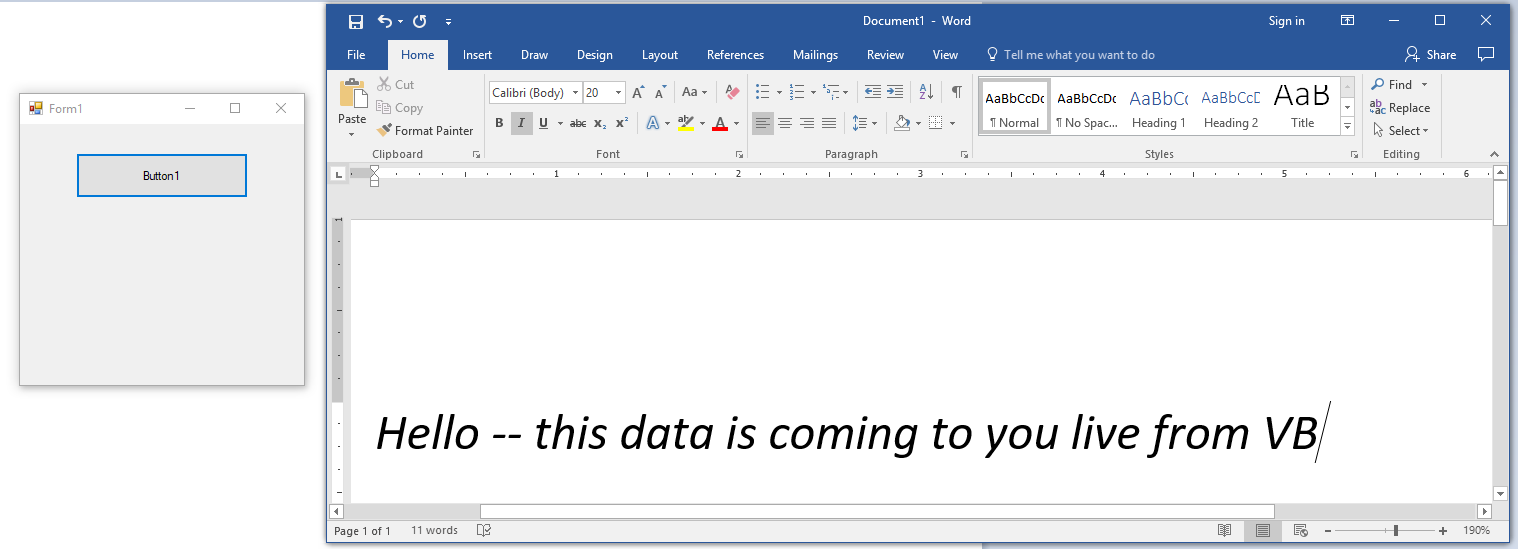
'aWordDoc.Quit

'set aWordDoc = Nothing

End Sub

End Class

Here’s a screenshot of the application running after we’ve clicked on the button:



That was really quite simple to do – the toughest part is navigating the referenced object model to get the application to perform what you want from it. If you need more details about the various object models, consult the MSDN documentation – the Microsoft Office object models are documented there. If it’s a non-Microsoft application, you may need to contact the vendor for COM specific information on how their models are set up.

Let’s write another application that’s easy to work with: Excel. Remember that in this case, you will need to add a reference to the Microsoft Excel Object Library!

Here’s the code for our Excel automation program. We'll do a couple of different things in this application. We will check to see if Excel is already running in memory before we simply start another copy up. We will also close the Excel spreadsheet down when we are done with our program. You'll see that Excel will ask you if you want to save the changes to the worksheet that we created before it closes, just like what would expect if you were using Excel yourself.

The application has one Command button on it as before. Here’s the code:

'Chapter 17 - Program 2

Imports Microsoft.Office.Interop

Public Class Form1

Private Sub Button1\_Click(sender As Object, e As EventArgs)

Handles Button1.Click

Dim CheckExcel As Object

Dim anExcelDoc As Excel.Application

Dim intLoop As Integer

'Check to see if Excel is already loaded in memory

Try

CheckExcel = GetObject(, "Excel.Application")

Catch Ex As Exception

'Excel was not running, so we got an error

End Try

If CheckExcel Is Nothing Then

'Create a new instance of Excel

anExcelDoc = New Excel.Application()

anExcelDoc.Visible = True

Else

anExcelDoc = CheckExcel

anExcelDoc.Visible = True

End If

'Add a new workbook and a new sheet

anExcelDoc.Workbooks.Add()

anExcelDoc.Sheets.Add()

'Put some data on the sheet

For intLoop = 1 To 5

anExcelDoc.Cells(intLoop, 1) = 100 \* intLoop

Next

MessageBox.Show("All data has been filled in")

'Throw some statistical information on the sheet

anExcelDoc.Cells(6, 1) = "Ave:"

anExcelDoc.Cells(6, 2) = "=average(a1..a5)"

anExcelDoc.Cells(7, 1) = "Min:"

anExcelDoc.Cells(7, 2) = "=min(a1..a5)"

anExcelDoc.Cells(8, 1) = "Max:"

anExcelDoc.Cells(8, 2) = "=max(a1..a5)"

anExcelDoc.Cells(9, 1) = "Stdev"

anExcelDoc.Cells(9, 2) = "=stdev(a1..a5)"

MessageBox.Show("All statistical data is now available")

'Clean things up

anExcelDoc.Quit()

