Use the Visual Basic 6 APP object to polish your applications

One of the more interesting 'built in' System Objects in Visual Basic is the App Object. The APP (short for Application) Object gives your Visual Basic program information about the Application or Project itself. If you've never used some of the APP Object's Properties and Methods, you don't know what you're missing. APP properties can be accessed at runtime to give your program additional functionality and polish. We'll see how in just a bit.

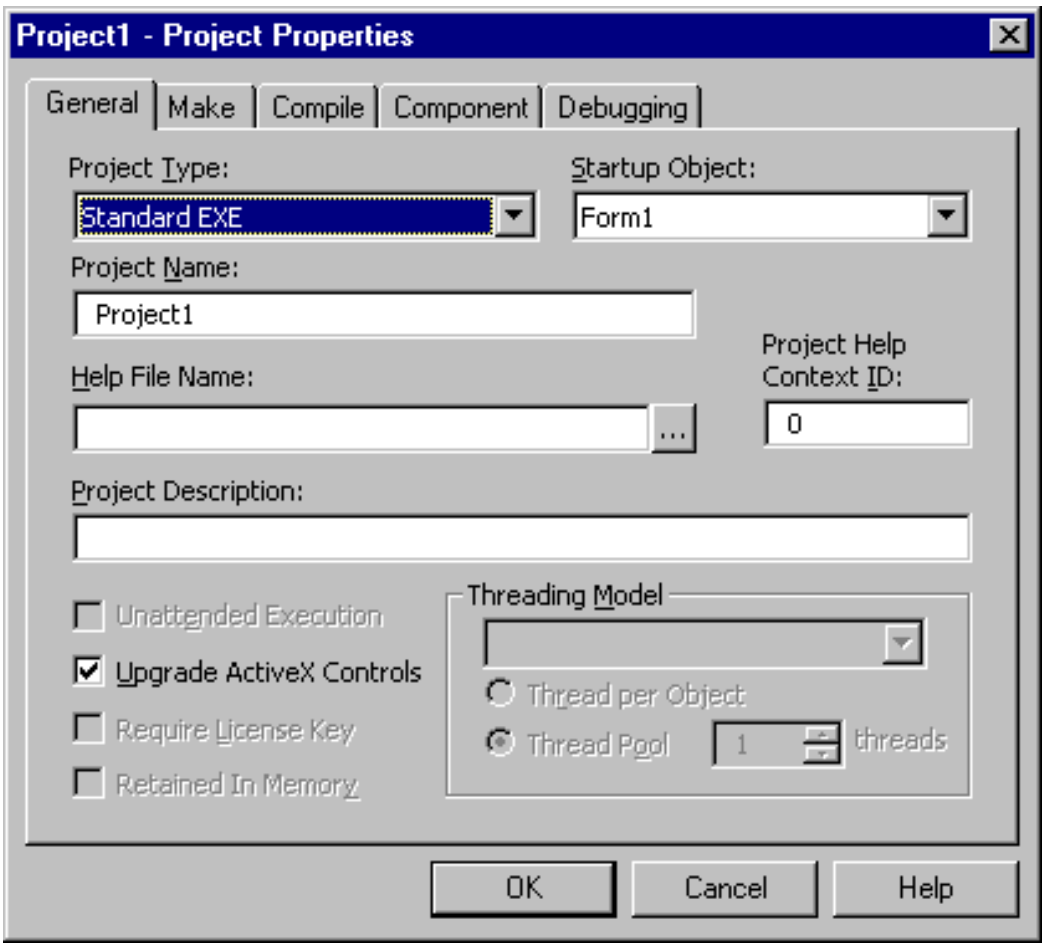
APP Object Properties

Let's start with the APP Object's Properties. As I mentioned, the APP Object specializes in providing you with information about your program. Most of this information can be entered by the program via the Project Properties Windows, although some of it is available only at runtime.

Let's examine the Project Properties Window now. The Project Properties window contains 5 tabs, two of which are of interest to us in this discussion of the APP Object. Let's start with the General Tab.

General Tab APP Properties

Here's a screen shot of the General Tab of a Visual Basic Project…

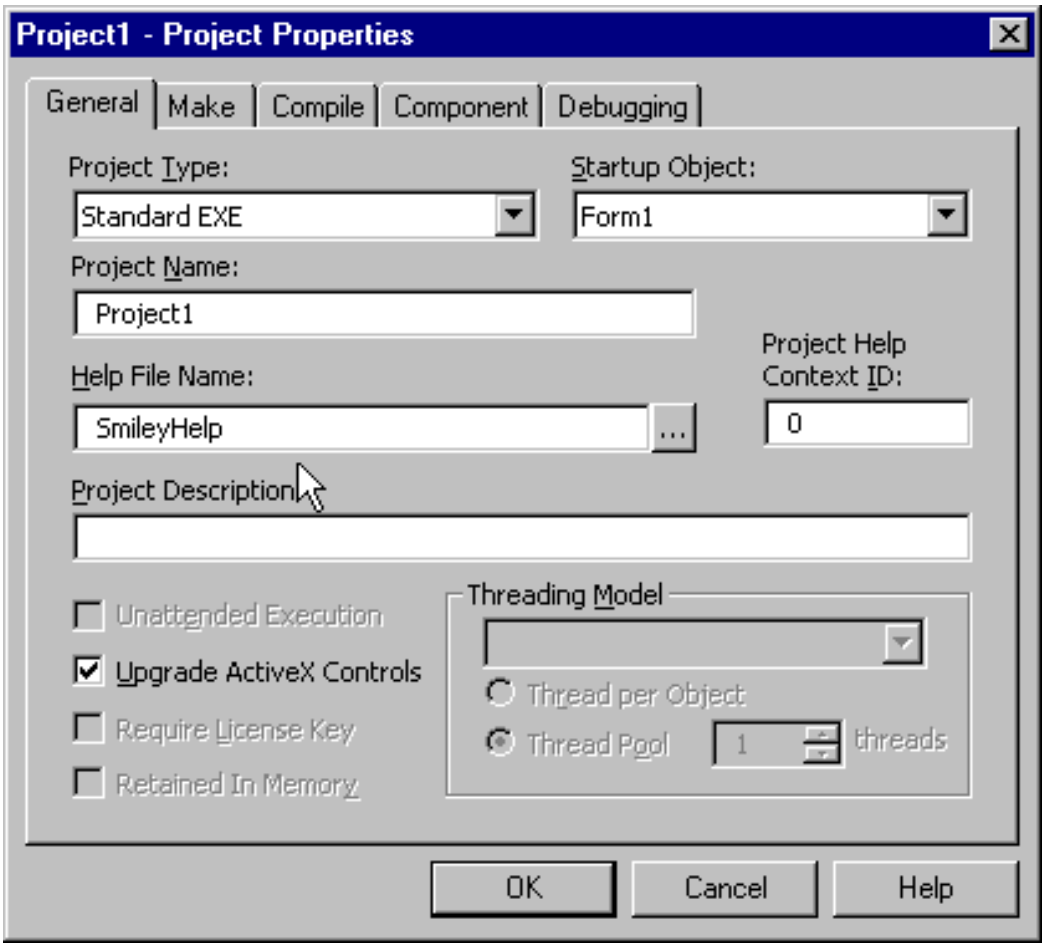


There's just one Property of the APP object which can be set via the General Tab, and that's the HelpFile

