Manipulating Collections, Folders and Files With VBScript's FileSystemObject

Implementation Specifications or Requirements

Category	Item	
Software	IWS Version:	6.0 and later
	Service Pack:	N/A
	Windows Version:	WinXP/2000/NT and Windows CE
	Web Thin Client:	Yes
Equipment	Panel Manufacturer:	N/A
	Panel Model:	N/A
	Other Hardware:	N/A
	Comm. Driver:	All
	Controller (e.g.: PLC)	All
	Application Language:	N/A
Software Demo Application	N/A	

.

Summary

In Application Note AN-00-0005, we examined various means to manipulate Collections, Folders and Files using InduSoft Web Studio's (IWS) built-in functions. As was shown, VBScript code segments can access these IWS built-in functions by adding the "$" character in front of the IWS built-in function. These IWS built-in functions can use VBScript variables, IWS tags and expressions as parameters.

VBScript was initially developed to be used with Web Servers (e.g. using ASP or Active Service Pages), but since VBScript does not have built-in file I/O language elements, a method to access the web server's file system was needed. Microsoft developed the FileSystemObject object model (FSO) which is included in VBScript's runtime library. This object allows for the creation of files, determining whether a file, folder or drive exists, opening a text file, as well as a variety of other tasks.

In this Application Note, we will examine how to manipulate Collections, Drives, Folders and Files from VBScript using FSO. Since FSO is part of VBScript's runtime library, its functions are not accessible from IWS in a native script (e.g. Math Worksheets, Screen Logic, Command Properties). Instead, FSO is only accessible from VBScript code segments (e.g. Global Procedures, Graphic Scripts, Screen Scripts, Command Properties and Background Scripts). However, IWS tags as well as VBScript variables and Expressions can be used as parameters when using FSO object model.

To use FSO, it is helpful to have a basic understanding of Objects, Methods and Properties available. An Object can refer to a self-contained programming entity (such as FSO) that has a collection of functions, called Methods. Objects can also refer to individual entities, such as a Drive, a Folder or a File. These Objects usually have Properties, some which are read-only and others that can be written to. Additionally, a list of parameters may sometimes be required to perform an operation.

Microsoft's MSDN website provides a complete description of FSO.[1]

[1] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/af4423b2-4ee8-41d6-a704-49926cd4d2e8.asp