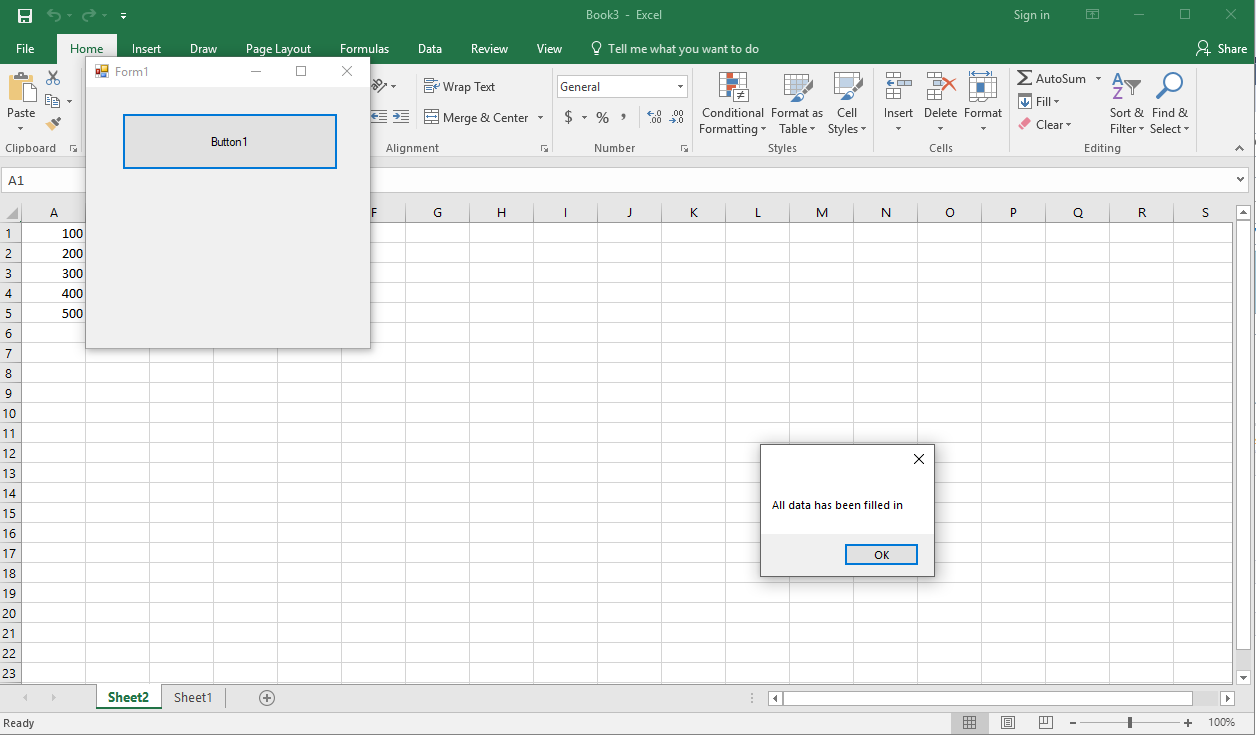
anExcelDoc = Nothing

End Sub

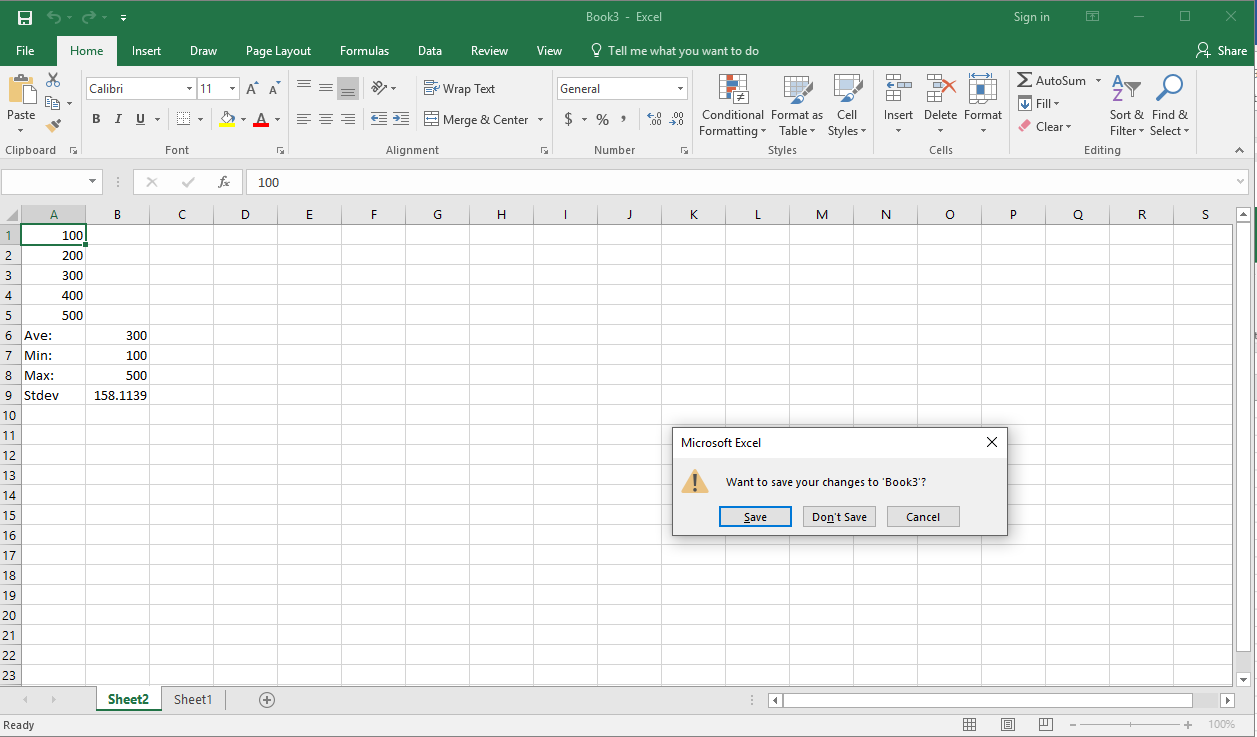
End Class

You will need to hop back and forth between your application and the Excel sheet to see the message boxes as they pop up. Remember that the message boxes are triggered by your application and not by Excel. These were put in to pause the VB program and help you visualize what’s going on.

Here’s a screenshot of the Excel application in progress. Notice that the VB application is the small "Form1" window floating over top of Excel. The message box that has appeared saying “All data has been filled in” also belongs to the VB application, while the remainder of the screen is Excel. The five initial data values have been added to the spreadsheet:



After clicking OK on the message box, we send some formulas to Excel and then we try to shut Excel down from VB. Excel then prompts the user about saving the spreadsheet:



# Using the Shell Command

The Shell method can be used to run another application that's external to your VB program without using COM. Think of the Shell command as if you pointed and clicked to launch an application. If the second application successfully starts up, the application’s process ID number will be returned. The Shell command is located in the Microsoft.VisualBasic.Interaction class.

The syntax for the Shell command is Shell *pathname [,windowstyle] [,wait]*, where the *windowstyle* is a constant from the AppWinStyle namespace, and *wait* refers to a Boolean value that indicates whether or not the Shell should wait for the named program to complete before closing the Shell window that the program runs in.

Here are the various window styles that you have available to you. Remember to preface any style with AppWinStyle. before the style name:

|  |  |
| --- | --- |
| *Style* | *Purpose* |
| Hide | The new application window is hidden |
| MaximizedFocus | The new application is shown maximized |
| MinimizedFocus | The new application is shown minimized |
| MinimizedNoFocus | The new application is shown minimized but with no focus |
| NormalFocus | The new application is shown in its original size and position |
| NormalNoFocus | The new application is shown normal size but with no focus |

Here’s the code for a sample application that will use the Shell method to execute a DOS command window, the FTP program and Notepad. After each program is started, a message box will appear that provides the process ID number of each program. There is a single Command button with the default name added to the form:

'Chapter 17 - Program 3

Public Class Form1

Private Sub Button1\_Click(sender As Object, e As EventArgs)

Handles Button1.Click

'These variables will hold the process ID numbers of

'the three programs that we are going to run.

