Console

Console

Description

The whole enchilada. If you want to write a real console application, using Classic VB, this is the ticket. Create a new project, set it to start with Sub Main, drop the MConsole.bas file into your application, and you're almost ready to rock. This sample provides complete support for writing *and testing* console applications within the IDE.

MConsole is designed as a lightweight COM object, using techniques developed by Matt Curland, and published in his book *Advanced Visual Basic 6*. This means you only need to initialize the object, and from there it handles teardown itself at application completion. The lightweight console object creates its own console window for output to while running within the IDE, which eliminates the need to compile before each test, and makes interactive debugging a possibility for the first time.

All the standard output functionalities you would expect are available. In this little example, you can see how easy it is to initialize the console, and start writing to it. This snippet even changes the background and foreground colors, as well as the command window caption:


```
Public Sub Main()
   Dim sName As String
   Dim fColor As Long, bColor As Long
   Dim sCaption As String

   ' Required in all MConsole.bas supported apps!
   Con.Initialize

   ' Stash value(s) we'll later reset.
   bColor = Con.BackColor
   fColor = Con.ForeColor
   sCaption = Con.Title

   ' Read and write a simple response:
   If Con.Height < 50 Then Con.Height = 50
   Con.ForeColor = conGreenHi
   Con.WriteLine "What's your name? ", False
   Con.ForeColor = fColor
   sName = Con.ReadLine()
   Con.ForeColor = conGreenHi
   Con.WriteLine "Hello " & sName, False
   Con.Title = "Console Demo for " & sName

   ' Restore original console colors and caption.
   Con.BackColor = bColor
   Con.ForeColor = fColor
   Con.Title = sCaption
End Sub
```

Of course, redirection and pipes are fully supported as well. Here's a snippet showing how to read standard input, and send it directly to the clipboard. (This is the heart of a neat little utility I wrote that allows you to capture the output of any console application and place it on the clipboard.) This snippet also demonstrates output of debugging information, output to standard error, and the issuance of an exitcode upon completion:


```
Public Sub Main()
   Dim sData As String
   Dim sMessage As String

   ' Required in all MConsole.bas supported apps!
   Con.Initialize

   ' Check to see if we have any waiting input.
   If Con.Piped Then
      ' Slurp it all in a single stream.
```

```
      sData = Con.ReadStream()
      ' Just to prove we did it, place text on clipboard.
      Clipboard.Clear
      Clipboard.SetText sData
      ' Write some debugging information.
      Con.DebugOutput "Wrote " & CStr(Len(sData)) & _
                      " characters to clipboard."
   Else
      sMessage = "No redirection detected; nothing to read?"
      ' Send error condition to Standard Error!
      Con.WriteLine sMessage, True, conStandardError
      Con.DebugOutput sMessage
      ' Set an exit code appropriate to this error.
      ' Application MUST BE COMPILED TO NATIVE CODE to
      ' avoid a GPF in the runtime!!!
      Con.ExitCode = 1
   End If
End Sub
```

There sample is probably one of the most comprehensive available on this site. There are uncountable uses for it, as it covers just about every conceivable aspect of console application authoring. Please let me know if you feel I've overlooked anything!

Required Modification of Executable

The only difference between a console application and a windowed application, is a single bit in the Portable Executable (PE) file header which is set during the compilation process. Classic VB doesn't natively support setting this PE attribute. There are several easy ways to to approach this limitation. The oldest method, first published by L.J. Johnson, is to toggle a bit in the PE header of your executable following compilation. This alerts Windows to start it up as a console, rather than windowed, application. The extra step can be a real hassle, though. Matt Curland's book, referenced above, includes a free add-in that modifies the flags passed to the compiler, so that it produces a console application for you. My favorite though, which offers the same capability, is to use the vbAdvance add-in, which offers many other unique capabilities in addition to this vital one.