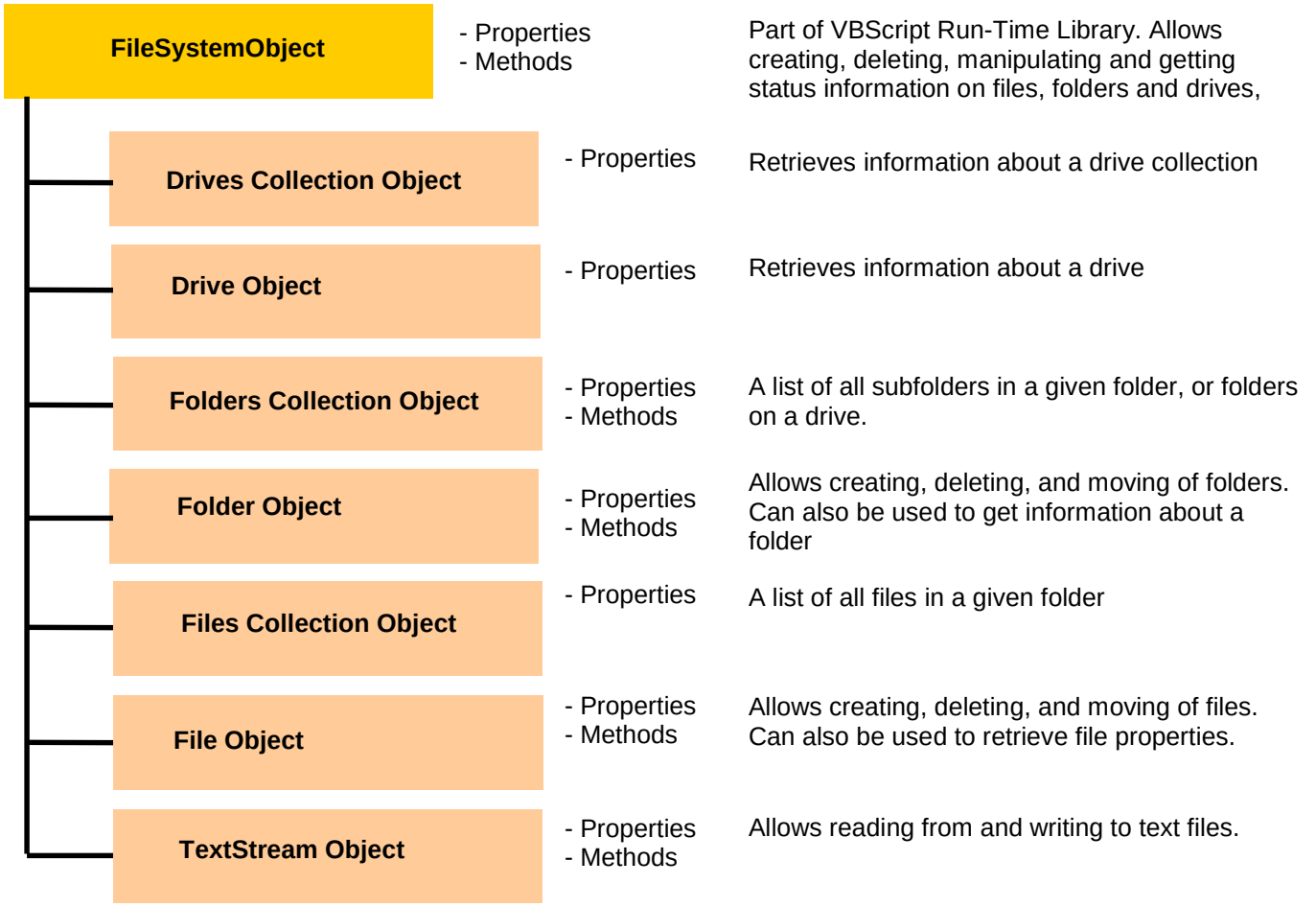
COMPARISON CHART BETWEEN IWS BUILT-IN FUNCTIONS & THE VBSCRIPT FSO

Category	Function	IWS Built-in Function	VBScript FSO
Drive Collection	Returns collection of local and network drives		✓
Drive	Free Space on a disk	✓	✓
Drive	Returns drive type		✓
Drive	Returns drive status (e.g. Ready, Name, Total Size, Volume Name)		✓
Drive	Returns drive root folder		✓
Drive	Drive Exists		✓
Folder Collection	Return a collection of folders in a specified path		✓
Folder	Create folder	✓	✓
Folder	Copy folder		✓
Folder	Delete folder	✓	✓
Folder	Determine folder size	✓	✓
Folder	Rename folder	✓	✓
Folder	Verify folder exists	✓	✓
Folder	Return name of parent folder		✓
Folder	Return a random folder name (for temp storage)		✓
File Collection	Delete files older than a specified date	✓	*
File Collection	Find collection of files that match a path and file mask criteria	✓	*
File Collection	Open dialog box of files in a specified directory matching a criteria	✓	
File	Copy a file	✓	✓
File	Delete a file	✓	✓
File	Determine file size	✓	✓
File	Rename file	✓	✓
File	Get file attributes	✓	✓
File	Get date/time file was last modified	✓	✓
File	Get date/time file was last accessed		✓
File	Determine if file exists	✓	✓
Text File	Create Text File		✓
Text File	Write ASCII string to file	✓	✓
Text File	Write Unicode string to file		✓
Text File	Read text file		✓
Text File	Search a text file for a specific string	✓	*
Text File	Print a text file	✓	
Text File	Set up local printer	✓	
Miscellaneous	Return directory of current application	✓	
Miscellaneous	Return directory of IWS Program files	✓	
Miscellaneous	Read a specified parameter from an INI file	✓	
Miscellaneous	Return directory where Alarm files are stored	✓	
Miscellaneous	Return directory where History files are stored	✓	
Miscellaneous	Export historical Trend files to a .TXT file	✓	
Miscellaneous	Verify conversion of Trend files is complete	✓	
Miscellaneous	Set path for Alarm files	✓	
Miscellaneous	Set path for historical Tend files	✓	
Miscellaneous	Set new path for the Application	✓	
Miscellaneous	Set file used for runtime translation	✓	
Miscellaneous	Enable/disable saving to historical Alarm and historical Trend file	✓	

*can be implemented using combination of statements

The FileSystemObject Object Model

As shown in Figure 1 below, the FileSystemObject object model consists of the FileSystemObject (FSO) object and seven other objects that are “components” of the FileSystemObject model. Each object can have its own set of Properties and Methods.



The FileSystemObject Object Model
Figure 1

Collections are groups of similar objects. For example, a Drives Collection is a group of drives on the local computer or network share drives accessible by the local computer. Folders Collection generally refers to a set of subfolders in a parent folder, while Files Collection refers to a set of files. Collections and other objects (Drive, Folder, File, and TextStream) are usually created from the FSO object.

The FSO, like most other VBScript objects, must first be instantiated. This simply means that a unique instance of the object must be defined in a VBScript code segment and the instance of the object must be assigned to a VBScript variable through the SET statement. After this is done, all Methods and Properties

for that object are referenced through the VBScript variable. Additionally, other objects within the FSO can be instantiated. The following command is used to instantiate the FSO:

```
Dim fso, myFile                                  'Declare the variables
Set fso = CreateObject("Scripting.FileSystemObject")   'Instantiate the FileSystemObject
```

Once the FSO is instantiated, there is one Property and a variety of Methods that can be used on the FSO. These Properties and Methods are itemized in Table A and B, respectively. These Properties and Methods can be used to perform specific operations or generate Collections. Following Table A & B is a detailed description of the various Properties and Methods.

Table A: FileSystemObject Properties

Property	Description
Drives	Returns a Drives Collection object consisting of all Drive objects available to the local machine.

Table B: FileSystemObject Methods

Method	Description
BuildPath	Adds a file or folder specified to the existing path
CopyFile	Copies the file or files to a folder
CopyFolder	Copies the folder or folders to a another folder
CreateFolder	Creates a new folder
CreateTextFile	Creates a new text file on a disk
DeleteFile	Deletes a file or files
DeleteFolder	Deletes a folder or folders
DriveExists	Verifies is a drive exists
FileExists	Verifies if a file exists
FolderExists	Verifies if a folder exists
GetAbsolutePathName	Used to build an unambiguous path to a folder
GetBaseName	Returns the name of a file or folder specified (removes path and extension)
GetDrive	Returns a Drive object
GetDriveName	Returns the name of the drive
GetExtensionName	Returns the extension of a file or folder
GetFile	Returns a File object
GetFileName	Returns the name part of a file (removes path and extension)
GetFileVersion	Returns the version information from a file XXXXXXXXXXXXXXXXXXXXXXX
GetFolder	Returns a Folder object
GetParentFolderName	Returns the name of the parent folder of a folder or file
GetSpecialFolder	Returns a Folder object corresponding to a special Windows folder
GetTempName	Returns a randomly generated file name to be used for a temporary file or folder name
MoveFile	Moves a file or files
MoveFolder	Moves a folder or folders
OpenTextFile	Creates a file (if non-existent) or opens a file (if it exists)

FileSystemObject (FSO)

Function Used to manipulate the Windows File System.

Remarks The FSO is part of VBScript's runtime library and is a COM component. It can be used to generate other objects or collections. The FSO is instantiated through the following statement:

```
Dim objFso                                       'Declare the variable(s)
Set objFso = CreateObject("Scripting.FileSystemObject")   'Instantiate the FileSystemObject
```

Property **Drives**
Description: Returns a collection of Drives **o**bjects.
Use: Set objDrive = *fso*.Drives
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated.
Return: An object containing a collection of Drives objects
Remarks: Returns a collection of Drives **o**bjects available on the local machine, including networked drives mapped to the local machine. Removable media drives do not have to have media inserted to appear in the Drives Collection.
Example:

```
Dim fso, dc, d, strDrvList
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set dc = fso.Drives                                       'Instantiate the Drives collection object
strDrvList = ""
For each d in dc                                          'Evaluate each drive in the drives collection
    strDrvList = strDrvList & d.driveLetter & " – "       'Get the Drive letter
    If d.DriveType = 3 Then                               'See if a network drive
        strDrvList = strDrvList & d.ShareName             'Yes
    ElseIf d.IsReady Then                                 'No – is a local drive. Check if ready
        strDrvList = strDrvList & d.VolumeName            'Yes – add to list
    End If
    strDrvList = strDrvList & vbCrLf                      'Add a Cr & Lf and then get next drive
Next
MsgBox strDrvList                                         'Display the list of drives
```