Dim intResult1 As Integer

Dim intResult2 As Integer

Dim intResult3 As Integer

'Start up a command prompt

intResult1 = Shell("cmd.exe", AppWinStyle.MaximizedFocus)

MessageBox.Show("cmd.exe = " & intResult1)

'Start up FTP

intResult2 = Shell("ftp.exe", AppWinStyle.NormalFocus)

MessageBox.Show("ftp.exe = " & intResult2)

'Start up Notepad

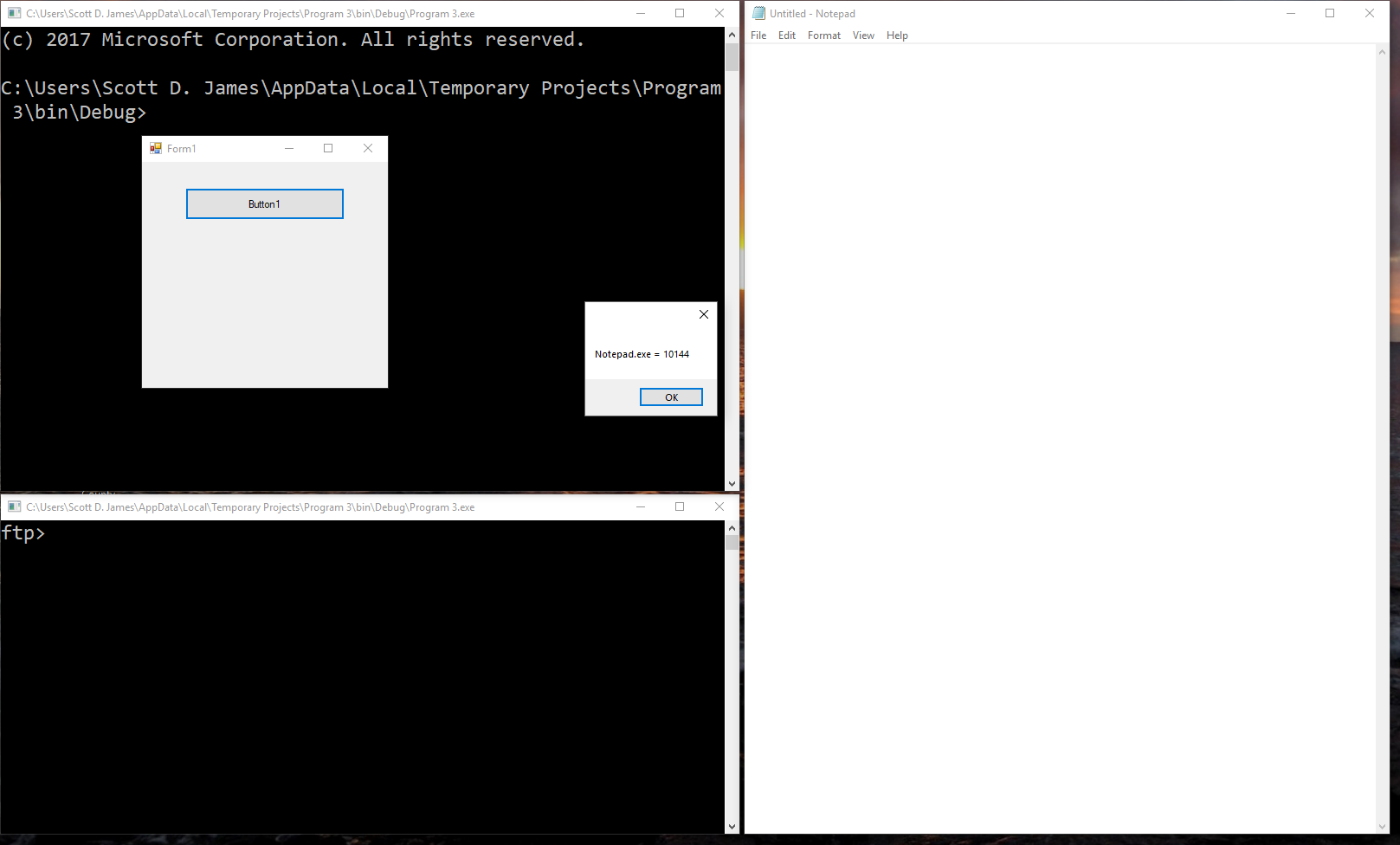
intResult3 = Shell("notepad.exe", AppWinStyle.NormalFocus)

MessageBox.Show("Notepad.exe = " & intResult2)

End Sub

End Class

Here’s a screenshot of our application after all three shell commands have executed. We can see the process ID of Notepad.exe still on the screen:



The process ID number that each shelled program receives is provided by the operating system. The number will not be the same each time you Shell execute that application.

# Using SendKeys

Another class that is handy to work with when using the Shell method to start programs is the SendKeys class. SendKeys is used to simulate keyboard entry to an application. The SendKeys.Send method sends its keystrokes to the application that has the focus.

The syntax for the most commonly used methods in the SendKeys class includes SendKeys.Send(*chars to send*) or SendKeys.SendWait(*chars to send*). If the SendWait method is used, SendKeys will wait until the process that SendWait sent the keystrokes to, to have handled the keystrokes before passing control back to the VB procedure that performed the SendWait.

Here’s an example program that launches the Windows calculator program, and then performs the sequence to add 5 + 3 using the Send method. The application has a single command button on it with the default name:

'Chapter 17 - Program 4

Public Class Form1

Private Sub Button1\_Click(sender As Object, e As EventArgs)

Handles Button1.Click

'Execute the Calculator program via the Shell command

Shell("calc.exe", vbMaximizedFocus)

'Wait 1 second before proceeding to

'give the calc program enough time to load up

'and become ready for input.

'

'Hmmm.... Haven't seen this class or method before.

'I do have a sneaking suspicion that you will see

'it again though...

System.Threading.Thread.Sleep(1000)

'Start sending keys to the calc program

'Clear the calculator

SendKeys.SendWait("C")

'Send the value 5

SendKeys.Send("5")

'Send the + key, but we have to use a

'special value for + since the symbol itself has

'a special meaning in SendKeys -- see the MSDN

'help for more information

SendKeys.Send("{ADD}")

'Now send the second value to add, 3

SendKeys.Send("3")

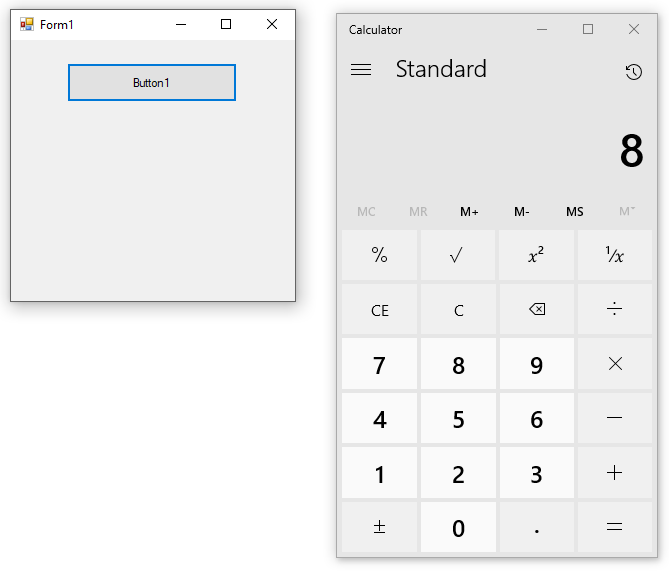
'Finally, send the = to do the math

SendKeys.Send("=")

End Sub

End Class

Here’s a screenshot with the results of the addition shown in the calculator:



As you learned through the code of the above application, there are some symbols that have special meanings within SendKeys. Here is a partial list of the special characters that can be sent in SendKeys:

|  |
| --- |
| *Special Characters* |
| + placed before some other character means Shift |
| ^ placed before some other character means Ctrl |
| % placed before some other character means Alt |
| ~ means Enter |
| {Backspace}, {Bs} or {Bksp} |
| {Break} |
| {CapsLock} |
| {Del},{Delete} |
| {Down} |
| {End} |
| {Enter} |
| {Esc} |
| {F1} … {F12} |
| {Help} |
| {Home} |
| {Ins},{Insert} |
| {Left} |
| {Numlock} |
| {Pgdn} |
| {Pgup} |
| {Right} |
| {Scrolllock} |
| {Tab} |
| {Up} |

# Other Application Related Commands

This chapter has focused on examining how to interact with object models in other applications through automation and system level entities such as remote program execution and communication using Shell and SendKeys. There are several other methods in the Microsoft.VisualBasic.Interaction class that you might find useful when performing this type of work. Remember to preface each method with the namespace or import it.