Another option is to post-process the EXE, immediately after compilation. Bob Riemersma has offered a short VBS script that makes this process incredibly simple. Just copy this text to a file named LinkConsole.vbs either on your desktop or in the project directory:

```
' LinkConsole.vbs
' http://www.angelfire.com/mi4/bvo/vb/vbconio.htm
'
' This is a WSH script used to make it easier to edit
' a compiled VB6 EXE using LINK.EXE to create a console
' mode program.
'
' Drag the EXE's icon onto the icon for this file (or onto
' a shortcut to this file), or execute it from a command
' prompt as in:
'
'    LinkConsole.vbs Project1.exe
'
Option Explicit

Dim strLINK, strEXE, WSHShell

' Be sure to set up strLINK to match your VB6 installation.
strLINK = """C:\Program Files\Microsoft Visual Studio\VB98\LINK.EXE"""

strEXE = """" & WScript.Arguments(0) & """"

Set WSHShell = CreateObject("WScript.Shell")

WSHShell.Run strLINK & " /EDIT /SUBSYSTEM:CONSOLE " & strEXE

Set WSHShell = Nothing
WScript.Echo "Complete!"
```

Then, after compiling your project, drag the resulting EXE and drop it on the VBS (or a shortcut to it). All done. Your EXE is now set to run within the console subsystem. *(Be sure to edit the path to LINK.EXE if you installed VB6 to a nonstandard location.)*

Lightweight Console Object Methods

Initialize	Must be the first method called. Creates the lightweight object, which is used for automated tear down support. Determines the initial state of numerous other

	properties, and obtains standard IO handles for input, output, and error.
DebugOutput	Writes debugging information to both the Immediate window and through a call to OutputDebugString so that it can be read even in a compiled application.
FlashWindow	Flashes console window titlebar and taskbar icon to get users attention. 
Flush	Flushes the console input buffer of any waiting input records.
ReadChar	Reads a single character of user input, without displaying it, and immediately returns. 
ReadLine	Reads one line of input from user. Optionally writes prompt to screen.
ReadPassword	Obscures user input behind a specified password character. 
ReadStream	Reads all available input records.
Resize	Sets new height and width values for console text buffer.
PressAnyKey	Prompts the user to "press any key" to continue, and waits for them to do so.
SetFocus	Sets focus, optionally with force, to the console window.
ShowCursor	Toggles visibility of console cursor, in an identical manner to that of the ShowCursor API.
WaitingInput	Signals that input records are waiting to be read.
WriteLine	Writes a line of text to either standard output or standard error. Optional alignment support offered.

Lightweight Console Object Properties

BackColor	Background color used for new console output.
Break	Flag value that indicates the user has attempted to manually break execution of the console application, by pressing Control-C or similar. *See: ControlEvent property.*
BufferHeight	Height of text buffer, in lines, for current console.
BufferWidth	Width of text buffer, in characters, for current console.
CodePage	Codepage of current console.
ControlEvent	*Read-only:* Indicates what sort of event occurred that requires the application to shutdown. Possibilities include Control-C, Control-Break, Close [X], Logoff, and Shutdown. *See: Break property*
Compiled	*Read-only:* Flags whether the application is compiled or running in the IDE.
CurrentX	X position for next output.
CurrentY	Y position for next output.
CursorHeight	Height of cursor, in scan lines.
CursorVisible	Current state of cursor visibility.
ExitCode	Exitcode to emit when process completes.
ForeColor	Foregoundcolor used for new console output.
FullScreen	Toggles between full-screen and windowed mode, or indicates current status.
Height	Height of current console, in lines.
hStdErr	*Read-only:* Handle to standard error.
hStdIn	*Read-only:* Handle to standard input.
hStdOut	*Read-only:* Handle to standard output.
hWnd	*Read-only:* Handle to console window.
LaunchMode	*Read-only:* Indicates whether the application was launched from within the IDE, directly from the command line, or from Explorer via a shortcut or double-click.

ParentFilename	*Read-only:* Returns the filename of the process that launched the application.
ParentProcessID	*Read-only:* Returns the process ID of the process that launched the application.
Piped	*Read-only:* Indicates whether piped or redirected input is waiting to be read from standard input. Test this value just once, at application initialization, to determine whether to proceed as a filter or interactively.
Redirected	*Read-only:* Indicates whether standard output from this application has been redirected.
TaskVisible	Current visibility of task, including whether it can be seen as a separate application within Task Manager.
Title	Caption of console window.
Visible	Visibility of console window.
Width	Width of current console, in characters.
WindowState	Normal, minimized, or maximized.