Note:
- This function is useful for informational display purposes and displays similar information to what would be shown using "My Computer" with the Windows OS.
- IWS does not have a comparable built-in function

Method: **BuildPath**
Description: Appends a name to an existing path
Use: *fso*.**BuildPath**(*path, name*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated.
path
Required. Existing path to which *name* is appended. Path can be absolute or relative, and need not specify an existing folder
name
Required. Name being appended to the existing path.
Return: None
Remarks: The **BuildPath** method inserts an additional path separator between the existing path and the name, only if necessary. Does not check for a valid path.
Example:

```
Dim fso, path, newpath
Set fso = CreateObject("Scripting.FileSystemObject")
path = $getAppPath()
newpath = fso.BuildPath(path, "SubFolder")
```

Note:
- This same function can be easily accomplished in VBScript by string concatenation:

```
path = $getAppPath()                  'Built-in IWS function that returns the current application path
path = path & "subfolder"             'String concatenation
```

Method: **CopyFile**
Description: Copies one or more files from one location to a new location
Use: *fso*.**CopyFile** (*source, destination*[, *overwrite*])
Arguments: *fso*

Required. The name of a FileSystemObject object previously instantiated.

source

Required. A character string file specification, which can include wildcard characters, for one or more files to be copied.

destination

Required. Character string destination where the file or files from *source* are to be copied. Wildcard characters are not allowed in the destination string.

overwrite

Optional. Boolean value that indicates if existing files are to be overwritten. If **True**, files are overwritten; if **False**, they are not. The default is **True**. Note that **CopyFile** will fail if *destination* has the read-only attribute set, regardless of the value of overwrite.

Return: None
Remarks: Wildcard characters can only be used in the last path component of the source argument. If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching files. Otherwise, *destination* is assumed to be the name of a file to create. In either case, three things can happen when a file is copied.

- If *destination* does not exist, *source* gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs if overwrite is **False**. Otherwise, an attempt is made to copy *source* over the existing file.
- If *destination* is a directory, an error occurs. (Occurs because the directory doesn't exist).

An error also occurs if a *source* using wildcard characters doesn't match any files. The **CopyFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs.

Example:

```
Const OverWrite = False
Dim fso, srcFiles, destPath
Set fso = CreateObject("Scripting.FileSystemObject")
srcFiles = $getAppPath() & "Alarm\*.*"
destPath = $getAppPath() & "AlarmHistory"
If fso.FolderExists (destPath) = False Then
    fso.CreateFolder (destPath)
End If
fso.CopyFile  srcFiles, destPath
```

Note:

- If copying a set of files (by using the wildcard) to a destination folder, make sure the destination folder exists otherwise an error will occur.
- The comparable IWS built-in function is **FileCopy**

Method: **CopyFolder**
Description: Copies a folder to a new location
Use: *fso.***CopyFolder** (*source, destination*[, *overwrite*])
Arguments: *fso*

Required. The name of a FileSystemObject object previously instantiated.

source

Required. A character string folder specification, which can include wildcard characters, for one or more folders to be copied. Wildcard characters can only be used in the last path component of the *source* argument.

destination

Required. Character string destination where the folder and subfolders from *source* are to be copied. Wildcard characters are not allowed in the destination string.

overwrite

Optional. Boolean value that indicates if existing folders are to be overwritten. If **True**, files are overwritten; if **False**, they are not. The default is **True**.

Return: None
Remarks: If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching folders and subfolders. Otherwise, *destination* is assumed to be the name of a folder to create. In either case, four things can happen when an individual folder is copied.

- If *destination* does not exist, the source folder and all its contents gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an attempt is made to copy the folder and all its contents. If a file contained in *source* already exists in *destination*, an error occurs if overwrite is false. Otherwise, it will attempt to copy the file over the existing file.
- If *destination* is a read-only directory, an error occurs if an attempt is made to copy an existing read-only file into that directory and overwrite is false.

An error also occurs if a *source* using wildcard characters doesn't match any folders. The **CopyFolder** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs

Example:

```
Const OverWrite = False
Dim fso, srcPath, destPath
Set fso = CreateObject("Scripting.FileSystemObject")
srcPath = $getAppPath() & “*”
destPath = fso.GetParentFolderName(srcPath) & “SaveApp”
If fso.FolderExists (destPath) = False Then
    fso.CreateFolder (destPath)
End If
fso.CopyFolder  srcPath, destPath, OverWrite
```

Notes:

- If copying a set of folders (by using the wildcard) to a destination folder, you can designate subfolders using the path separator “\” and a wildcard “*”; e.g “c:\myAppFolder*” or “c:\myAppFolder**”
- **CopyFolder** will generate an “Invalid Path” error is you specify subfolders that do not exist, so be careful not to specify subfolders at a level where they do not exist.
- IWS does not have a comparable built-in Function

Method: **CreateFolder**
Description: Creates a new folder in the specified location
Use: *fso*.CreateFolder(*foldername*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated.
foldername
Required. A character string expression that identifies the folder to create.
Return: None
Remarks: An error occurs if the specified folder already exists.
Example:

```
Dim fso, destPath
Set fso = CreateObject("Scripting.FileSystemObject")
destPath = $getAppPath() & “AlarmHistory”
If fso.FolderExists (destPath) = False Then
    fso.CreateFolder (destPath)
End If
```

Note:
- The comparable IWS built-in function is **DirCreate**

Method: **CreateTextFile**
Description: Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file
Use: Set objfile = *fso.***CreateTextFile**(*filename*[, *overwrite*[, *Unicode*]])
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
filename
Required. A string expression that identifies the file to create
overwrite
Optional. Boolean value that indicates whether you can overwrite an existing file. The value is **True** if the file can be overwritten, **False** if it can't be overwritten. If omitted, existing files are not overwritten (default **False**).
unicode
Optional. Boolean value that indicates whether the file is created as a Unicode or ASCII file. If the value is **True,** the file is created as a Unicode file. If the value is **False,** the file is created as an ASCII file. If omitted, an ASCII file is assumed.
Remarks: None
Example:

```
Dim fso, myFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set myFile = fso.CreateTextFile("c:\testfile.txt", True, False)
myFile.WriteLine("This is a test.")
myFile.Close
Set Myfile = Nothing
Set fso = Nothing
```

Notes:
- The **CreateTextFile** method allows you to create a text file for UniCode characters. Compare this to the IWS built-in FileWrite function which only supports ASCII files.
- One weakness with FSO is that there is no command to search a text file for a specified string (like the IWS built-in function GetLine). However, this function can be accomplished with VBScript code.
- Although the **CreateTextFile** method indicates that it will support file reads, it does not appear to work. For reading to TextStream files, use the **OpenTextFile** or **OpenAsTextStream** methods.