Some of these commands are throwbacks allowing compatibility with pre-.NET versions of VB. I’ll point out where there are newer ways to handle some of these commands through .NET:

## AppActivate

Switches focus to the specified application. The syntax is AppActivate(*"Window Title"*) or AppActivate(*processIDNumber*). The first syntactic example activates an application based on the application's window title while the second example activates an application based on the application's process ID.

## Beep

Makes the system speaker beep. The syntax is simply Beep.

## CallByName

Sets and gets properties and invokes methods at runtime by providing arguments to the CallByName method. Here are the different forms of syntax for the CallByName method:

*To set a property:*

CallByName(*object or control*, *property*, CallType.Set*, value to*

*place in property*)

*To get a property:*

CallByName(*object or control*, *property*, CallType.Get)

*To invoke a method:*

CallByName(*object or control*, *method*, CallType.Method)

Here's an example of setting the caption on a command button and then making the command button hide:

'We have to import the following namespace

Imports Microsoft.VisualBasic.Interaction

'.

'.

'.

' Set the Text property on a command button

CallByName(Command1, "Text", CallType.Set, "Push Me")

' Retrieve the text property

MessageBox.Show(CallByName(Command1, "Text", CallType.Get))

' Make the command button hide itself

CallByName(Command1, "Hide", CallType.Method)

You may be wondering what the usefulness of this command is, since we can directly manipulate a Command button’s properties and methods at runtime. Not all objects will have this capability, therefore CallByName provides us a workaround when that situation arises.

## Command

Returns the argument portion of the command line that was used to launch VB or a VB developed application. The syntax for this is simply Command and a string will be returned. .NET has a newer method of handling this in the Environment class, which we will talk about below.

## CreateObject

Creates and returns a reference to a COM object. This isn’t quite so useful in VB, but it is still widely used in VBScript and VBA.

## Environ

Returns the operating system environment variables as strings. This is a compatibility feature from pre-.NET; you should now use the .NET Environment class which will be discussed below.

## Registry Settings

The following commands are related to manipulation of the Registry, which is the storage mechanism that Windows uses to keep information about hardware, applications, device drivers and so forth. If you mess up Registry information, you can brick your computer to the point that you would have to reload it.

For several generations of Visual Studio, this was the recommended way of storing your own application’s settings. Fortunately, Microsoft has a newer method which is much safer and doesn’t involve the Registry. We will be looking at saving application settings using the newer technique in a future chapter.

Because you may be porting older .NET applications, you at least need to know what the Registry manipulation commands look like. Today the Registry still exists, but is primarily used only by Windows to store operating system related information.

### DeleteSetting

Deletes a registry section or key setting. The syntax is DeleteSetting (*ApplicationName*, [*Section*], [*Key*]).

### GetSetting

Gets a registry key setting. The syntax is GetSetting (*ApplicationName*, [*Section*], [*Key*]).

### GetAllSettings

Returns all settings related to an application's registry settings in a list. The syntax is GetAllSettings (*ApplicationName*).

### SaveSetting

Saves a registry setting to the registry. The syntax is SaveSetting (*ApplicationName*, *Section*, *Key*, *Setting*).

# Using the Environment Class

The *Environment* class can be used to both access and set information that is related to the operating system’s runtime. It is much more powerful than the pre-.NET Environ function which simply returned environment strings.

Let's look at an application that prints out information about the current directory, the command line, the operating system version, the computer's name, the current path and all of the environment variables that have been set. This is a Windows forms application with all the processing happening in the Load event handler:

'Chapter 17 - Program 5

Public Class Form1

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Dim strMessage As String

Dim myEntry As DictionaryEntry

'Get basic information from the environment

strMessage = "Current directory is " & Environment.CurrentDirectory &

vbCrLf

strMessage &= "Command line launched " & Environment.CommandLine & vbCrLf

strMessage &= "O/S Platform " & Environment.OSVersion.Platform & vbCrLf

strMessage &= "Machine Name " & Environment.MachineName & vbCrLf

MessageBox.Show(strMessage)

'Expand some environment variables with the values

strMessage = "CPU is %PROCESSOR\_LEVEL% Path is %Path%"

MessageBox.Show(Environment.ExpandEnvironmentVariables(strMessage))

'Print out all environment variables

strMessage = ""

