November 30 – December 3, 2004 ◊ Las Vegas, Nevada

Automating Boring Mundane Tasks Using WindowsScriptHost

Gordon Price – albedoConsulting

CP33-1 A continuation of last year's WSH class; this time around we focus more on AutoCAD® -specific administration and WSH-based AutoCAD automation, culminating in a script that builds an entire set of architectural base files, cross-xrefed, layer-controlled, and ready to use. Don't let the architectural reference deter you, these scripts can be adapted for use by anyone who finds them selves setting up projects the same way every time. And again, because we will be using VBScript, everything you learn can be applied to programming AutoCAD or MS Office in VBA, or even standalone applications in VB. This session is designed for CAD and IT managers who want to use WindowsScriptHost and VBScript to help install and manage AutoCAD and AutoCAD-based products, and who want to add functionality to help users manage projects and files. Users with an interest in customization who want another way to make the computer do more will also benefit from this session.

Who Should Attend
CAD and IT managers, and intermediate-level users

Topics Covered
* WSH-based tools for automating everyday tasks
* WSH-based management of current AutoCAD installs, via Remote Scripts and Logon Scripts
* Script-based installs compared to the Network Installation Wizard
* Accessing lots of DWGs quickly with the AutoCAD DBX API
* WSH-based tools to automate schemes, xref management, and more

About the Speaker:
An AutoCAD® user since 1990 and a consultant since 1994, Gordon has most recently been the CAD manager at Thomas Hacker Architects in Portland, Oregon. In October he returned to the wild world of consulting full time, focusing on small-offices. His architectural experience includes work as a designer, job captain, and web developer. He is also currently finishing his M.Arch at San Francisco Institute of Architecture.

www.albedoconsulting.com
gordon@albedoconsulting.com

What is included in this class?

- A quick general overview of WSH, VBScript and related components you are likely to make use of writing basic System Administration and CAD Administration scripts, as well as scripts to automate certain user tasks.
- Information on the different ways to interact with WSH based tools, and how to make them available to users.
- A quick look at demos and example code for a couple of useful CAD Administration and User scripts.
- Demos of some more complex scripts which are too large to print for the AU handout (but which can be downloaded after the conference)

What we won't be covering includes WSC (Windows Script Components), ADO (database access), ADSI & WMI (for advanced Windows system admin) or issues of connecting to applications like Word or Excel.

What is WSH?

In a way, the better title for this class might have been **Windows Script Technologies**. This encompasses the whole gamut of scripting tools and components, of which…

WSH (Windows Script Host) is just that, a HOST environment in which scripts in various languages can execute. Internet Explorer is also a script host, as it can implement client side scripting. However, for security reasons IE client side Scripting cannot access anything on the local machine or domain, and so is useless as an administrative tool. IIS is also a script host, in that it supports ASP (Active Server Pages), but ASP is also limited, this time to connecting with local databases and modifying the HTML stream going to the receiving browser. Again, not much use for administrative purposes. WSH was created specifically for System Administrators as an upgrade and replacement for DOS BAT (batch) files. However, WSH is incomplete by itself. You also need…

VBScript, or another scripting language, which interacts with WSH via WindowsScript (part of the OS). The scripting language gives you the basic programming language to create scripts, but VBScript alone has no way to interact with useful parts of the OS, such as the File System. For that you use the ability of WSH to access…

COM objects (Component Object Model). Applications that are COM enabled expose their internal functions to outside control. By using WSH to connect to a COM object, you gain access to whatever functionality that object exposes. Some examples of useful COM objects include…

Scripting.FileSystemObject – Manipulate files & folders.
ADO (ActiveX Data Objects) – Connect to a database.
CDO (Collaborative Data Objects) – E:mail functions without Outlook or Exchange.
ADSI (Active Directory Services Interfaces) – Network Administration, including ActiveDirectory, Exchange, InternetInformationServer, etc.
WMI (Windows Management Instrumentation) – Local system administration, user interface, desktop settings, etc.
Shell.Application – Windows shell functions, including the BrowseForFolder dialog, Move & Copy functions with status indication, etc.
AutoCAD, **MS Word**, etc. The standard application COM objects generally give you access to most all features and commands available in the software, and can be manipulated either with the application window visible or not.
AXDB15Lib – This activeX dll allows you to very quickly access AutoCAD DWG files, but with the trade off of loosing access to any user interface features. It is useful when you need to modify a large number of drawings quickly.
Some objects available only within the WSH environment, but which you will use much like COM objects are…

WScript.Shell – Run (start) applications, create shortcuts, manipulate environment variables and the Windows Registry.
WScript.Network – Manipulate network resources (shared printers & folders)

Also, you can use **WindowsScriptComponents** to create your own light weight COM objects, to securely encapsulate business data in reusable code, create IE DHTML Behaviors, etc.