Method: **DeleteFile**
Description: Deletes a specified file
Use: *fso*.DeleteFile (*filename[, force]*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
filename
Required. The name of the file to delete. The filename can contain wildcard characters in the last path component.
force
Optional. Boolean value that is **True** of files with the read-only attribute set are to be deleted; **False** if they are not. **False** is the default.
Return: None
Remarks: An error occurs if no matching files are found. The **DeleteFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.
Example:

```
Dim fso, myFile
Set fso = CreateObject("Scripting.FileSystemObject")
myFile = “C:\TempData\Log*.dat”
fso.DeleteFile(myFile)
Set fso = Nothing
```

Notes:
- The **DeleteFile** method allows you to specify wildcard characters in the last path component. The comparable IWS built-in function **FileDelete** does not let you do this.

Method: **DeleteFolder**
Description: Deletes the specified folder and its contents
Use: *fso*.DeleteFolder (*folderspec[, force]*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
folderspec
Required. The name of the folder to delete. The folderspec can contain wildcard characters in the last path component.
force
Optional. Boolean value that is **True** of folders with the read-only attribute set are to be deleted; **False** if they are not. **False** is the default.
Return: None
Remarks: The **DeleteFolder** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents. An error occurs if no matching folders are found. The **DeleteFolder** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.
Example:

```
Dim fso, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")
myFolder = “C:\TempData\”
fso.DeleteFolder(myFolder)
Set fso = Nothing
```

Note:
- The **DeleteFolder** method allows you to specify wildcard characters in the last path component. The comparable IWS built-in function DirDelete does not let you do this.

Method: **DriveExists**
Description: Determines whether or not a specified drive exists
Use: *fso*.DriveExists (*drivespec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
drivespec
Required. A drive letter or a complete path specification.
Return: Returns a boolean **True** if the specified drives exists, otherwise returns **False.**
Remarks: For drives with removable media, the **DriveExists** method returns **true** even if there are no media present. Use the **IsReady** property of the **Drive** object to determine if a drive is ready.
Example:

```
Dim fso, drv, msg
Set fso = CreateObject("Scripting.FileSystemObject")
drv = “e:\”
If fso.DriveExists(drv) Then
    msg = "Drive " & UCase(drv) & " exists."
Else
    msg = "Drive " & UCase(drv) & " doesn't exist."
End If
MsgBox msg
```

Note:
- IWS does not have a comparable built-in Function

Method: **FileExists**
Description: Determines whether or not a specified file exists
Use: *fso*.FileExists (*filespec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
filespec
Required. The name of the file whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the file isn't expected to exist in the current folder
Return: Returns a boolean **True** if the specified file exists, otherwise returns **False.**
Remarks: None
Example:

```
Dim fso, myFile, msg
Set fso = CreateObject("Scripting.FileSystemObject")
myFile = $getAppPath() & “data\Mydata.mdb”
If fso.FileExists(myFile) Then
    msg = myFile & " exists."
Else
    msg = myFile & "doesn't exist."
End If
MsgBox msg
```

Note:
- The comparable IWS built-in function is **FindFile. FindFile** is more powerful in that it allows a file mask (i.e. wildcard as the last path component) whereas FSO **FileExist** does not.

Method: **FolderExists**
Description: Determines whether or not a specified folder exists
Use: *fso*.FolderExists (*folderspec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
folderspec
Required. The name of the folder whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the folder isn't expected to exist in the current folder
Return: Returns a boolean **True** if the specified folder exists, otherwise returns **False.**
Remarks: None
Example:

```
Dim fso, myFolder, msg
Set fso = CreateObject("Scripting.FileSystemObject")
myFolder = $getAppPath() & “data\”
If fso.FolderExists(myFolder) Then
    msg = myFolder & " exists."
Else
    msg = myFolder & "doesn't exist."
End If
MsgBox msg
```

Note:
- The comparable IWS built-in function is **FindPath.**

Method: **GetAbsolutePathName**
Description: Returns a complete and unambiguous path name that cannot be easily determined from the specified path information.
Use: strPath = *fso*.GetAbsolutePathName(*pathspec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
pathspec
Required. Path specification to change to a complete and unambiguous path
Return: String containing a complete and unambiguous path name
Remarks: A path is complete and unambiguous if it provides a complete reference from the root of the specified drive. A complete path can only end with a path separator character (\) if it specifies the root folder of a mapped drive. Assuming the current directory is c:\mydocuments\reports, the following table illustrates the behavior of the **GetAbsolutePathName** method:

pathspec	Returned path
"c:"	"c:\mydocuments\reports"
"c:.."	"c:\mydocuments"
"c:\"	"c:\"
"c:*.*\may97"	"c:\mydocuments\reports*.*\may97"
"region1"	"c:\mydocuments\reports\region1"
"c:\..\..\mydocuments"	"c:\mydocuments"

Example:

```
Dim fso, pathSpec, myPath
Set fso = CreateObject("Scripting.FileSystemObject"     ‘Current directory is c:\mydocuments\reports
pathSpec = “C:\”
myPath = fso.GetAbsolutePathName(pathSpec)              ‘Returns c:\mydocuments\reports
```

Note:
- The comparable IWS built-in function is **GetAppPath()**. Note that the **GetAbsolutePathName** function does not put a path delimiter “\” on the last path component, whereas the IWS built-in function always does.