Published

This sample, or the one from which it originally derived, was published (or at least peripherally mentioned) in the following article(s):

- **The Only Add-In Always Used**, Classic VB Corner, *VSM Online*, June 2009
- **Persisting Environment Variables**, Classic VB Corner, *VSM Online*, August 2009

APIs Usage

This sample uses the following API calls:

Module	Library	Function
Demo-Codepage.bas	kernel32	Sleep
Demo-EnvChange.bas	advapi32	RegCloseKey RegFlushKey RegOpenKeyEx RegSetValueEx
	shlwapi	SHDeleteValue
	user32	SendMessageTimeout
Demo-ExitCode.bas	kernel32	Sleep
Demo-Interactive.bas	kernel32	Sleep
Demo-Parent.bas	kernel32	Sleep
MConsole.bas	kernel32	AllocConsole CloseHandle CreateToolhelp32Snapshot ExitProcess FlushConsoleInputBuffer FreeConsole FreeLibrary GetConsoleCP GetConsoleCursorInfo GetConsoleDisplayMode GetConsoleMode GetConsoleOutputCP GetConsoleScreenBufferInfo GetConsoleTitle GetConsoleWindow GetCurrentProcessId GetCurrentThreadId GetFileSize GetLogicalDriveStrings GetModuleFileName

		GetModuleHandle
		GetNumberOfConsoleInputEvents
		GetProcAddress
		GetStdHandle
		GetVersionEx
		LoadLibrary
		OpenProcess
		OutputDebugString
		PeekConsoleInput
		Process32First
		Process32Next
		QueryDosDevice
		ReadConsole
		ReadConsoleInput
		ReadConsoleOutput
		ReadConsoleOutputAttribute
		ReadConsoleOutputCharacter
		ReadFile
		ReadFileEx
		RtlMoveMemory
		SetConsoleActiveScreenBuffer
		SetConsoleCP
		SetConsoleCtrlHandler
		SetConsoleCursorInfo
		SetConsoleCursorPosition
		SetConsoleDisplayMode
		SetConsoleMode
		SetConsoleOutputCP
		SetConsoleScreenBufferSize
		SetConsoleTextAttribute
		SetConsoleTitle
		SetConsoleWindowInfo
		SetStdHandle
		Sleep
		SleepEx
		WriteConsole
		WriteConsoleOutput
		WriteConsoleOutput
		WriteConsoleOutputAttribute
		WriteConsoleOutputCharacter
		WriteFile
		WriteFileEx
	ntdll	NtQueryInformationProcess
	psapi	GetModuleFileNameEx
		GetProcessImageFileName
	user32	AttachThreadInput
		CharToOemBuff
		EnumThreadWindows
		FindWindow
		FlashWindowEx
		GetForegroundWindow
		GetSystemMenu
		GetWindowLong
		GetWindowThreadProcessId
		IsIconic
		IsWindowVisible
		IsZoomed
		OemToCharBuff
		RemoveMenu
		SetForegroundWindow
		ShowWindow
MEnvVars.bas	advapi32	RegCloseKey
		RegFlushKey
		RegOpenKeyEx

	RegSetValueEx
shlwapi	SHDeleteValue
user32	SendMessageTimeout

Don't see what you're looking for? Here's a complete API cross-reference.

Download

Please, enjoy and learn from this sample. Include its code within your own projects, if you wish. But, in order to insure only the most recent code is available to all, I ask that you don't share the sample by any form of mass distribution.

Download Console.zip, *95Kb, Last Updated: Wednesday, February 17, 2010*

See Also

The following resources may also be of interest:

- **ConClip** - Console utilities to read/write the Windows clipboard.
- **KeyStuff** - Stuff the keyboard buffer from your console applications.
- **Uptime** - Determine how long since the computer was last rebooted.
- **Which** - Finds first or all executables on path, or which executable is associated with a document file or extension.

Do This site has been designed with and for Firefox.
It may work in Internet Explorer as well.