WSH/VBScript versions

Windows 98	1.0 (client side scripting in IE only)
Windows NT (part of SP4)	1.0
Windows ME	2.0 (WSH introduced, common version numbers)
Windows 2000	2.0
InternetExplorer 5.5	5.5 (VBScript only, WSH is still 2.0)
IIS 5	5.5
Windows XP	5.6 (Shared version numbers again)
Windows Server 2003	5.6

NOTE: Make sure you upgrade to WSH/VBS 5.6, especially in a mixed environment where you might run scripts on Windows XP Pro & Windows Server 2000. Version 5.6 includes Regular Expressions, which alone is worth the (free) upgrade. Version 5.6 also includes RemoteScripting (running scripts on other machines), code signing for security, named Arguments (with WSF files only) and StandardI/O for interacting with many Command Line tools. You can download 5.6 for free.

Some VB terms & definitions

- **Object** – An item that has Properties and/or methods. An AutoCAD block is like an object.
- **Class** – Defines an type of object (the object is instantiated in VBese). An AutoCAD block definition is similar.
- **Property** – A setting of an object that can be read or applied. Similar to the value of an attribute in a block. Properties can control how an object behaves, how it looks, etc.
- **Method** – A task or command that an object can perform. An Xref has an Unload method.
- **Event** – A notification that something has happened, caused by the user, the OS or the software. You can have your own Subroutine react to a specific event.
- **Subroutine** – A reusable bit of code that does a certain task. You can pass arguments to a subroutine.
- **Function** – A Subroutine that returns a value is a function.

Comparison of VB/VBA & VBScript

For our purposes VB & VBA are the same. There are actually some differences, but you will discover those if you move on to VB/VBA programming. In general VBScript is a stripped down, simplified version of the others. Specifically…

- VBScript is 'weakly typed'. In other words all variables are Variants, with the actual content being of a particular 'subtype'. The actual content of a variable can be determined, but any variable can contain any type of data. This can lead to runtime errors if you are not careful.
- VBScript does not have an IDE (Integrated Development Environment). VBA & VB have IDEs similar to the VisualLisp IDE in AutoCAD.
- VBScript does not support 'early binding'. In the VB or VBA IDE you can 'bind' objects as you code (early) and the IDE IntelliSense will provide you with the properties & methods of that object as you code. VBScript doesn't bind until you run the code (late). This again can lead to runtime errors. Since there is no VBScript IDE, early binding would serve no purpose.

- VBScript does not support 'named arguments'. In VB and VBA you can pass and receive arguments by name, but VBScript only supports 'positional arguments', so you have to make sure you pass your arguments in the right order. Note that WSH does support named arguments passed to the script, but between internal routines and functions you are limited to positional arguments.
- VBScript has limited error handling. VB/VBA has the *On Error Goto Label* statement, which is similar to error handling in Lisp. VBScript only has *On Error Goto Next*, which forces you to deal with an error in the very next line of code.
- VBScript is slower that VB/VBA. This is because VBScript is not compiled code. Speed is relative however, as this is only a comparison of the same code in VBA as compared to VBScript. When you compare the time it takes to do a task manually, which is what you will use WSH to automate, then VBScript is lightning fast.
- VBScript doesn't support a number of VB/VBA features such as Optional arguments, Option Base for Arrays, etc. See the VBScript Help file for more details.
- In the context of WSH, VBScript is not generally 'event driven' so much as 'task driven'. Most scripts will set out to do something, perhaps based on a passed argument, or user input, or perhaps not. It is possible to create scripts that listen for events, you can sink events from IE for example, and an HTA is fully event driven, but for the majority of WSH scripts the only 'event' is starting the script.
- VBScript has Regular Expressions, which are NOT included in VB/VBA.

Scripting & Security

Given the prevalence of viruses, and especially script based viruses, some discussion of script security is required. Of course good anti virus software and regular virus definition updates are assumed, but this does not protect you from viruses not yet identified by the anti virus companies, nor does it protect you from staff running scripts that are not viruses but that you still don't want being used.