Method: **GetBaseName**
Description: Returns just the name of the object specified. It removes all other information including the extension
Use: strBaseName = *fso*.GetBaseName(*path*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
path
Required. The path specification for the component whose base name is to be returned.
Return: String containing the name of the object specified.
Remarks: The **GetBaseName** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path. The **GetBaseName** method returns a zero-length string (“”) if no component matches the *path* argument.
Example:

```
Dim fso, filespec, baseName
Set fso = CreateObject("Scripting.FileSystemObject"
filespec = $getAppPath() & “recipes.xml”
baseName = fso.GetBaseName (filespec)                ‘Returns “recipes”
```

Note:
- There is no comparable IWS built-in function, but the **GetBaseName** method is of little use in an IWS application.

Method: **GetDrive**
Description: Returns a **Drive** object corresponding to the drive for a specified path
Use: objDrv = *fso*.GetDrive(*drivespec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
drivespec
Required. The *drivespec* argument can be a drive letter (c), a drive letter with a colon appended (c:), a drive letter with a colon and path separator appended (c:\), or any network share specification (\\computer2\share1).
Return: Drive Object corresponding to the drive for a specified path
Remarks: For network shares, a check is made to ensure that the share exists. An error occurs if drivespec does not conform to one of the accepted forms or does not exist.
Example:

```
Dim fso, drvPath, d, s
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = “c:”
Set d = fso.GetDrive(fso.GetDriveName(drvPath))
s = "Drive " & UCase(drvPath) & " - "
s = s & d.VolumeName   & vbCrLf
s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
s = s & " Kbytes"
MsgBox s
```

Note:
- There is no comparable IWS built-in function. **GetDrive** returns a Drive object for subsequent processing.

Method: **GetDriveName**
Description: Returns a string containing the name of the drive for a specified path
Use: strName = *fso*.GetDriveName(*path*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
path
Required. The path specification for the component whose drive name is to be returned.
Return: String containing the name of the drive for a specified path
Remarks: The **GetDriveName** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path. The **GetDriveName** method returns a zero-length string (“”) if the drive can’t be determined.
Example:

```
Dim fso, drvPath, GetAName
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = “c:”
GetAName = fso.GetDriveName(drvPath)          ‘Returns “c:”
```

Note:
- There is no comparable IWS built-in function but **GetDriveName** is of little use in an IWS application.

Method: **GetExtensionName**
Description: Returns a string containing the extension name for the last component in a path.
Use: strExtName = *fso*.GetExtensionName(*path*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
path
Required. The path specification for the component whose drive name is to be returned.
Return: String containing the extension name for the last component in a path.
Remarks: For network drives, the root directory (\) is considered to be a component. The **GetExtensionName** method returns a zero-length string ("") if no component matches the path argument.
Example:

```
Dim fso, drvPath, ExtName
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = $getAppPath() & “recipes.xml”
ExtName = fso.GetExtensionName(drvPath)          ‘Returns “xml”
```

Note:
- There is no comparable IWS built-in function but **GetDriveName** is of little use in an IWS application.

Method: **GetFile**
Description: Returns a **File** object corresponding to the file in the specified path. The file object methods and properties can be accessed. See ***File Object*** for the file object's methods and properties.
Use: objFile = *fso*.GetFile(*fileSpec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
fileSpec
Required. The filespec is the path (absolute or relative) to a specific file.
Return: File Object
Remarks: An error occurs if the specified file does not exist. The **GetFile** method does not support the use of wildcard characters, such as ? or *.
Example:

```
Dim fso, fileSpec, f, s
Set fso = CreateObject("Scripting.FileSystemObject")
fileSpec = $getAppPath() & "recipes.xml"
Set f = fso.GetFile(fileSpec)
s = f.Path & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
MsgBox s
```

Note:
- There is no comparable IWS built-in function. **GetFile** returns a File object for subsequent processing.

Method: **GetFileName**
Description: Returns the last component of a specified path (file name or folder name) that is not part of the drive specification.
Use: strName = *fso*.GetFileName(*fileSpec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
fileSpec
Required. The path (absolute or relative) to a specific file.
Return: String containing the last component of a specified path
Remarks: The **GetFileName** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path. The **GetFileName** method returns a zero-length string ("") if *pathspec* does not end with the named component.
Example:

```
Dim fso, fileSpec, s
Set fso = CreateObject("Scripting.FileSystemObject")
fileSpec = $getAppPath() & "recipes.xml"
s = fso.GetFile(fileSpec)                    'Returns "recipes.xml"
MsgBox s
```

Note:
- There is no comparable IWS built-in function but **GetFileName** is of little use in an IWS application.

Method: **GetFileVersion**
Description: Returns the version number of a specified file
Use: strVersionNum = *fso*.GetFileVersion(*fileSpec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
fileSpec
Required. The path (absolute or relative) to a specific file.
Return: String containing the version number of a specified file
Remarks: The **GetFileVersion** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path. The **GetFileVersion** method returns a zero-length string (“”) if *pathspec* does not end with the named component.
Example:

```
Dim fso, fileSpec, s
Set fso = CreateObject("Scripting.FileSystemObject")
fileSpec = “c:\windows\system32\notepad.exe”
s = fso.GetFile(fileSpec)                    ‘Returns “5.1.2600.2180”
If Len(s) Then
    MsgBox “File Version is : “ & s
Else
    MsgBox “No File Version information is available”
End If
```

Note:
- There is no comparable IWS built-in function but **GetFileVersion** is of little use in an IWS application.

Method: **GetFolder**
Description: Returns a **Folder** object corresponding to the folder in a specified path
Use: objFolder = *fso*.**GetFolder**(*folderSpec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
folderSpec
Required. The folderspec is the path (absolute or relative) to a specific folder.
Return: Returns a folder object
Remarks: Since this method creates an object, you need to use it with the Set command. An error occurs if the specified folder does not exist.
Example:

```
Dim fso, drvPath, f, fc, s
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = $getAppPath()
Set f = fso.GetFolder(drvPath)
Set fc = f.SubFolders
s = “”
For Each x in fc
    s = s & x.Name & vbCrLf
Next
MsgBox s                                     ‘Displays a list of folders in the App directory
```

Note:
- There is no comparable IWS built-in function. **GetFolder** returns a File object for subsequent processing.