Property…

HelpFile

The HelpFile Property of the APP Object specifies the path and filename of a Help file used by your application to display Help or online documentation, and can be specified in the Help File Name of the **General** Tab of your Project's Properties Window…

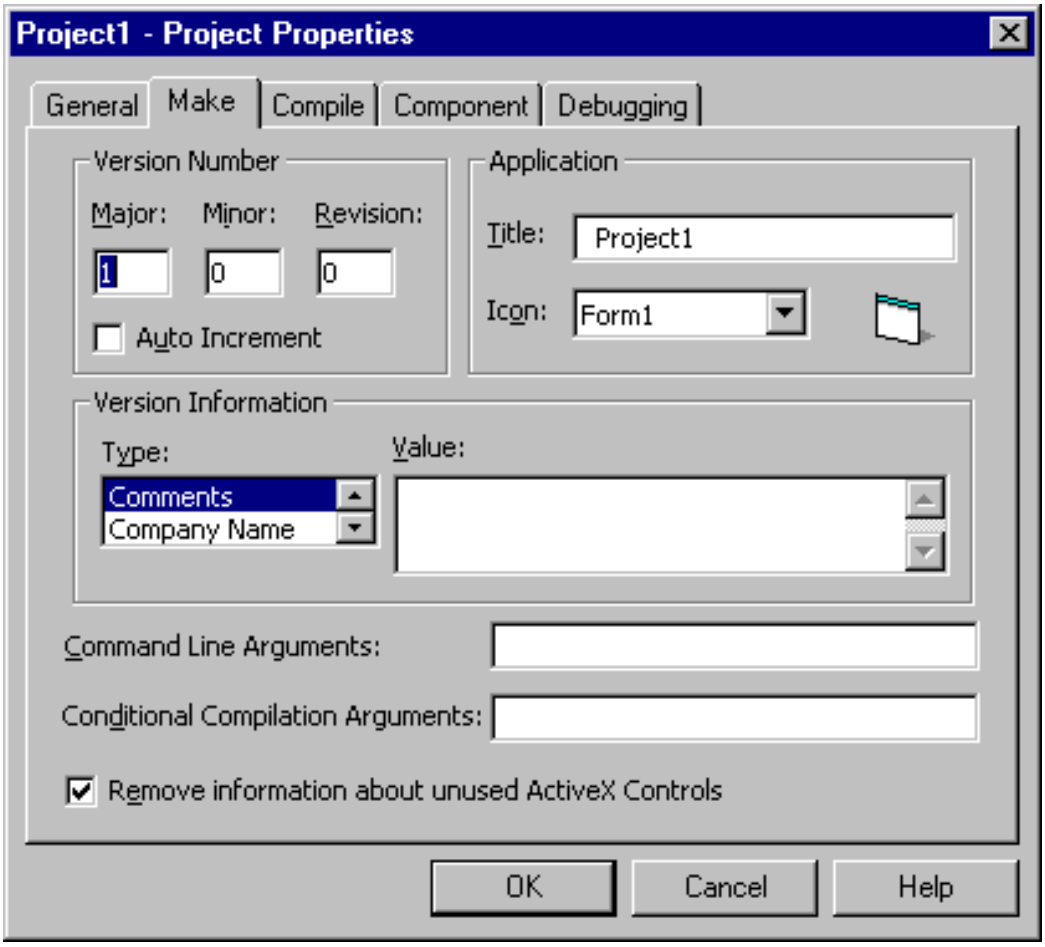


If you've created a Help file for your application and set the application's **HelpFile** property, Visual Basic automatically calls Help when a user presses the F1 key. If there is a context number in the **HelpContextID** property for either the active control or the active form, Help displays a topic corresponding to the current Help context; otherwise it displays the main contents screen.

From a programming perspective, there's not much we can do with the HelpFile property---suffice to say that if it's set to a valid Help File, it can provide additional functionality for our program.

Make Tab APP Properties

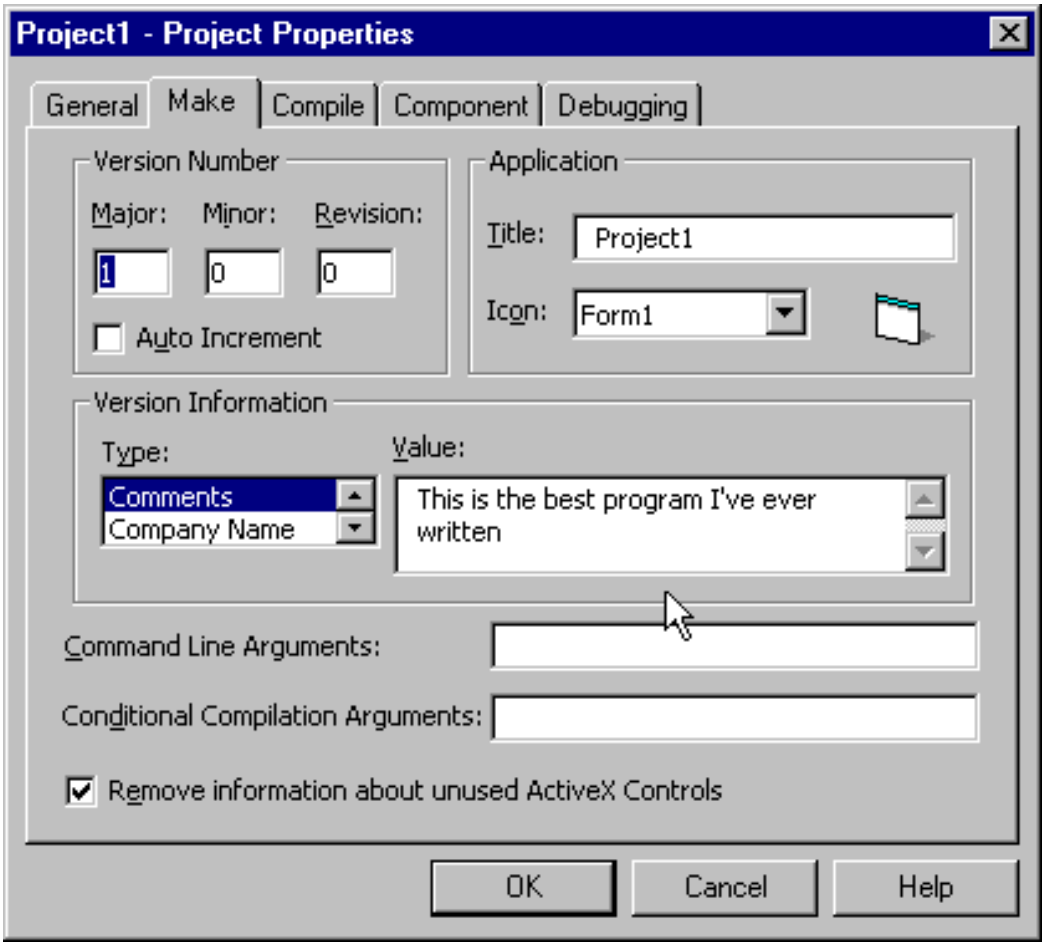
Many more APP Object properties can be set via the Make Tab of the Project Properties Window, and many of these are very valuable. Here's a screen shot of the Make Tab of a Visual Basic Project…



Let's start an alphabetical review of these properties.