The details of scripting security are beyond the scope of this class, but some basic ideas are worth noting.

- Use WSH & VBScript 5.6, as this version has much better security capabilities.
- Windows XP is also much more secure, and flexible in application of security, than older versions of Windows, especially Windows 9X.
- If you already have a digital certificate you can use this to sign your own scripts. If you don't already have one, you can get buy one from Verisign or another Certificate Authority, or get one from your own Certificate Server.
- If you use Windows XP you can limit script execution to specific read only folders using Software Restriction Policies. If you are running Windows Server 2003 you can manage Software Restriction Policies using Group Policy.

Other Script Engine options

For the purposes of this class we will focus on VBScript as our scripting language. This has the added benefit of making what you do in WSH a learning opportunity for VB/VBA. However, there are things that other scripting languages do better (text handling in Perl and true compiled code in Jscript for example) and if you have a task that VBScript isn't up to, by all means look at other scripting languages. With WSH 5.6 you can access a subroutine or function written in one language from a main script written in another, so you could tie together a Perl subroutine and JScript function with a VBScript main script. Those other components might even be things you downloaded or had someone else write. That said, some of the other Script Language options are:

- Jscript (aka JavaScript, aka ECMAScript, included with WSH)
- Perl

- Python
- Rexx

File Types

WSH scripts can be implemented either in language dependent files (simple) or XML based language independent files (more complex, and more powerful). You will probably use both eventually. These files include…

VBS – Basic language dependent script file containing only VBScript code. A VBS file (and any language dependent file) can contain script in Functions and/or Subroutines that can be called from within the script, or from another script in a WSF file. A language dependent file can also contain script code that is not part of a Function or Subroutine, and is executed when the file launched.

JS – Basic language dependent script file containing only JScript code.

PLS – Basic language dependent script file containing only PerlScript code.

WSF – Language independent XML based WindowsScriptFile which can contain a single script (and Subroutines/Functions) in any language, or multiple scripts in multiple languages, each identified as a different 'job', all tied together with XML to make it work. Each job within the script can be called independently via WScript or CScript. Only a WSF file can reference 'source' files, also known as 'includes'. Source files allow you to reuse code by having multiple jobs or multiple WSF files reference the same 'source' files and make use of the Subroutines & Functions found there. Only language dependent files can be used as source files. Note that routines in one job are not available to other jobs. Source files are the only way to share code between multiple jobs or files.

Code Execution

There are a number of different ways to actually execute script code.

- CScript (Command Script) WSH execution via the Command Prompt – When a script will execute in the background transparent to the user, use CScript. With CScript you can use command line switches to run in batch mode with error messages and user prompts suppressed, run in debug mode, etc. Also, when running via CScript WScript.Echo commands will echo to the Command Prompt (rather than multiple msgBox windows), so lots of echos are easier to deal with. Lastly, if you want to schedule a script using Windows Scheduled Tasks (which usually means no user intervention) then you will want to use CScript. CScript is usually best used for System Administration type scripts, such as a scheduled script to parse a series of log files looking for certain types of items and leaving the compiled information in a text file for you to look at the next day. For scripts that a user will need to interact with you will probably use WScript. Using CScript will result in the Command Prompt being visible for the duration of the script, even if the actual execution of the script was started with a shortcut. In general, CScript is much more like BAT file execution.
- WScript (Windows Script) WSH execution via the Windows interface – If you don't specifically need the silent execution of CScript or the ability to echo to the Command Prompt you will probably use WScript. Unless you specifically change file affiliations, all WSH files execute with WScript by default for this reason.

User Interface

While CScript & WScript offer the Command Prompt and the messageBox & inputBox functions, you will sometimes need more UI than this. On these occasions you have two options.

- **Internet Explorer** – Because IE is a COM enabled application, you can use WSH to drive IE externally, providing a much more robust place to dump text than either the Command

Prompt of CScript execution or a messageBox. You can use IE in this way for both user input and output, however, there are some limitations. Because the script and IE are running separately, if you close the browser window the script continues unless you include code to terminate the script as well. If you don't terminate the script then the lack of an IE object can cause errors, not to mention ruining the user experience. The basic page design can be done in FrontPage or another HTML editor, but you have to manage the HTML stream for any dynamic aspects, and your application now involves two files, the actual script and an HTM file for the UI. To avoid this you can use…