Method: **GetParentFolderName**
Description: Returns a string containing the name of the parent folder of the last component in the specified path
Use: strName = *fso*.GetParentFolderName(*path*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
path
Required. The path specification for the component whose parent folder name is to be returned.
Return: String containing the name of the parent folder
Remarks: The **GetParentFolderName** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path. The **GetParentFolderName** method returns a zero-length string (“”) if there is no parent folder for the component specified in the *path* argument.
Example:

```
Dim fso, drvPath, s
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = $getAppPath()
s = fso.GetParentFolderName(drvPath)
MsgBox “Parent Folder = “ & s        ‘Returns “c:\My Documents\InduSoft Web Studio v6.1 Projects”
```

Note:
- There is no comparable IWS built-in function.

Method: **GetSpecialFolder**
Description: Returns the special folder specified
Use: strFolderName = *fso*.GetSpecialFolder(*folderSpec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
folderSpec
Required. Then name of the special folder to be returned. Can be any of the following constants:

Constant	Value	Description
WindowsFolder	0	The Windows folder containing files installed by the Windows operating system
SystemFolder	1	The (Windows) System folder containing libraries, fonts and device drivers
TemporaryFolder	2	The Temp folder is used to store temporary files. Its path is found in the TMP environment variable.

Return: String containing the name of the parent folder
Remarks: None
Example:

```
Dim fso, WinFolder, SysFolder
Set fso = CreateObject("Scripting.FileSystemObject")
WinFolder = fso.GetSpecialFolder(0) & “\”        ‘Result is “C:\Windows\”
SysFolder = fso.GetSpecialFolder(1) & “\”        ‘Result is “C:\Windows\system32\”
```

Note:
- There is no comparable IWS built-in function.

Method: **GetStandardStream**
Description: Returns a **TextStream** object corresponding to the standard input, output, or error stream

Note:
- The **GetStandardStream** Method does not work with IWS and if you use it, you will get an error. **GetStandardStream** only works for standard I/O when CScript is the VBScript Interpreter. For operator I/O, use MsgBox and InputBox instead.

Method: **GetTempName**
Description: Returns a randomly generated temporary file or folder name that is useful for performing operations that require a temporary file or folder
Use: strName = *fso.*GetTempName
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
Return: String that contains a randomly generated temporary file or folder name. A random name with a .tmp extension will be returned.
Remarks: The **GetTempName** method does not create a file. It only provides only a temporary file name that can be used with **CreateTextFile** to create a file.
Example:

```
Dim fso, tempFile

Function CreateTempFile
    Const TemporaryFolder = 2
    Dim tfolder, tname, tfile
    Set tfolder = fso.GetSpecialFolder(TemporaryFolder)
    tname = fso.GetTempName
    Set tfile = tfolder.CreateTextFile(tname)
    Set CreateTempFile = tfile
End Function

Set fso = CreateObject("Scripting.FileSystemObject")
Set tempFile = CreateTempFile
tempFile.WriteLine "Hello World"
tempFile.Close
```

Note:
- IWS has the built-in function **DirCreate** to create a folder but there is no IWS built-in function to create a text file.
- The **GetTempName** function can be used to create a temporary file for data logging or any other purpose. The file can subsequently be renamed and moved or copied to another location.

Method: **MoveFile**
Description: Moves one or more files from one location to another
Use: *fso*.MoveFile (*source, destination*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
source
Required. The path to the file or files to be moved. The source argument string can contain wildcard characters in the last path component only.
destination
Required. The path where the file or files are to be moved. The destination argument can't contain wildcard characters.
Return: None
Remarks: If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination file to create. In either case, three things can happen when an individual file is moved:

- If *destination* does not exist, the file gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in source doesn't match any files. The **MoveFile** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs

Example:
```
Dim fso, drvPath
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = $getAppPath() & “recipes.xml”
fso.MoveFile drvPath, "c:\backup\"
```

Note:

- The comparable IWS built-in function is **FileRename.**
- This Method allows moving files between volumes only if supported by the operating system.

Method: **MoveFolder**
Description: Moves one or more folders from one location to another.
Use: *fso*.MoveFolder (*source, destination*)
Arguments: *fso*

Required. The name of a FileSystemObject object previously instantiated

source

Required. The path to the folder or folders to be moved. The source argument string can contain wildcard characters in the last path component only.

destination

Required. The path where the folder or folders are to be moved. The destination argument can't contain wildcard characters.

Return: None
Remarks: If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching folders. Otherwise, *destination* is assumed to be the name of a destination folder to create. In either case, three things can happen when an individual folder is moved:

- If *destination* does not exist, the folder gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in source doesn't match any folders. The **MoveFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs

Example:
```
Dim fso, drvPath
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = $getAppPath()
fso.MoveFolder drvPath, "c:\backup\"
```

Notes:

- The comparable IWS built-in function is **DirRename.**
- The FSO **MoveFolder** method allows moving folders between volumes only if supported by the operating system. .
- You can use the Folder Object **Move** method to move an individual folder. The FSO **Move** method supports moving multiple folders.

Method: **OpenTextFile**

Description: Opens a specified file and returns a **TextStream** object that can be used to read from, write to, or append to a file.

Use: oTSO = *fso*.OpenTextFile(*filename* [*, iomode[, create[, format]]*])

Arguments:

fso

Required. The name of a FileSystemObject object previously instantiated

filename

Required. A string expression that identifies the file to open.

iomode

Optional. Indicates the file input/output mode. Can be one of three constants:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file
ForWriting	2	Open a file for reading & writing
ForAppending	8	Open a file and write to the end of the file

create

Optional. Boolean value that indicates whether a new file can be created if the specified *filename* doesn't exist. The value is **True** if a new file is to be created if it doesn't exist, **False** if it isn't to be created if it doesn't exist. If omitted, a new file isn't created (default = **FALSE**).

format

Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default
TristateTrue	-1	Opens the file as Unicode
TristateFalse	0	Opens the file as ASCII

Return: A **TextStream** object

Remarks: None

Example:

```
Const ForReading=1, ForWriting=2, ForAppending=8
Dim fso, f
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
f.Write "Hello world!"
f.Close
```

Notes:

- The IWS built-in function **FileWrite** can be used to create a file and write an ASCII string into it. However, **FileWrite** does not support UniCode characters.
- The VBScript **OpenAsTextStream** Method is functionally equivalent to the **OpenTextFile** Method. The difference is that the **OpenTextFile** Method is called using a FileSystemObject object, while the **OpenAsTextStream** method is called using a File object.