For Each myEntry In Environment.GetEnvironmentVariables

strMessage &= myEntry.Key & " = " & myEntry.Value & vbCrLf

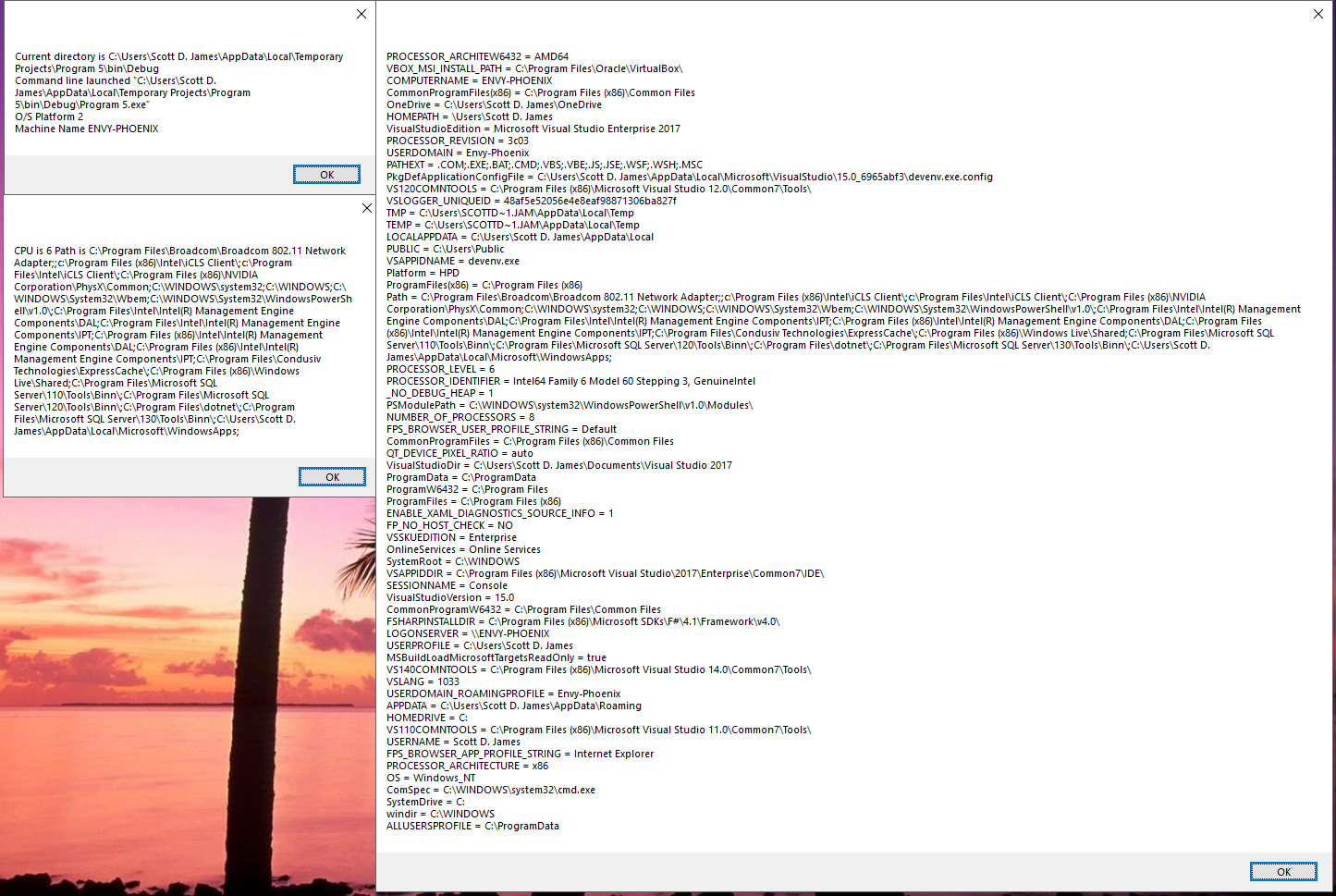
Next

MessageBox.Show(strMessage)

End Sub

End Class

Here's a screenshot showing the details from the program’s three messageboxes:



Here are the properties of the Environment Class:

|  |  |
| --- | --- |
| *Property* | *Description* |
| [CommandLine](http://msdn.microsoft.com/en-us/library/system.environment.commandline.aspx) | Gets the command line for this process. |
| [CurrentDirectory](http://msdn.microsoft.com/en-us/library/system.environment.currentdirectory.aspx) | Gets or sets the fully qualified path of the current working directory. |
| [CurrentManagedThreadId](http://msdn.microsoft.com/en-us/library/system.environment.currentmanagedthreadid.aspx) | Gets a unique identifier for the current managed thread. |
| [ExitCode](http://msdn.microsoft.com/en-us/library/system.environment.exitcode.aspx) | Gets or sets the exit code of the process. |
| [HasShutdownStarted](http://msdn.microsoft.com/en-us/library/system.environment.hasshutdownstarted.aspx) | Gets a value indicating whether the common language runtime (CLR) is shutting down. |
| [Is64BitOperatingSystem](http://msdn.microsoft.com/en-us/library/system.environment.is64bitoperatingsystem.aspx) | Determines whether the current operating system is a 64-bit operating system. |
| [Is64BitProcess](http://msdn.microsoft.com/en-us/library/system.environment.is64bitprocess.aspx) | Determines whether the current process is a 64-bit process. |
| [MachineName](http://msdn.microsoft.com/en-us/library/system.environment.machinename.aspx) | Gets the NetBIOS name of this local computer. |
| [NewLine](http://msdn.microsoft.com/en-us/library/system.environment.newline.aspx) | Gets the newline string defined for this environment. |
| [OSVersion](http://msdn.microsoft.com/en-us/library/system.environment.osversion.aspx) | Gets an [OperatingSystem](http://msdn.microsoft.com/en-us/library/system.operatingsystem.aspx) object that contains the current platform identifier and version number. |
| [ProcessorCount](http://msdn.microsoft.com/en-us/library/system.environment.processorcount.aspx) | Gets the number of processors on the current machine. |
| [StackTrace](http://msdn.microsoft.com/en-us/library/system.environment.stacktrace.aspx) | Gets current stack trace information. |
| [SystemDirectory](http://msdn.microsoft.com/en-us/library/system.environment.systemdirectory.aspx) | Gets the fully qualified path of the system directory. |
| [SystemPageSize](http://msdn.microsoft.com/en-us/library/system.environment.systempagesize.aspx) | Gets the amount of memory for an operating system's page file. |
| [TickCount](http://msdn.microsoft.com/en-us/library/system.environment.tickcount.aspx) | Gets the number of milliseconds elapsed since the system started. |
| [UserDomainName](http://msdn.microsoft.com/en-us/library/system.environment.userdomainname.aspx) | Gets the network domain name associated with the current user. |
| [UserInteractive](http://msdn.microsoft.com/en-us/library/system.environment.userinteractive.aspx) | Gets a value indicating whether the current process is running in user interactive mode. |
| [UserName](http://msdn.microsoft.com/en-us/library/system.environment.username.aspx) | Gets the user name of the person who is currently logged on to the Windows operating system. |
| [Version](http://msdn.microsoft.com/en-us/library/system.environment.version.aspx) | Gets a [Version](http://msdn.microsoft.com/en-us/library/system.version.aspx) object that describes the major, minor, build, and revision numbers of the common language runtime. |
| [WorkingSet](http://msdn.microsoft.com/en-us/library/system.environment.workingset.aspx) | Gets the amount of physical memory mapped to the process context. |

Here are the Environment class methods:

|  |  |
| --- | --- |
| *Method* | *Description* |
| [Exit](http://msdn.microsoft.com/en-us/library/system.environment.exit.aspx) | Terminates this process and gives the underlying operating system the specified exit code. |
| [ExpandEnvironmentVariables](http://msdn.microsoft.com/en-us/library/system.environment.expandenvironmentvariables.aspx) | Replaces the name of each environment variable embedded in the specified string with the string equivalent of the value of the variable, then returns the resulting string. |
| [FailFast(String)](http://msdn.microsoft.com/en-us/library/ms131100.aspx) | Immediately terminates a process after writing a message to the Windows Application event log, and then includes the message in error reporting to Microsoft. |
| [FailFast(String, Exception)](http://msdn.microsoft.com/en-us/library/dd289240.aspx) | Immediately terminates a process after writing a message to the Windows Application event log, and then includes the message and exception information in error reporting to Microsoft. |
| [GetCommandLineArgs](http://msdn.microsoft.com/en-us/library/system.environment.getcommandlineargs.aspx) | Returns a string array containing the command-line arguments for the current process. |
| [GetEnvironmentVariable(String)](http://msdn.microsoft.com/en-us/library/77zkk0b6.aspx) | Retrieves the value of an environment variable from the current process. |
| [GetEnvironmentVariable(String, EnvironmentVariableTarget)](http://msdn.microsoft.com/en-us/library/y6k3c7b0.aspx) | Retrieves the value of an environment variable from the current process or from the Windows operating system registry key for the current user or local machine. |
| [GetEnvironmentVariables](http://msdn.microsoft.com/en-us/library/1h5xxewc.aspx) | Retrieves all environment variable names and their values from the current process. |
| [GetEnvironmentVariables(EnvironmentVariableTarget)](http://msdn.microsoft.com/en-us/library/40df59yb.aspx) | Retrieves all environment variable names and their values from the current process, or from the Windows operating system registry key for the current user or local machine. |
| [GetFolderPath(Environment.SpecialFolder)](http://msdn.microsoft.com/en-us/library/14tx8hby.aspx) | Gets the path to the system special folder that is identified by the specified enumeration. |
| [GetFolderPath(Environment.SpecialFolder, Environment.SpecialFolderOption)](http://msdn.microsoft.com/en-us/library/dd992682.aspx) | Gets the path to the system special folder that is identified by the specified enumeration, and uses a specified option for accessing special folders. |
| [GetLogicalDrives](http://msdn.microsoft.com/en-us/library/system.environment.getlogicaldrives.aspx) | Returns an array of string containing the names of the logical drives on the current computer. |
| [SetEnvironmentVariable(String, String)](http://msdn.microsoft.com/en-us/library/z46c489x.aspx) | Creates, modifies, or deletes an environment variable stored in the current process. |
| [SetEnvironmentVariable(String, String, EnvironmentVariableTarget)](http://msdn.microsoft.com/en-us/library/96xafkes.aspx) | Creates, modifies, or deletes an environment variable stored in the current process or in the Windows operating system registry key reserved for the current user or local machine. |

# Using the Application Class

Whereas the Environment class looked at the operating system/runtime environment, the Application class provides many properties, methods and events that relate to an application and its execution. Part of the Application class deals with thread-related issues. We will examine threads in an upcoming chapter.

Let's begin by taking a look at the properties, methods and events available inside of the class, starting with the properties:

|  |  |
| --- | --- |
| *Property* | *Purpose* |
| AllowQuit | Gets a value indicating whether the caller can quit this application. |
| CommonAppDataPath | Provides the path of application data that is shared among all users. |
| CommonAppDataRegistry | Provides the key of the application data that is shared among all users. |
| CompanyName | Provides the company name that the application has been developed by. |
| CurrentCulture | Provides culture information about the current thread. |
| CurrentInputLanguage | Provides the current input language for the current thread. |
| ExecutablePath | Provides the complete path and filename of the executable. |
| LocalUserAppDataPath | Gets the path for the application data of a local, non-roaming user. |
| OpenForms | Gets a collection of open forms owned by this application. |
| ProductName | Provides the product name of the running application. |
| ProductVersion | Provides the version number of the running application. |
| RenderWithVisualStyles | Gets a value specifying whether the current application is drawing controls with visual styles. |
| SafeTopLevelCaptionFormat | Gets or sets the format string to apply to top-level window captions when they are displayed with a warning banner. |
| StartupPath | Provides the path of the executable file that started the application. |
| UserAppDataPath | Provides the path of the application data of a local user. |
| UserAppDataRegistry | Provides the key of the application data of a local user. |
| UseWaitCursor | Gets or sets whether the wait cursor is used for all open forms of the application. |
| VisualStyleState | Gets a value that specifies how visual styles are applied to application windows. |

Here are the methods of the Application class:

|  |  |
| --- | --- |
| *Method* | *Purpose* |
| [AddMessageFilter](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.addmessagefilter.aspx) | Adds a message filter to monitor Windows messages as they are routed to their destinations. |
| [DoEvents](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.doevents.aspx) | Processes all Windows messages currently in the message queue. |
| [EnableVisualStyles](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.enablevisualstyles.aspx) | Enables visual styles for the application. |
| [Exit](http://msdn.microsoft.com/en-us/library/ms157894.aspx) | Informs all message pumps that they must terminate, and then closes all application windows after the messages have been processed. |
| [Exit(CancelEventArgs)](http://msdn.microsoft.com/en-us/library/ms223873.aspx) | Informs all message pumps that they must terminate, and then closes all application windows after the messages have been processed. |
| [ExitThread](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.exitthread.aspx) | Exits the message loop on the current thread and closes all windows on the thread. |
| [FilterMessage](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.filtermessage.aspx) | Runs any filters against a window message, and returns a copy of the modified message. |
| [OleRequired](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.olerequired.aspx) | Initializes OLE on the current thread. |
| [OnThreadException](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.onthreadexception.aspx) | Raises the [ThreadException](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.threadexception.aspx) event. |
| [RaiseIdle](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.raiseidle.aspx) | Raises the [Idle](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.idle.aspx) event in hosted scenarios. |
| [RegisterMessageLoop](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.registermessageloop.aspx) | Registers a callback for checking whether the message loop is running in hosted environments. |
| [RemoveMessageFilter](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.removemessagefilter.aspx) | Removes a message filter from the message pump of the application. |
| [Restart](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.restart.aspx) | Shuts down the application and starts a new instance immediately. |
| [Run](http://msdn.microsoft.com/en-us/library/ms157900.aspx) | Begins running a standard application message loop on the current thread, without a form. |
| [Run(ApplicationContext)](http://msdn.microsoft.com/en-us/library/ms157901.aspx) | Begins running a standard application message loop on the current thread, with an [ApplicationContext](http://msdn.microsoft.com/en-us/library/system.windows.forms.applicationcontext.aspx). |
| [Run(Form)](http://msdn.microsoft.com/en-us/library/ms157902.aspx) | Begins running a standard application message loop on the current thread, and makes the specified form visible. |
| [SetCompatibleTextRenderingDefault](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.setcompatibletextrenderingdefault.aspx) | Sets the application-wide default for the UseCompatibleTextRendering property defined on certain controls. |
| [SetSuspendState](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.setsuspendstate.aspx) | Suspends or hibernates the system, or requests that the system be suspended or hibernated. |
| [SetUnhandledExceptionMode(UnhandledExceptionMode)](http://msdn.microsoft.com/en-us/library/ms157905.aspx) | Instructs the application how to respond to unhandled exceptions. |
| [SetUnhandledExceptionMode(UnhandledExceptionMode, Boolean)](http://msdn.microsoft.com/en-us/library/ms223898.aspx) | Instructs the application how to respond to unhandled exceptions, optionally applying thread-specific behavior. |
| [UnregisterMessageLoop](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.unregistermessageloop.aspx) | Unregisters the message loop callback made with [RegisterMessageLoop](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.registermessageloop.aspx). |

Events of the Application class:

|  |  |
| --- | --- |
| *Event* | *Purpose* |
| ApplicationExit | This event fires when the application is about to shut down and all of its forms have been closed. |
| EnterThreadModal | Occurs when the application is about to enter a modal state. |
| Idle | This event fires after the application has processed all the messages in the input queue and is entering the idle state. |
| LeaveThreadModal | Occurs when the application is about to leave a modal state. |
| ThreadException | This event fires when an unhandled thread exception occurs and the event permits the user to continue or abort the application. |
| ThreadExit | This event fires when a thread is about to terminate. |

Let's take a look at a program that shows some of the more commonly used items from the Application class. This program will display information about the application itself: the startup path, the company name and the product name and version. Remember that we saw some of this same information in the About form which VB can generate for us.