- **HTA** – An HTA (HyperText Application) is simply an HTML document with the extension changed to HTA. The HTA file contains all the required HTML code to build the page, as well as the script code you would have put in a VBS or WSF file. In an HTM file, you have two things to deal with, the full IE user interface that has no relevance (the Forward & Back buttons for example) and the fact that for very good security reasons the script code in a web page can't access many objects (the FileSystemObject for example) and many things that you can do include a security warning for the user. In HTA mode, the entire IE user interface is stripped away, leaving the HTML window for you to create your own interface, and all code that would work in a VBS file will work in an HTA file. And because you are now building your script into HTML, your scripts become fully event driven. Lastly, you can use FrontPage, or even MS Word in a pinch) to create your HTM file, then add your script and change the extension. HTA allows you to create full featured applications with a rich user interface, all with tools you already have at your disposal.

User Access

users should not be getting direct access to your scripts, but rather using shortcuts (LNK files) to your scripts or similar methods to keep users one step removed from the actual script files. Depending on how the script is going to be used, there are a number of ways of giving your users access to those shortcuts.

Working with files & folders

When the user will pick files or folder to be dealt with in some way, you can…

- **Drag'n'drop to shortcut –** Provide your user with a shortcut (on the desktop, on a network share, etc.) to which they drag'n'drop their selected files or folders to be handled by the script. Because the user could select the wrong kind of item your script must deal with this, as well as when your user just clicks the shortcut (i.e. nothing sent as an argument to the script)
- **SendTo** – Put your shortcut in the user's SendTo folder. This is similar to drag'n'drop, with a common, logical location for the script shortcuts. Note that all the default SendTo functionality actually sends your selected files or folders to another location. If your script just manipulates the item where it is, the SendTo option can be confusing for users. Also remember that the SendTo folder is unique for each user, and there is no All User SendTo. And if you don't have Roaming Profiles enabled a user will not have the SendTo options they expect when they change machines, unless you standardize all machines with the same SendTo options.
- **Windows Context Menu** – when your script is intended to only handle items of one type (folders, or BAK files, for example) you can add your script as an 'action' for that file type, which then becomes available as a right click Context Menu option (and could be the Default action as well). Windows handles limiting file types, and makes a separate call to your script for each item, so you don't have to deal with looping through arguments. Because this is implemented in the HKEY_CLASSES_ROOT section of the registry they are available to all users on the machine, and are not part of a particular user's roaming profile. Look at my web site soon for a white paper with more detail about this.

Working like a traditional application
When your script will act like a traditional application, you can…

- **Shortcut on a network share** – Quick and easy.
- **Integrate with the OS** – Here you place a shortcut to the script on the desktop, in the Startup menu or in the QuickLaunch toolbar. Where the shortcut goes depends on script usage and perhaps user preference.
- **Integrate with another application** – Most applications have the ability to run an external application. For example your script could be launched with a toolbar button in Word or AutoCAD. An 'AutoSave Recover' application might be a good candidate.

Working like a Service
Sometimes you have a script that needs to monitor a situation and react to changes. In this case you want your script to emulate a Windows Service, quietly waiting for the proper situation and then acting. While a script cannot be a true service you can get the effect by placing your code in an intentional eternal loop. The script will seem to wait quietly, checking the situation with each cycle of the loop, and taking action when appropriate. It is usually a good idea to use the Wscipt.Sleep method to manage your loop, as it takes up very little processing power, and allows you to control your script like a timer. Because you are using an endless loop, you need to provide some mechanism for stopping the script. The sledge hammer approach is to kill the script with Task Manager. Another more graceful approach is to monitor the value of an environment variable, and exit the loop when the variable changes. Then create another script to change the variable. You run the second script to stop the first.

As a Scheduled Task
You can use Task Scheduler to run your scripts at preset times. You can even use scripts to setup the scheduled task.

Run at Startup
If you need your script to run when a user starts a machine or logs on, you can do it a number of ways.

- You can run your scripts from your logon BAT file, or even replace the BAT file with a VBS logon file, which then calls your scripts that you want run.
- You can add a LNK to your script to the user Startup folder. This can be used to mimic Domain Logon Scripts on a Peer to Peer network.
- You can edit the registry to add your script as a run once item. This happens before logon, which you sometimes need.

Script access installation

There are a number of different ways to roll out your scripts for users, depending on you network setup and how you intend to make the scripts available to users.