Comments

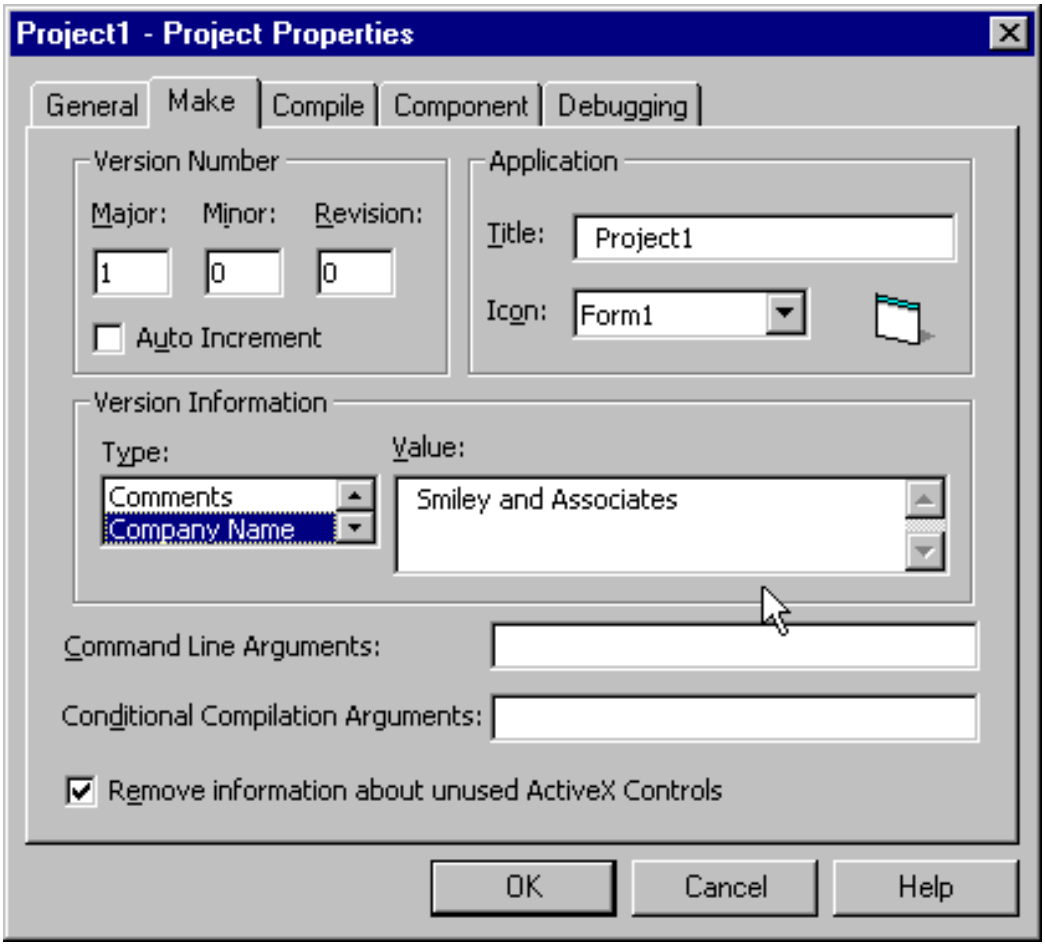
The Comments property returns or sets a string containing comments about the running application. You can set this property at design time in the Type box in the **Make** tab of the Project Properties dialog box…



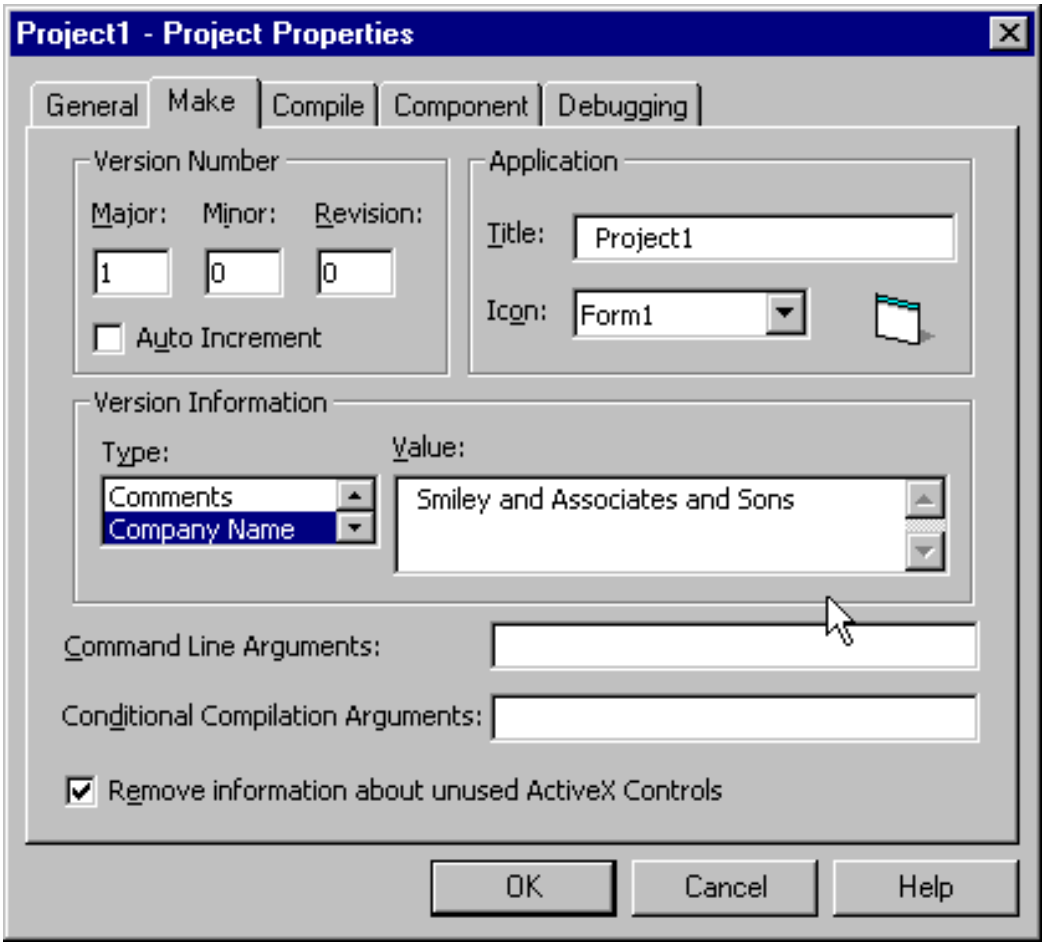
You can also access this property at runtime if you wish.