The Drives Collection Object

The FileSystemObject (FSO) Object model can return three types of object collections, or groupings of like objects. These collections are the **Drives** collection (a collection of local and network shared drives), the **Folders** collection (a collection of subfolders under a parent folder) and the **Files** collection (a collection of files under a folder). Since each of these collections is itself an object, the collection must be instantiated with the **Set** command. However, the method of instantiation is different for each type of collection.

The first of these collections is the Drives collection, which is retrieved from the Drives property of the FSO object. Once the Drives collection object is instantiated, you can iterate through the collection to retrieve each of the objects (individual drives) contained in the collection. The syntax for the Drives collection use is as follows:

FSO Property **Drives**
Description: Returns a collection of Drives **o**bjects.
Use: Set objDrives = *fso*.Drives
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated.
objDrives
Required. The name of a Drive Collection.
Return: An object containing a collection of Drives objects
Remarks: Returns a collection of Drives **o**bjects available on the local machine, including networked drives mapped to the local machine. Removable media drives do not have to have media inserted to appear in the Drives Collection.
Example:

```
Dim fso, dc, d, strDrvList
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set dc = fso.Drives                                       'Instantiate the Drives collection object
strDrvList = ""
For each d in dc                                          'Evaluate each drive in the drives collection
    strDrvList = strDrvList & d.driveLetter & " – "       'Get the Drive letter
    If d.DriveType = 3 Then                               'See if a network drive
        strDrvList = strDrvList & d.ShareName             'Yes
    ElseIf d.IsReady Then                                 'No – is a local drive. Check if ready
        strDrvList = strDrvList & d.VolumeName            'Yes – add to list
    End If
    strDrvList = strDrvList & vbCrLf                      'Add a Cr & Lf and then get next drive
Next
MsgBox strDrvList                                         'Display the list of drives
```

Table C: Drives Collection Properties

Property	Description
Count	Returns the number of items in the collection
Item	Returns an item from the collection based on the specified key

Property **Count**
Description: Returns the number of items in a collection. Read only.
Use: intCount = *objDrives*.Count
Arguments: *objDrives*
Required. The name of a Drive Collection previously instantiated.
Return: The number of items in a collection.
Remarks: Read only.
Example:
```
Dim fso, dc, totDrives
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
Set dc = fso.Drives                                      'Instantiate the Drives collection object
totDrives = dc.Count
MsgBox "There are " & totDrives & " drives available"
```

Property **Item**
Description: Returns an item (a Drive Name) based on the specified key.
Use: strName = *objDrives*.Item(*key*)
Arguments: *objDrives*
Required. The name of a Drive Collection previously instantiated.
key
Required. The *key* is associated with the *item* being retrieved.
Return: The drive name for a specified key.
Remarks: Read only. This is a function more commonly used with the VBScript dictionary object. (Scripting.Dictionary). The "Item" is similar to a numerical-based index in an array, except that an Item can be character based and it must be unique.
Example:
```
Dim fso, dc, myItem
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
Set dc = fso.Drives                                      'Instantiate the Drives collection object
myItem = dc.Item ("c")
MsgBox myItem                                            'Displays "c:"
```

Notes:

- The **Item** Property of a Drives Collection is of little value in a typical IWS application.
- The Drives Collection by itself is of limited use other than to give a count of the number of drives available to the local computer.
- The Drives Collection provides an object which can be further manipulated to access the individual drives in the collection. E.g.:

```
Dim fso, dc, d, strDrvList
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
Set dc = fso.Drives                                      'Instantiate the Drives collection object
strDrvList = ""
For each d in dc                                         'Evaluate each drive in the drives collection
    strDrvList = strDrvList & d.driveLetter & " – "      'Get the Drive letter
        If d.DriveType = 3 Then                          'See if a network drive
            strDrvList = strDrvList & d.ShareName        'Yes
        ElseIf d.IsReady Then                            'No – is a local drive. Check if ready
            strDrvList = strDrvList & d.VolumeName       'Yes – add to list
        End If
        strDrvList = strDrvList & vbCrLf                 'Add a Cr & Lf and then get next drive
Next
MsgBox strDrvList                                        'Display the list of drives
```

- There are no Methods for the Drive Collection object
- There is no built-in IWS function that returns a Drives Collection
- Drives Collection objects are not necessarily sorted.

The Folders Collection Object

The Folders Collection is the second of the collection objects available to the FSO object model. The Folders collection object is a collection of subfolders contained in a parent folder or path. Once instantiated, you can iterate through the Folders collection to retrieve an individual subfolder or information about each of the subfolders.

The method of instantiating the Folders collection object is different than a Drives collection object. The steps to instantiating the Folders collection is to first instantiate the parent folder by the FSO GetFolder method. Then, the Folders Collection object is instantiated by calling the SubFolders method on the parent folder object. This method returns a Folders Collection object which you can iterate through as shown below:

FSO Method **GetFolder**
Description: Returns a **Folder** object corresponding to the folder in a specified path
Use: objFolder = *fso*.**GetFolder**(*folderspec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
folderspec
Required. The folderspec is the path (absolute or relative) to a specific folder.
Return: Returns a folder object
Remarks: Since this method creates an object, you need to use it with the Set command. An error occurs if the specified folder does not exist.
Example:

```
Dim fso, drvPath, f, fc, nf,
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = $getAppPath()
Set f = fso.GetFolder(drvPath)              'Instantiate the parent folder object
Set fc = f.SubFolders                       'Return the subfolder Folders collection
s = ""
For Each x in fc
    s = s & x.Name & vbCrLf                 'Iterate through the Folders collection object
Next
MsgBox s                                    'Displays a list of subfolders in the App directory
```

Table D: Folders Collection Properties

Property	Description
Count	Returns the number of items in the collection
Item	Returns an item from the collection based on the specified key

Table E: Folders Collection Methods

Method	Description
Add	Adds a new folder to the Folders Collection

Property **Count**
Description: Returns the number of items in a collection. Read only.
Use: intCount = *objFolders*.Count
Arguments: *objFolders*
Required. The name of a Folders Collection previously instantiated.
Return: The number of items in a collection.
Remarks: Read only.
Example:

```
Dim drvPath, fso, fc, f, numf
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
drvPath = $getAppPath()
Set f = fso.GetFolder(drvPath)                          'Instantiate the parent folder object
Set fc = f.SubFolders                                   'Return the subfolder Folders collection
numf = fc.Count
MsgBox "There are " & numf & " subfolders"
```