- Place script shortcuts on a network share. This is by far the simplest method as once you add a shortcut to the shared folder your users have instant access. However, it is not particularly convenient, and not at all integrated with the OS and other applications.
- 'Install Scripts' can be used to copy script shortcuts to the user's SendTo, Startup & Start Menu folders, as well to the QuickLaunch toolbar or the Desktop. An install script can also be used to set up Shell Extensions and scheduled tasks. You can make install scripts available on the network, or you can e:mail them (yes, you can modify the Outlook Security settings with a script), or run them manually yourself from each machine.

- Logon scripts can be used to run install scripts. This works best if you use a text file or database to log who has successfully run your install, so that the install procedure is not run more than once for each user, and so you know when you can remove the reference to the install script from the logon script. And of course logon scripts can be VBS files now. One trick is to add some code to your VBS based logon script that looks in a particular network location for a shortcut called username.LNK on the network which points to the actual install script and runs it if found. The install script then looks for the same shortcut and deletes it, after successfully completing the install. The administrator simply adds LNK files for each user that needs an install. When all the installs have completed all the LNK files will be gone, ready for the next round of installs. The effect is much like the Run Once registry setting.
- Remote scripts. You can use WSH to run an install script on a remote machine. From your Admin machine you could loop through a list of all the machines in your office, running the install script on each one in sequence. The limitation is the user running the 'controller' script must have local admin rights on the remote machines running the 'worker' script, and a registry setting must be made on each machine that will run a 'worker' script. Once this setup is done, Remote Scripts can offer a lot of other nice features.

If you are running Windows Server you can do some other tricks that make installation and management easy.

- You can map the Desktop (and My Documents, by the way) to a network location. Now you can just copy a shortcut to everyone's Desktop folder (with a script, of course) from the server. This also makes it easy to back up everyone's Desktop and My Documents folders, though you need room on the server for this extra data.
  One other nice trick here. Since the Desktops are now on the server, you could add a SendTo 'Other User Desktop' option via WSH. The script could give a list of desktops and the user selects which one or ones the file goes to, and those users will see the file beam down onto their desktop. At a minimum you administrators should be able to dump stuff to your users' desktops.
- With Roaming Profiles you have similar access to the users' SendTo, Startup, Start Menu and QuickLaunch folders. The users won't see added tools until their next logon, however. With Roaming Profiles you can use the 'Script deletes its own shortcut' trick by putting the shortcut in the Startup folder for each user.

One trick for System Administrators to think about. If you find yourself with a lot of admin scripts, try this. Add your network script share to the PATH environment variable for the Administrator user, or add some code to the Administrator logon script to add your script folder to the Volatile (current session only) PATH environment variable. Now when you logon as Administrator you can run your scripts from the Command Prompt with no path!

Longhorn, .NET & MSH

Monad is the code name for MSH (MicrosoftScriptHost?), the .NET flavor of WSH to be included in the next version of Windows. Actually it is the Longhorn Shell, so it encompasses both WSH type functionality, along with command line stuff, and is integrated with .NET. In short, WSH is going to get more powerful, and yes more complicated, but it is not going to die. And MSH will allow you to learn .NET stuff on the cheap, which you can then apply to learning VB.NET programming later, much like your VBScript knowledge can later be applied to learning VBA/VB.

Conclusion

I hope this class has provided you with some scripts that you can take back with you and apply immediately, as well as some ideas as to how you could make use of WSH to develop your own scripts to deal with boring and mundane tasks, leaving you more time to do something interesting.

Thank you.

Further Reading

VBScript In A Nutshell, 2nd Ed.
Paul Lomax, Matt Childs & Ron Petrusha
O'Reilly & Associates
Overview and reference for VBScript 5.6 with some info on using VBScript in WSH, as well as InternetExplorer

Microsoft Windows 2000 Scripting Guide
The Windows Resource Kit Scripting Team
Microsoft Press
Actually covers WSH/VBScript 5.6, even if it is a 'Windows 2000' book. Good coverage of WSH, ADSI, WMI etc. Also includes accessing the Windows Shell, Windows API, and basic good programming conventions.