CompanyName

The CompanyName Property returns or sets a string value containing the name of the company or creator of the running application. You can set this property at design time in the Type box in the **Make** tab of the Project Properties dialog box…

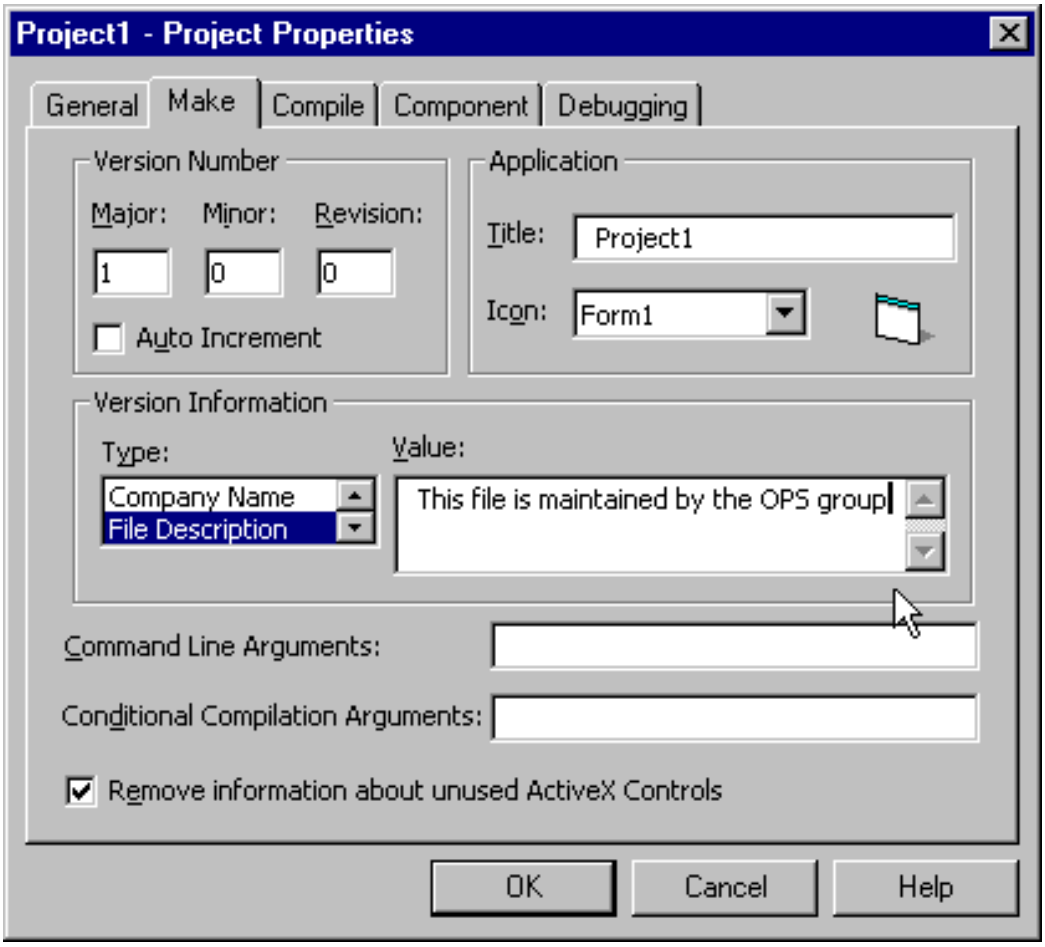


By default, the CompanyName property contains the Company Name specified when you installed Visual Basic. You can override this value if you wish…



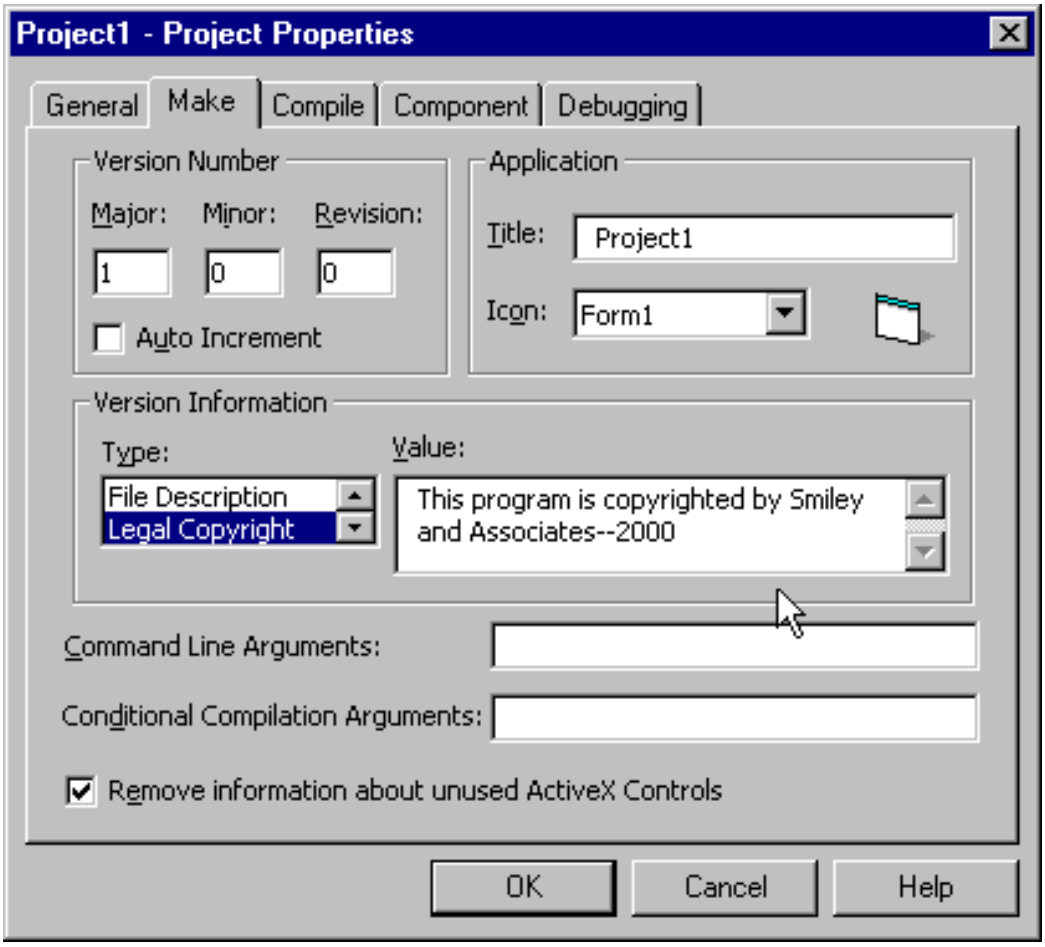
FileDescription

The FileDescription Property of the APP Object returns or sets a string value containing a file description information about the running application. You can set this property at design time in the Type box in the **Make** tab of the Project Properties dialog box…