The form has a single Command button with the default name:

'Chapter 17 - Program 6

Public Class Form1

Private Sub Button1\_Click(sender As Object, e As EventArgs)

Handles Button1.Click

Dim strMsg As String

'Get some attributes from the Application class

strMsg = "StartupPath " & Application.StartupPath & vbCrLf

strMsg &= "CompanyName " & Application.CompanyName & vbCrLf

strMsg &= "ProductName " & Application.ProductName & vbCrLf

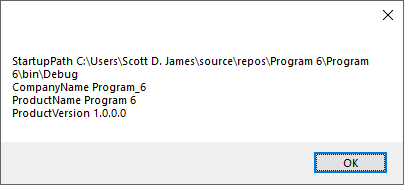
strMsg &= "ProductVersion " & Application.ProductVersion & vbCrLf

MessageBox.Show(strMsg)

End Sub

End Class

Here's the output from the application:



# Making API Calls

Although VB is very powerful on its own, there are still some things that it cannot directly perform. Fortunately most of what VB cannot do natively, the Windows API (Application Programmer Interface) can. The API is an interface to the Windows operating system itself – the core files that provide most of the functionality of Windows.

This section is going to show you how to make Windows API calls from VB. This is probably as good a place as any to discuss API calls since we have been concentrating on getting VB to communicate with other applications and objects in this chapter.

These are the main functional category libraries that make up the Windows API:

1. Windows Management (USER32.DLL) – this library provides the essential functions to instantiating and managing your applications. All basic I/O goes through this library including mouse and keyboard input and all messages sent to your application.
2. Graphics Device Interface (GDI32.DLL) – this library provides the functions you use to manage all supported graphical devices on your system such as the display monitor and printer.
3. System Services (KERNEL32.DLL) – this library provides the functions that access the resources of the computer and the operating system.
4. Advanced Services (ADVAPI32.DLL) – this library provides additional functionality located in the kernel, access to the registry, services, user accounts and so forth.
5. Common Dialog Box Library (COMDLG32.DLL) – this library contains the file open/save, fonts, color and printing dialog boxes.
6. Common Control Library (COMCTL32.DLL) – this library is where the basic controls (TextBox, Command buttons, ListBox, et cetera) for our Windows forms applications live.
7. Windows Shell (SHELL32.DLL) – this library is what allows our applications to access the functionality available in the operating system’s shell.
8. Network Services (NETAPI32.DLL) – this library is where we access the various networking capabilities of Windows.

Whew! The easiest way to work with API calls is through the Windows API Code Pack for the Microsoft .NET Framework. There is a viewer utility that allows you to see the API declarations and data structures that are available and can be copied from the clipboard into your VB application. Microsoft used to release an official version of this, but discontinued it (of course). However, a group of developers have created and released an unofficial version that is on Git and NuGet… It’s worthwhile to find and install if you need to do much with the Windows API.

Let’s write an application that uses the Windows API to get the name of the current user logged onto our computer. The form has a default named Command button on it:

'Chapter 17 - Program 7

Public Class Form1

'Here's the necessary API call to get the current user

'name on the system. I did not type this function in --

'instead I copied it from the API viewer

Declare Function GetUserName Lib "advapi32.dll" Alias \_

"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Integer) \_

As Integer

Private Sub Button1\_Click(sender As Object, e As EventArgs)

Handles Button1.Click

Dim RetVal As Integer

Dim UserName As String

Dim Buffer As String

'This is where the user name will be stored

'We need to make a buffer that is 25 characters long

'so let's using String to do that. 25 characters should

'be long enough for most usernames.

Buffer = New String(CChar(" "), 25)

'Call the API function:

'Two parameters are sent in - the buffer and the size

'of the buffer

RetVal = GetUserName(Buffer, 25)

'Parse out the username. Chr(0) the Null character

'marks the end of the string, so all the characters from

'the leftmost up to Null - 1 are the username

UserName = Strings.Left(Buffer, InStr(Buffer, Chr(0)) - 1)

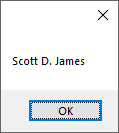
'Show the username

MessageBox.Show(UserName)

End Sub

End Class

Here is a screenshot of the application in action:



A word of caution needs to be mentioned here! You can really mess things up on your computer if you’re not careful when making API calls. You could create enough problems that you may have to reformat and reload your system. I provided this section as a simple example so that you would know what API calls are, why they are used and how to perform one. You need to get very good at understanding the parameter information that is expected in an API call and what the API call is for …

I do not recommend that you blindly try guessing about API calls' purposes or parameters. There are a few books on the market, most of them you will have to purchase used, that discuss API calls. My personal favorite is an oldie: Dan Appleman’s “Visual Basic Programmer’s Guide to the Win32 API” which was published by ZD Press. If you are going to work with the API, this book is a must have! I haven’t seen many .NET books that even mention the API since Microsoft really doesn’t want you messing with it. They have tried to put everything they think you would need into the managed .NET libraries. There are still occasional calls to use the API however and it was a standard piece of programming knowledge and practice required back in the pre-.NET days!

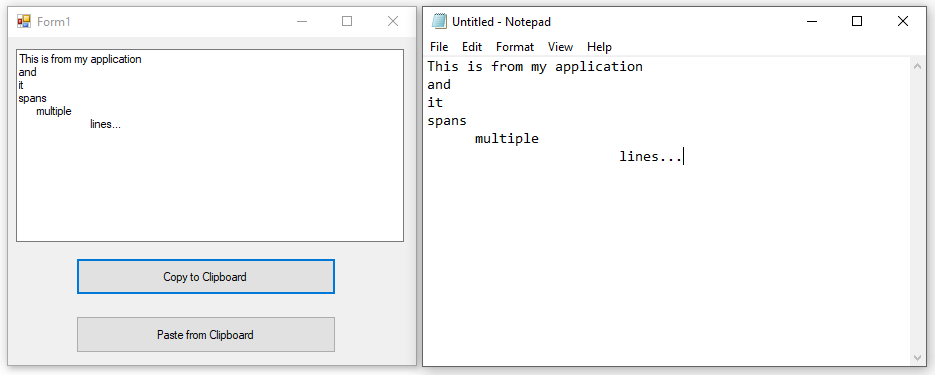
It’s also worthwhile to note that many calls require integers, longs and so forth – these data types have changed in size from pre-.NET to .NET, thus you will have to perform some data conversions to get some of the older API code to work right. Realize that Win64 and WinRT have completely different APIs, so this is something to consider when you decide that you think you need to do something via an API call!

YOU HAVE BEEN WARNED! This is really the first chapter where I’ve shown you anything that could really screw up your computer. Microsoft moved to the managed .NET framework, specifically in part, to try to safeguard the system against programmers messing it up. It’s not to say that you can’t do stupid things in .NET that will break your system, it’s just that it’s a lot more difficult to accidentally do.