Microsoft Office XP Developers Guide
Microsoft Press
MS Office VBA, but gives you the basics for driving these apps from WSH

Managing Enterprise Systems with the Windows Script Host
Stein Borge
Apress
Good overview of basic WSH, ADSI & WMI scripting

Newsgroups
@ msnews.microsoft.com
microsoft.public.scripting.vbscript
microsoft.public.scripting.wsh
microsoft.public.server.scripting

Web sites

Microsoft® Windows®2000 Scripting Guide
www.microsoft.com/resources/documentation/windows/2000/server/scriptguide/en-us/sagsas_overview.mspx

Windows XP Software Restriction Policies whitepaper
www.microsoft.com/technet/prodtechnol/winxppro/maintain/rstrplcy.asp

MSDN Scripting web site (downloads, help files and more)
msdn.microsoft.com/scripting

Clarence Washington's Script Repository (win32scripting)
cwashington.netreach.net

WindowsScript at MSN Groups
groups.msn.com/windowsscript/

My website, for demo downloads and future info
www.albedoconsulting.com

```
'*******************************************************************************
'* Script Name:   bak2dwg_sendto.vbs
'* Created:       8.24.1999
'* Author:        Gordon Price
'* Purpose:       Convert passed list of BAK files to DWG files
'* Usage:         Place shortcut to script in SendTo folder
'*                User selects files, right clicks and chooses
'*                shortcut from SendTo menu
'* History:       09.20.2003 - Revised by GP to use For Each loop
'*******************************************************************************
Option Explicit
'*******************************************************************************
'* Variables
Dim objArguments
Dim objFileSystem
Dim objSourceFile
Dim strArgument
'*******************************************************************************
'* Main Script
' create basic objects
Set objArguments = WScript.Arguments
Set objFileSystem = CreateObject("Scripting.FileSystemObject")
' loop thru the arguments list
For Each strArgument In objArguments
   ' start Error trap
   On Error Resume Next
   ' set source file to the next argument
   Set objSourceFile = objFileSystem.GetFile(strArgument)
   ' if no Error then we have a file, not a folder or missing file
   If Err.Number = 0 Then
      'if the passed object is a BAK file copy to DWG & increment success counter
      If objSourceFile.type = "BAK File" Then objSourceFile.Name = objSourceFile.Name & ".dwg"
   End If
' clear the Err object and continue For Each loop
Err.Clear
Next


'*******************************************************************************
'* Script Name:   bak2dwg_shell.vbs
'* Created:       8.24.1999
'* Author:        Gordon Price
'* Purpose:       Convert passed BAK file to DWG file
'* Usage:         Implement via WindowsShellExtension
'*                (See ShellExtensions.doc)
'* History:       09.20.2003 - Revised by GP to use For Each loop
'*                10.27.2003 - Revised by GP for ShellExtension usage
'*******************************************************************************
Option Explicit
'*******************************************************************************
'* Variables
Dim objArguments
Dim objFileSystem
Dim objSourceFile
'*******************************************************************************
'* Main Script
' create basic objects
Set objArguments = WScript.Arguments
Set objFileSystem = CreateObject("Scripting.FileSystemObject")
' start Error trap
On Error Resume Next
' set source file to argument(0)
Set objSourceFile = objFileSystem.GetFile(objArguments(0))
If Err.Number = 0 Then objSourceFile.Name = objSourceFile.Name & ".dwg"
```

```
'*************************************************************************
'* File Name:   SendToFavorites.vbs
'* Created:     10.13.2003
'* Author:      Gordon Price
'* Purpose:     From passed list of objects; add LNK fles to user specified
'*              folder in Favorites. For LNK files, move file; for other files,
'*              create File LNK; for folders, create folder LNK.
'* Usage:       Place shortcut to script in SendTo folder
'* History:
'*************************************************************************
Option Explicit

'*************************************************************************
'* Constants
Const WINDOW_HANDLE = 0
Const NO_OPTIONS = 0

'*************************************************************************
'* Variables
Dim objArguments
Dim objFileSystem
Dim objWSHShell
Dim objWindowsShell
Dim objShortcut
Dim objFolder
Dim objFolderItem
Dim objFile
Dim strFavoritesPath
Dim strFolderPath
Dim strArgument
Dim strLinkPath