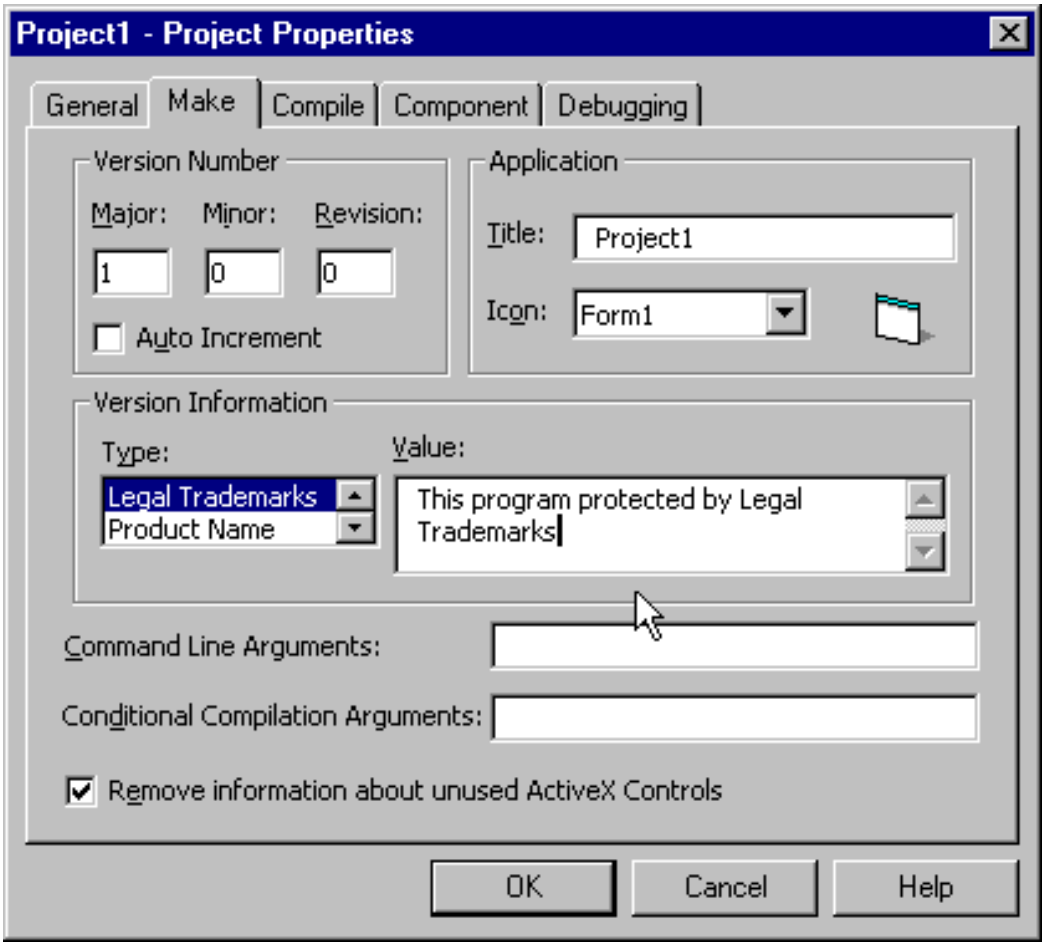
LegalCopyright

The LegalCopyright property of the APP Object returns or sets a string value containing legal copyright information about the running application. This property is Read only at run time. You can set this property at design time in the Type box in the **Make** tab of the Project Properties dialog box…



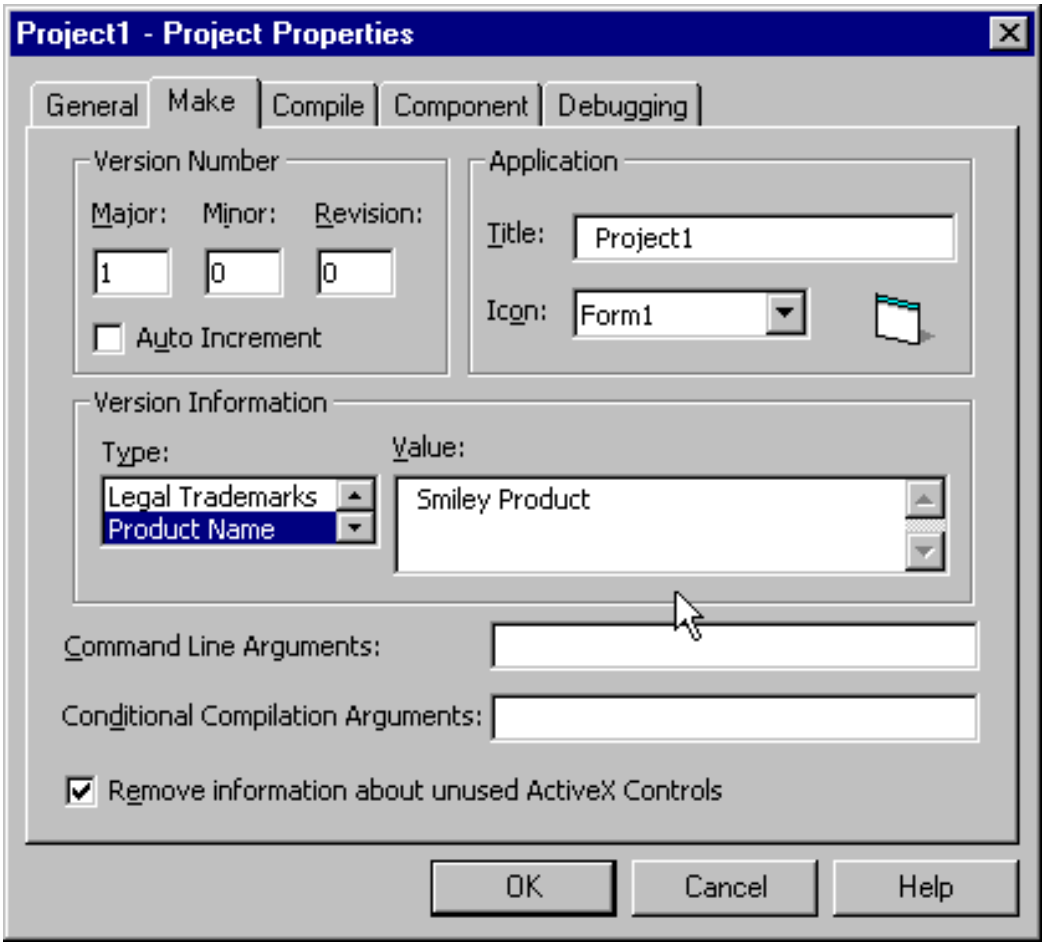
LegalTrademarks

The LegalTrademarks property of the APP Object returns or sets a string value containing legal trademark information about the running application. You can set this property at design time in the Type box in the **Make** tab of the Project Properties dialog box…