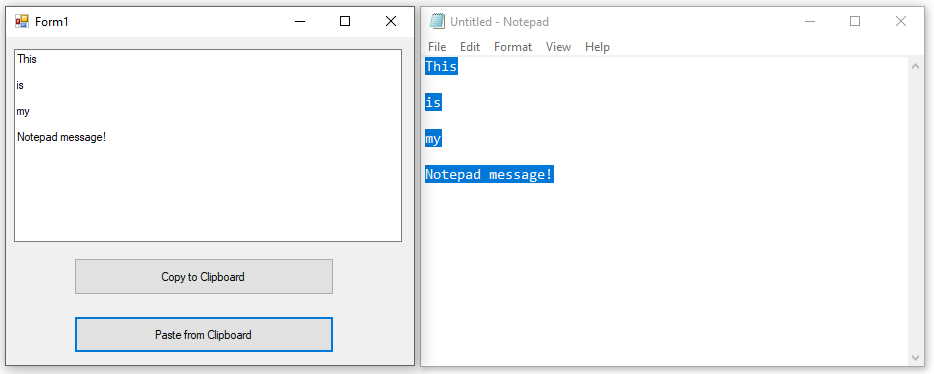
So, if you decide to start messing around with changing registry entries or making API calls, the onus is yours to ensure that things work right before you execute them. Little mistakes at that level can result in rebuilding your entire computer system. YOU HAVE BEEN WARNED!

# The Clipboard

So, every time you’ve cut, copied or pasted, you’ve used the clipboard. While we are on the subject of communicating between applications, it seems appropriate to visit this concept. In the example program we will examine, I can press the Copy to Clipboard button to copy whatever is in the application’s textbox to the clipboard. I then hopped over to Notepad and pasted it in there:



I can copy and paste in the opposite direction as well:



There’s not a whole lot control-wise on the form. We have a multiline TextBox named txtMyText and the two Command buttons are cmdCopy and cmdPaste, from top to bottom.

Here’s the code for the application:

'Chapter 17 - Program 8

Public Class Form1

Private Sub cmdCopy\_Click(sender As Object, e As EventArgs)

Handles cmdCopy.Click

Clipboard.SetText(txtMyText.Text)

End Sub

Private Sub cmdPaste\_Click(sender As Object, e As EventArgs)

Handles cmdPaste.Click

txtMyText.Text = Clipboard.GetText()

End Sub

End Class

The clipboard object exposes the following properties and methods:

|  |  |
| --- | --- |
| *Name* | *Description* |
| [Clear](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.clear.aspx) | Removes all data from the Clipboard. |
| [ContainsAudio](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.containsaudio.aspx) | Indicates whether there is data on the Clipboard in an Audio format or can be converted to that format. |
| [ContainsData](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.containsdata.aspx) | Indicates whether there is data on the Clipboard that is in the specified format or can be converted to that format. |
| [ContainsFileDropList](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.containsfiledroplist.aspx) | Indicates whether there is data on the Clipboard that is in the [FileDrop](http://msdn.microsoft.com/en-us/library/system.windows.forms.dataformats.filedrop.aspx)List format or can be converted to that format. |
| [ContainsImage](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.containsimage.aspx) | Indicates whether there is data on the Clipboard in an image format or can be converted to that format. |
| [ContainsText](http://msdn.microsoft.com/en-us/library/a3cyzt72.aspx) | Indicates whether there is data on the Clipboard in the [Text](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) or [UnicodeText](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) format, depending on the operating system. |
| [ContainsText(TextDataFormat)](http://msdn.microsoft.com/en-us/library/t81k888y.aspx) | Indicates whether there is text data on the Clipboard in the format indicated by the specified [TextDataFormat](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) value. |
| [GetAudioStream](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.getaudiostream.aspx) | Retrieves an audio stream from the Clipboard. |
| [GetData](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.getdata.aspx) | Retrieves data from the Clipboard in the specified format. |
| [GetDataObject](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.getdataobject.aspx) | Retrieves the data that is currently on the system Clipboard. |
| [GetFileDropList](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.getfiledroplist.aspx) | Retrieves a collection of filenames from the Clipboard. |
| [GetImage](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.getimage.aspx) | Retrieves an image from the Clipboard. |
| [GetText](http://msdn.microsoft.com/en-us/library/kz40084e.aspx) | Retrieves text data from the Clipboard in the [Text](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) or [UnicodeText](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) format, depending on the operating system. |
| [GetText(TextDataFormat)](http://msdn.microsoft.com/en-us/library/944ft9kf.aspx) | Retrieves text data from the Clipboard in the format indicated by the specified [TextDataFormat](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) value. |
| [SetAudio(Byte())](http://msdn.microsoft.com/en-us/library/ea51k1k6.aspx) | Clears the Clipboard and then adds a [Byte](http://msdn.microsoft.com/en-us/library/system.byte.aspx) array in the [WaveAudio](http://msdn.microsoft.com/en-us/library/system.windows.forms.dataformats.waveaudio.aspx) format after converting it to a [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream.aspx). |
| [SetAudio(Stream)](http://msdn.microsoft.com/en-us/library/s39c87sa.aspx) | Clears the Clipboard and then adds a [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream.aspx) in the [WaveAudio](http://msdn.microsoft.com/en-us/library/system.windows.forms.dataformats.waveaudio.aspx) format. |
| [SetData](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.setdata.aspx) | Clears the Clipboard and then adds data in the specified format. |
| [SetDataObject(Object)](http://msdn.microsoft.com/en-us/library/5b8kt5z4.aspx) | Clears the Clipboard and then places non-persistent data on it. |
| [SetDataObject(Object, Boolean)](http://msdn.microsoft.com/en-us/library/cs5ebdfz.aspx) | Clears the Clipboard and then places data on it and specifies whether the data should remain after the application exits. |
| [SetDataObject(Object, Boolean, Int32, Int32)](http://msdn.microsoft.com/en-us/library/ms158293.aspx) | Clears the Clipboard and then attempts to place data on it the specified number of times, with the specified delay between attempts, optionally leaving the data on the Clipboard after the application exits. |
| [SetFileDropList](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.setfiledroplist.aspx) | Clears the Clipboard and then adds a collection of filenames in the [FileDrop](http://msdn.microsoft.com/en-us/library/system.windows.forms.dataformats.filedrop.aspx)List format. |
| [SetImage](http://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard.setimage.aspx) | Clears the Clipboard and then adds an [Image](http://msdn.microsoft.com/en-us/library/system.drawing.image.aspx) to it. |
| [SetText(String)](http://msdn.microsoft.com/en-us/library/ydby206k.aspx) | Clears the Clipboard and then adds text data in the [Text](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) or [UnicodeText](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) format, depending on the operating system. |
| [SetText(String, TextDataFormat)](http://msdn.microsoft.com/en-us/library/tbfb3z56.aspx) | Clears the Clipboard and then adds text data in the format indicated by the specified [TextDataFormat](http://msdn.microsoft.com/en-us/library/system.windows.forms.textdataformat.aspx) value. |