'*************************************************************************
'* Main Script
Set objArguments = WScript.Arguments
Set objFileSystem = CreateObject("Scripting.FileSystemObject")
Set objWSHShell = CreateObject("WScript.Shell")
Set objWindowsShell = CreateObject("Shell.Application")

' get the User's Favorites folder
strFavoritesPath = objWSHShell.SpecialFolders("Favorites")

' user input: folder in Favorites to contain new LNKs
Set objFolder = objWindowsShell.BrowseForFolder (WINDOW_HANDLE, "Select a folder: ", NO_OPTIONS,
strFavoritesPath)
Set objFolderItem = objFolder.Self
strFolderPath = objFolderItem.Path

' loop through arguments
For Each strArgument In objArguments
   If objFileSystem.FileExists(strArgument) Then
      If (LCase (Right (strArgument,4))) = ".lnk" Then
         objFileSystem.MoveFile strArgument, strFolderPath & "\"
      Else
         Set objFile = objFileSystem.GetFile(strArgument)
         strLinkPath = strFolderPath & "\" & objFile.Name & ".lnk"
         Set objShortcut = objWSHShell.CreateShortcut(strLinkPath)
         objShortcut.TargetPath = strArgument
         objShortcut.Save
      End If
   End If
   If objFileSystem.FolderExists(strArgument) Then
      Set objFolder = objFileSystem.GetFolder(strArgument)
      strLinkPath = strFolderPath & "\" & objFolder.Name & ".lnk"
      Set objShortcut = objWSHShell.CreateShortcut(strLinkPath)
      objShortcut.TargetPath = strArgument
      objShortcut.Save
   End If
Next
```

```
'*************************************************************************
'* Script Name:   folderbuilder.vbs
'* Created:       08.25.1999
'* Author:        Gordon Price
'* Purpose:       Create & populate new project folder structure based on last
'*                project number and user input project name
'* Usage:         Place shortcut to script on Desktop or in Start Menu
'*                Modify Set objActiveJobsFolder line for office folder structure
'*                Script does not address absent root folder, or empty folder
'* History:
'*************************************************************************
Option Explicit

'*************************************************************************
'* Variables

Dim objFileSystem
Dim objActiveJobsFolder
Dim objSubFolderList
Dim objFolder

Dim intCurrentYear
Dim intLastJobYear
Dim intLastJobNumber

Dim strActiveJobsFolder
Dim strLastFolder
Dim strNextJobNumber
Dim strJobName
Dim strJobFolder

'*************************************************************************
'* Main Script
Set objFileSystem = CreateObject("Scripting.FileSystemObject")
Set objActiveJobsFolder = objFileSystem.getFolder("C:\_Projects\Active Jobs")
Set objSubFolderList = objActiveJobsFolder.subFolders

' get the current year
intCurrentYear = Year(Date)

' get the year and job# of the last job folder created
' this only works if the only folders in the Active folder
' are proper job folders
' at the end of the loop strLastFolder is the last folder created
For each objFolder in objSubFolderList
   strLastFolder = objFolder.Name
Next

' extract the year & job#
intLastJobYear = CInt (Mid (strLastFolder,1,4))
intLastJobNumber = CInt (Mid (strLastFolder,6,2))

' set the next job#
' if the next job# will be the same year as the last then increment job#
if intCurrentYear = intLastJobYear then
   strNextJobNumber = CStr(intLastJobNumber + 1)
   if Len(strNextJobNumber) < 2 then strNextJobNumber = "0" & strNextJobNumber
   strNextJobNumber = CStr(intCurrentYear) & "." & strNextJobNumber
' otherwise start a new year with job# 01
else
   strNextJobNumber = CStr(intCurrentYear) & ".01"
end if

' get the project name from the user
strJobName = InputBox("What is the name for project " & strNextJobNumber & "?",_
"AU project folder builder")
```

```
' if no job name then explain to user and exit
If strJobName = "" Then
   msgBox "No job name entered"
   Wscript.Quit
End If

' set the new job folder name
strJobFolder = objActiveJobsFolder.Path & "\" & strNextJobNumber & " " & strJobName

' Build the folders
objFileSystem.createFolder(strJobFolder)
objFileSystem.createFolder(strJobFolder & "\00 ad")
objFileSystem.createFolder(strJobFolder & "\01 ma")
objFileSystem.createFolder(strJobFolder & "\02 pd")
objFileSystem.createFolder(strJobFolder & "\03 sd")
objFileSystem.createFolder(strJobFolder & "\04 dd")
objFileSystem.createFolder(strJobFolder & "\05 cd")
objFileSystem.createFolder(strJobFolder & "\06 bd")
objFileSystem.createFolder(strJobFolder & "\07 ca")
objFileSystem.createFolder(strJobFolder & "\08 po")
objFileSystem.createFolder(strJobFolder & "\09 ss")


'*******************************************************************************