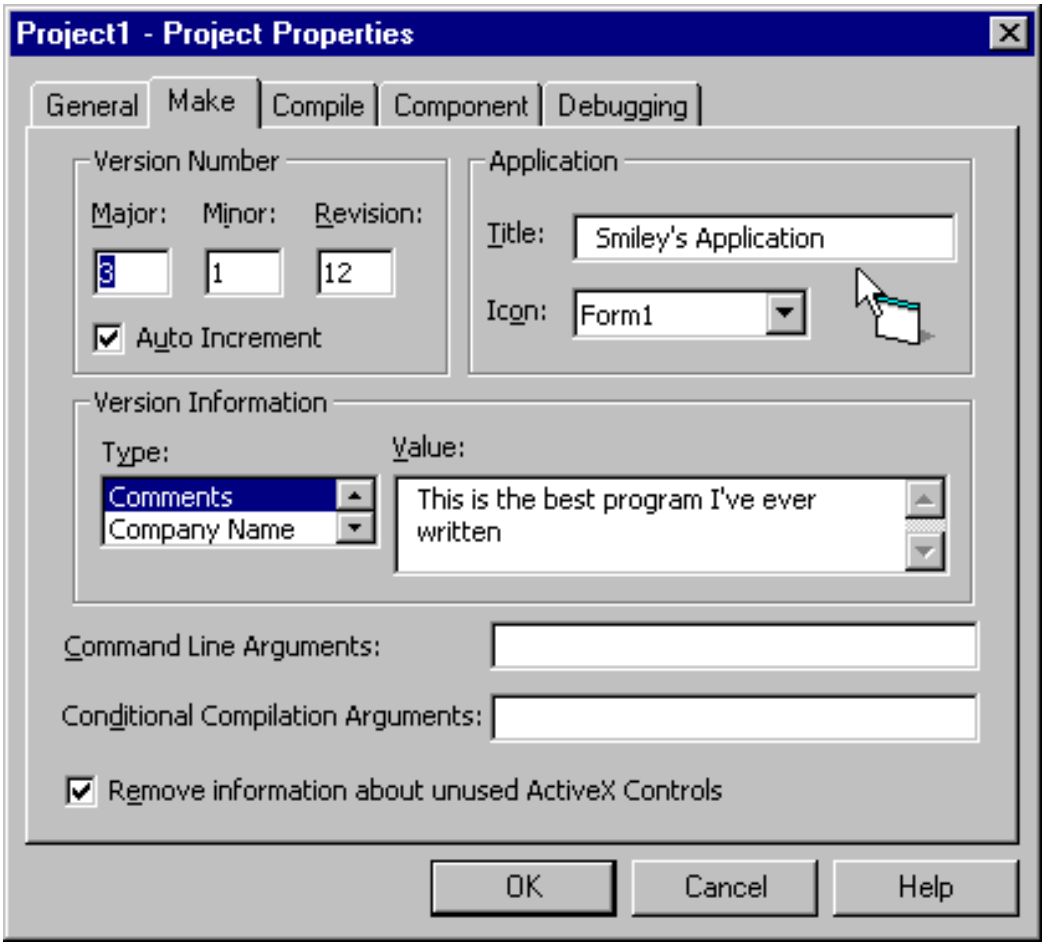
ProductName

The ProductName property of the APP Object returns or sets a string value containing the product name of the running application. You can set this property at design time in the Type box in the **Make** tab of the Project Properties dialog box…



Title

The Title Property of the APP Object returns or sets the title of the application that is displayed in the Microsoft Windows Task List. If changed at run time, changes aren't saved with the application. You can set this property at design time in the Title box in the **Make** tab of the Project Properties dialog box…



Version Numbers

So far, the properties that we've seen may not have been of great interest to you--but the next three properties---which are used to track version numbers of your compiled programs, can be a life saver.

Major

Returns or sets the major release number of the project. The value of the **Major** property is in the range from 0 to 9999. This property provides version information about the running application.

You can set this property at design time in the Major box in the **Make** tab of the Project Properties dialog box.

Minor

Returns or sets the minor release number of the project. The value of the **Minor** property is in the range from 0 to 9999. This property provides version information about the running application.

You can set this property at design time in the Minor box in the **Make** tab of the Project Properties dialog box.

Revision

Returns or sets the revision version number of the project The value of the **Revision** property is in the range from 0 to 9999.

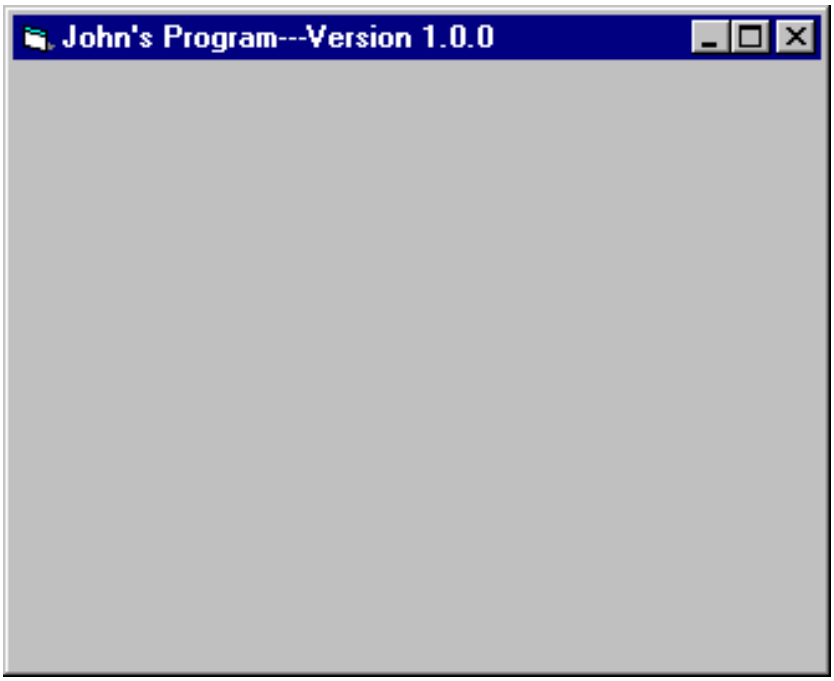
This property provides version information about the running application. You can set this property at design time in the Revision box in the **Make** tab of the Project Properties dialog box.

These number can be anything you want--typically, the initial release of a new program will have a Major value of 1, a Minor value of 0, and a Revision value of 0. You can increment either the Minor or Revision numbers for very minor changes to the program, and increment the Major value for significant changes to your program.

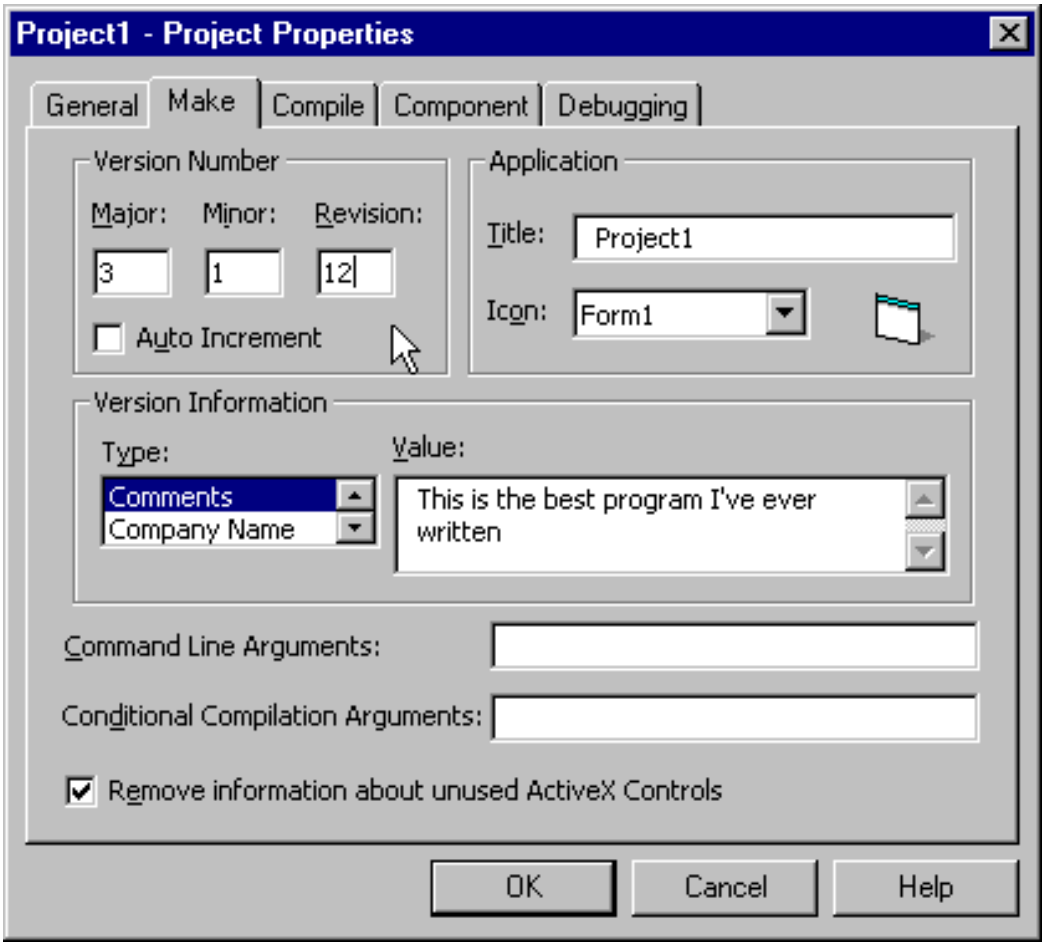
I typically display these properties as part of the Caption property of the main form of my application. This can help to avoid a lot of confusion when you distribute programs (and revisions) to your users. For instance, here's some code that will display the Major, Minor and Revision Properties of the App Object…

```
Private Sub Form_Load()

Form1.Caption = "John's Program---Version " & _
   App.Major & "." & _
   App.Minor & "." & _
   App.Revision

End Sub
```

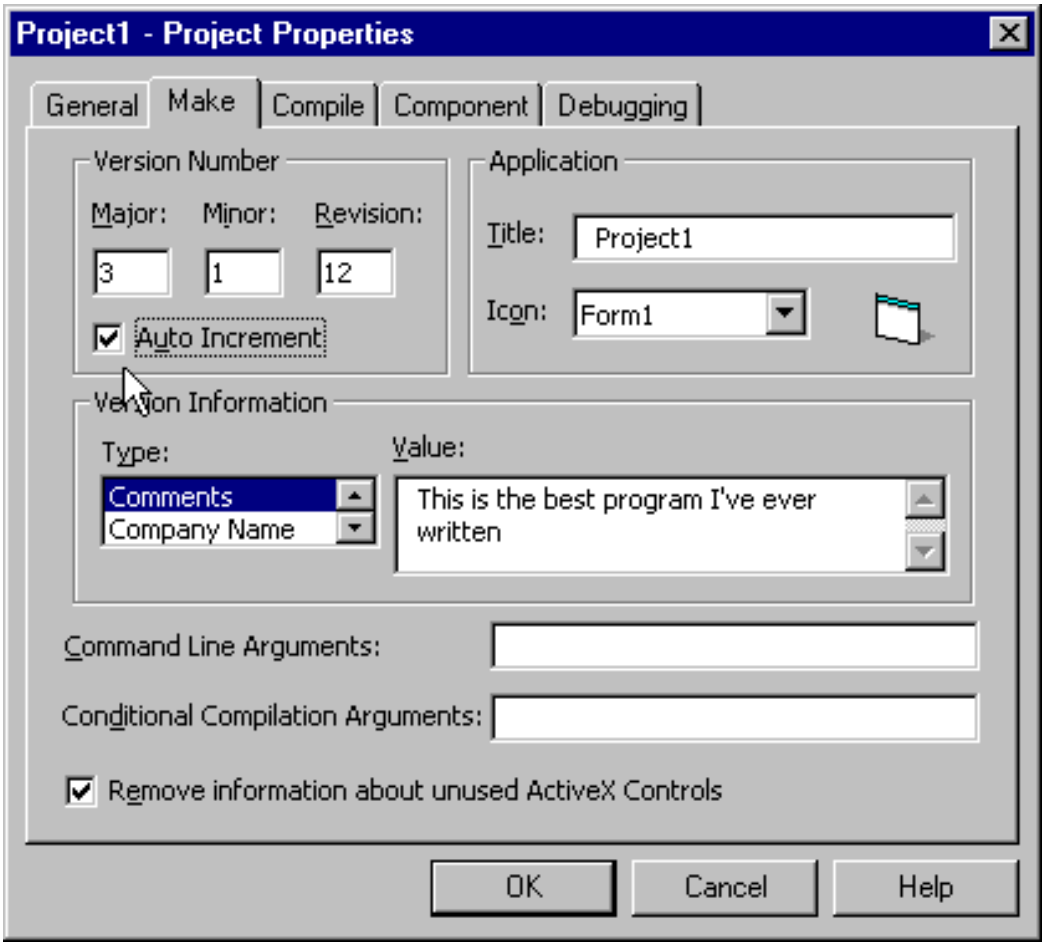
When the program starts up, the Form's caption will then appear like so…



You can make changes to the Major, Minor and Revision Properties by updating them directly via the Make Tab, like so…



You can also have Visual Basic automatically increment the Revision Property of the APP Object by checking the Auto Increment checkbox…



Placing a check in the Auto Increment checkbox tells Visual Basic to automatically increment the Revision number when your program is compiled. This can be a life saver if you make lots of revisions to your program, and are likely to forget to adjust the Revision property manually---it's done for you, ensuring that the user of your program always knows the version of the program they are running.

This series of Version Properties marks the last of the Properties that can be accessed or modified via a Property Page. Here are the remainder of the APP Object's Properties I'd like to discuss in this article.

EXEName

The EXEName Property of the APP Object returns the root name of the executable file (without the extension) that is currently running. If running in the development environment, this property returns the name of the Visual Basic project. If running as an executable, it's the name of the compiled executable.

Logging Properties and Methods

A series of very useful Properties and Methods of the APP object pertain to the little known capability of Visual Basic to log events and messages to a log file. You can use a Log File to provide an 'audit trail' of events that take place in your program. Other programmers I know make writing to a Log file a standard part of their Error Handling Procedures. Let's take a look at the Logging Properties first--both of which are read-only, and are actually set by the StartLogging Method of the APP object. **CAUTION**: On Windows NT machines, events can only be written to the Windows NT Event Log.

LogMode

The LogMode Property of the APP Object returns a value which determines how logging (through the **LogEvent** method) will be carried out. **CAUTION**: On Windows NT machines, events can only be written to the Windows NT Event Log. The possible values for the LogMode Property are:

Constant	Value	Description
vbLogAuto	0	If running on Windows 95 or later, this option logs messages to the file specified in the **LogPath** property. If running on Windows NT, messages are logged to the Windows NT Application Event Log, with "VBRunTime" used as the application source and App.Title appearing in the description.
VbLogOff	1	Turns all logging off. Messages from UI shunts as well as from the **LogEvent** method are ignored and discarded.
VbLogToFile	2	Forces logging to a file. If no valid filename is present in **LogPath**, logging is ignored, and the property is set to **vbLogOff**.
VbLogToNT	3	Forces logging to the NT event log. If not running on Windows NT, or the event log is unavailable, logging is ignored and the property is set to **vbLogOff**.
VbLogOverwrite	0x10	Indicates that the logfile should be recreated each time the application starts. This value can be combined with other mode options using the OR operator. The default action for logging is to append to the existing file. In the case of NT event logging, this flag has no meaning.
VbLogThreadID	0x20	Indicates that the current thread ID be prepended to the message, in the form "[T:0nnn] ". This value can be combined with other mode options using the OR operator. The default action is to show the thread ID only when the application is multi-threaded (either explicitly marked as thread-safe, or implemented as an implicit multithreaded app, such as a local server with the instancing property set to Single-Use, multithreaded).