'* File Name:              foldercleanup.vbs
'* Created:                10.12.2003
'* Author:                 Gordon Price
'* Subroutine Name:        CleanFolders
'* Purpose:                clean folder (and subfolders if flagged)
'*                         of all files that match passed Regular Expression
'* Arguments:              objFolder - folder object to clean
'*                         strFileTypes - file search string (RegExp format)
'*                         blnRecurse - recursion flag
'* Function is called by:  projectarchive.wsf
'*                         projectcleanup.wsf
'*                         BAKCleanup.wsf
'* Function calls:         Self (recursive)
'*******************************************************************************
Sub CleanFolders (objFolder, strFileTypes, blnRecurse)
'*******************************************************************************
'* Variables
Dim objFile
Dim objSubFolder
Dim objRegularExpression

' initialize Regular Expression
Set objRegularExpression = New RegExp
objRegularExpression.IgnoreCase = True
objRegularExpression.Pattern = strFileTypes

' loop through files in folder and delete all that match Regular Expression
For Each objFile in objFolder.Files
   If objRegularExpression.Test(objFile.Name) Then objFile.Delete
Next

' if recursion was flagged then call self for sub folders
If blnRecurse Then
   For Each objSubFolder in objFolder.SubFolders
      CleanFolders objSubFolder,strFileTypes,1
   Next
End If

End Sub



<job>
<script language="VBScript" src="foldercleanup.vbs">
```

```
'*******************************************************************************
'* File Name:   BAKcleanup.wsf
'* Created:     10.12.2003
'* Author:      Gordon Price
'*******************************************************************************
Option Explicit

Dim objArguments
Dim objFileSystem
Dim objFolder

Dim strArgument

Set objArguments = WScript.Arguments
Set objFileSystem = CreateObject("Scripting.FileSystemObject")

On Error Resume Next

For Each strArgument In objArguments
   Set objFolder = objFileSystem.GetFolder(strArgument)
   If Err.Number = 0 Then CleanFolders objFolder,"(.bak)$",0
   Err.Clear
Next

</script>
</job>
```

```
<job>
<script language="VBScript" src="FolderCleanup.vbs">
'*******************************************************************************
'* File Name:   projectcleanup.wsf
'* Created:     10.12.2003
'* Author:      Gordon Price
'*******************************************************************************
Option Explicit

Dim objArguments
Dim objFileSystem
Dim objFolder

Dim strArgument

Set objArguments = WScript.Arguments
Set objFileSystem = CreateObject("Scripting.FileSystemObject")

On Error Resume Next

For Each strArgument In objArguments
   Set objFolder = objFileSystem.GetFolder(strArgument)
   If Err.Number = 0 Then CleanFolders objFolder,"^(\w{3}_)|junk|(.bak|.tmp)$",1
   Err.Clear
Next

</script>
</job>
```

```
<Job id="Project Archive">
<script language="VBScript" src="FolderCleanup.vbs"/>
'*****************************************************************************
'* File Name:   projectarchive.vbs
'* Created:     08.24.1999
'* Author:      Gordon Price
'* Purpose:     Cleanup specified files & file types then move
'*              entire folder to appropriate archive year
'* History:     10.30.2003 - Updated for Regular Expressions
'*****************************************************************************
Option Explicit

Const ROOT_ACTIVE_FOLDER = "C:\_Projects\Active Jobs"
Const ROOT_ARCHIVE_FOLDER = "C:\_Projects\Completed Jobs"
Dim objArguments
Dim objFileSystem
Dim objSourceFolder
Dim strArgument
Dim strJobYear
Dim strJobNumber
Dim strJobFolder
Dim strSourceFolder
Dim strDestinationFolder

Set objArguments = WScript.Arguments
Set objFileSystem = CreateObject("Scripting.FileSystemObject")

For Each strArgument In objArguments

If objFileSystem.FolderExists(strArgument) Then Set objSourceFolder =
objFileSystem.GetFolder(strArgument)

   If objSourceFolder.ParentFolder.Path = ROOT_ACTIVE_FOLDER Then
      strJobYear = Left(objSourceFolder.Name,4)
      If objFileSystem.FolderExists (ROOT_ARCHIVE_FOLDER & "\" & strJobYear) Then
         CleanFolders objSourceFolder,"^(\w{3}_)|junk|(.bak|.tmp)$",1
         strDestinationFolder = ROOT_ARCHIVE_FOLDER & "\" & strJobYear & "\" & objSourceFolder.Name
         objSourceFolder.Move strDestinationFolder
      Else
         msgBox "Archive not yet allowed."
      End If
   Else
      msgBox "This is not a Project Folder!"
   End If

Next

</Script>
</Job>
```