This property is read-only, and is actually determined by values specified for the logMode argument of the StartLogging Method.

LogPath

The LogPath Property returns the path and filename designated to capture output from the LogEvent Method. If no **LogPath** is set, the **LogEvent** method writes to the NT LogEvent file. This property is read-only, and is actually determined by values specified for the LogTarget argument of the StartLogging Method. **CAUTION**: On Windows NT machines, events can only be written to the Windows NT Event Log.

StartLogging (This is a method)

The StartLogging Method sets the log target and log mode of an operation. The StartLogging method requires two arguments---logTarget and logMode. LogTarget is the Path and filename of the file used to capture output from the LogEvent method, and logMode determines how logging will be carried out. Possible values for logMode are:

Constant	Value	Description
vbLogAuto	0	If running on Windows 95 or later, this option logs messages to the file specified in the **LogFile** property. If running on Windows NT, messages are logged to the Windows NT Application Event Log, with "VBRunTime" used as the application source and App.Title appearing in the description.
VbLogOff	1	Turns all logging off. Messages from UI shunts as well as from the **LogEvent** method are ignored and discarded.
VbLogToFile	2	Forces logging to a file. If no valid filename is present in **LogPath**, logging is ignored, and the property is set to **vbLogOff**.
VbLogToNT	3	Forces logging to the NT event log. If not running on Windows NT, or the event log is unavailable, logging is ignored and the property is set to **vbLogOff**.
VbLogOverwrite	16	Indicates that the logfile should be recreated each time the application starts. This value can be combined with other mode options using the **OR** operator. The default action for logging is to append to the existing file. In the case of NT event logging, this flag has no meaning.
VbLogThreadID	32	Indicates that the current thread ID be prepended to the message, in the form "[T:0nnn] ". This value can be combined with other mode options using the **OR** operator. The default action is to show the thread ID only when the application is multi-threaded (either explicitly marked as thread-safe, or implemented as an implicit multithreaded app, such as a local server with the instancing property set to Single-Use, multithreaded).

LogEvent (this is a method)

This method of the APP Object logs an event in the application's log target. On Windows NT platforms, the method writes to the NT Event log. On Windows 95/98 platforms, the method writes to the file specified in the **LogPath** property; by default, if no file is specified, events will be written to a file named `vbevents.log`. The LogEvent method requires two arguments---the string to be written to the log, and the event type which has the following possible values…

Constant	Value	Description

vbLogEventTypeError	1	Error.
vbLogEventTypeWarning	2	Warning.
vbLogEventTypeInformation	4	Information.

In summary, logging can be achieved by executing the StartLogging method of the APP object to set the LogMode and LogFile properties of the APP Object, and then executing the LogEvent method of the APP Object to actually write the log entry. Here's some code that will write an entry to a file called "Logfile.txt" (or to the Windows NT Event log if running on a Windows NT machine)

```
App.StartLogging "c:\Logfile.txt", vbLogFile

App.LogEvent "File Read Error", vbLogEventTypeError
```

We have two very useful properties of the APP object to discuss---PrevInstance and Path. Let's start with PrevInstance.

PrevInstance

The PrevIntance property of the APP Object returns a value indicating whether a previous instance of the application is already running. You can use this property in a Load event procedure to specify whether a user is already running an instance of an application. Depending on the application, you might want only one instance running in the Microsoft Windows operating environment at a time.

The PrevInstance property is only useful when working with a compiled program. This code, placed in the Load event of the startup form of your application, can detect when the program is already running…

```
If App.PrevInstance = True Then
   MsgBox "The program is already running..."
   Unload Me
   End
   Exit Sub
End If
```

Path

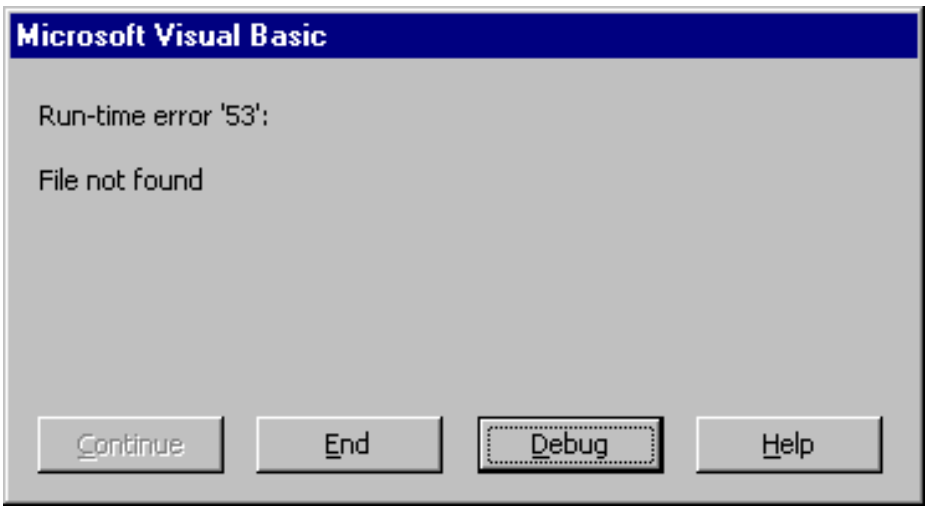
The Path property of the APP object returns a string indicating the path (drive and directory) from which the program is being run. When running from the development environment of Visual Basic, **Path** specifies the path of the project .VBP file. When running outside of Visual Basic, this property specifies the path of the executable file.

I use the Path property to build a little 'flexibility' into the programs I write. For instance, suppose I write, compile and distribute a program to a user, and part of the logic of the program is to read a disk file called

"smiley.txt" residing in the "\VBFiles" folder of the hard drive, and to display the results on the form. The code might look something like this…

```
Dim Retval

Open "c:\vbfiles\smiley.txt" For Input As #1

Do While Not EOF(1)
   Input #1, Retval
   Form1.Print Retval
Loop

Close #1
```

We're fine provided the user doesn't move the executable and the text file somewhere else. If the user does move them, this program will generate an error…



To get around that problem, we can use the Path property of the APP object to make our code a bit more flexible. The Path property will tell us the folder or directory from which our application is running---provided the text file we are looking for is in the same folder or directory, we can concatenate the name of the file to the end of the path, like this…

```
Open App.Path & "\smiley.txt" For Input As #1
```

There's just one problem---if the user moves the executable to the root directory of their hard drive or floppy, the return value of App.Path will end with a backslash

c:\

and cause the above code to bomb. For that reason, you should use an If statement to check if the length of the Path property is 3

```
If Len(App.Path) = 3 Then
```

if it is, then the user is running the program from the root directory, and you should open the file using this syntax…

```
Open App.Path & "smiley.txt" For Input As #1
```

otherwise, use this syntax…

```
Else
  Open App.Path & "\smiley.txt" For Input As #1
End If
```

Summary

I hope you enjoyed this article on the wonders of the App Object. For more information on Objects, be sure to pick up a copy of my book

Learn to Program Objects with Visual Basic