Property **Item**
Description: Returns an item (a Drive Name) based on the specified key.
Use: strName = *objFolders*.Item(*key*)
Arguments: *objFolders*
Required. The name of a Folders Collection.
key
Required. The *key* is associated with the *item* being retrieved.
Return: The drive name for a specified key.
Remarks: Read only. This is a function more commonly used with the VBScript dictionary object. (Scripting.Dictionary). The "Item" is similar to a numerical-based index in an array, except that an Item can be character based and it must be unique.
Example:

```
Dim drvPath, fso, fc, myItem
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
drvPath = $getAppPath()
Set f = fso.GetFolder(drvPath)                          'Instantiate the parent folder object
Set fc = f.SubFolders                                   'Return the subfolder Folders collection
myItem = fc.Item ("Web")
MsgBox myItem                                           'displays the entire path to the Web subfolder
```

Method	**Add**
Description:	Adds a new folder to the Folders collection.
Use:	*objFolders*.Add(*folderName*)
Arguments:	*objFolders* Required. The name of a Folders Collection previously instantiated. *folderName* Required. The name of the new Folder being added.
Return:	None
Remarks:	Adds a subfolder to the parent folder. An error occurs if the *folderName* already exists.

Example:

```
Dim drvPath, fso, fc, numf
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
drvPath = $getAppPath()
Set f = fso.GetFolder(drvPath)                           'Instantiate the parent folder object
Set fc = f.SubFolders                                    'Return the subfolder Folders collection
numf = fc.Count
MsgBox "There are " & numf & " subfolders"               'Returns "7"
fc.Add ("TempData")                                      'Add a "TempData" subfolder
numf = fc.Count
MsgBox "There are " & numf & " subfolders"               'Returns "8"
```

Notes:

- As with the Drives Collection, the **Item** Property of a Folders Collection is of little value in a typical IWS application.
- In addition to the Folders Collection **Add** method, the FSO **CreateFolder** method is another way to create a folder.
- There is no built-in IWS function that returns a Folders Collection
- Folders Collection objects are not sorted. This can be done by an external procedure.

The Files Collection Object

The Files Collection is the third (and final) type of collection objects available in the FSO object model. The Files collection object is a collection of files contained in a specified folder. Once instantiated, you can iterate through the Files collection to retrieve an individual file or information about each of the files in the specified folder.

The method of instantiating the Files collection object is similar to the Folders collection object. The steps to instantiating the Files collection is to first instantiate the specified folder by the FSO GetFolder method. Then, the Files Collection object is instantiated by calling the Files method on the folder object. This method returns a Files Collection object which you can iterate through as shown below:

FSO Method **GetFolder**
Description: Returns a **Folder** object corresponding to the folder in a specified path
Use: objFolder = *fso*.**GetFolder**(*folderspec*)
Arguments: *fso*
Required. The name of a FileSystemObject object previously instantiated
folderspec
Required. The folderspec is the path (absolute or relative) to a specific folder.
Return: Returns a folder object
Remarks: Since this method creates an object, you need to use it with the Set command. An error occurs if the specified folder does not exist.
Example:

```
Dim fso, drvPath, f, fc, x, s
Set fso = CreateObject("Scripting.FileSystemObject")
drvPath = $getAppPath() & “Hst”
Set f = fso.GetFolder(drvPath)          ‘Instantiate the folder object
Set fc = f.Files                        ‘Return the Files collection
s = “”
For Each x in fc
    s = s & x.Name & vbCrLf             ‘Iterate through the Files collection object
Next
MsgBox s                                ‘Displays a list of files in the “Hst” subfolder
```

Table F: Files Collection Properties

Property	Description
Count	Returns the number of items in the collection
Item	Returns an item from the collection based on the specified key

Property **Count**
Description: Returns the number of items in a collection. Read only.
Use: intCount = *objFiles*.Count
Arguments: *objFiles*
Required. The name of a Files Collection object previously instantiated.
Return: The number of items in a collection.
Remarks: Read only.
Example:

```
Dim drvPath, fso, fc, numf
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
drvPath = $getAppPath()
Set f = fso.GetFolder(drvPath)                          'Instantiate the parent folder object
Set fc = f.Files                                        'Return the subfolder Folders collection
numf = fc.Count
MsgBox "There are " & numf & " files"
```

Property **Item**
Description: Returns an item (a Drive Name) based on the specified key.
Use: strName = *objFiles*.Item(*key*)
Arguments: *objFiles*
Required. The name of a Folders Collection object previously instantiated.
key
Required. The *key* is associated with the *item* being retrieved.
Return: The drive name for a specified key.
Remarks: Read only. This is a function more commonly used with the VBScript dictionary object. (Scripting.Dictionary). The "Item" is similar to a numerical-based index in an array, except that an Item can be character based and it must be unique.
Example:

```
Dim drvPath, fso, fc, myItem
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
drvPath = $getAppPath()
Set f = fso.GetFolder(drvPath)                          'Instantiate the parent folder object
Set fc = f.Files                                        'Return the subfolder Folders collection
myItem = fc.Item ("myApp.app")
MsgBox myItem                                           'displays the entire path to myApp.app
```

Notes:

- As with the Drives and Folders Collection objects, the **Item** Property of a Files Collection is of little value in a typical IWS application.
- There is no built-in IWS function that returns a Files Collection object. However, there are specific built-in IWS functions that manipulate collections of files, such as **DeleteOlderFiles**, **FindFiles**, and **ReadFileN.** The functions **DeleteOlderFiles** and **FindFiles** can be implemented with additional logic in VBScript. The **ReadFileN** provides a dialog box and selection of an item in the dialog box, and this functionality is not easily replicated in VBScript.
- There are no Methods for Files Collection object.
- Files Collection objects are not sorted. This can be done by an external procedure.

The Drive Object

The Drive Object lets the programmer refer to a specific disk drive or network share drive. Once the Drive object is instantiated, it can be referred to as an object from VBScript and its various Properties accessed.

The Drive Object is instantiated as follows:

```
Dim fso, d, driveSpec
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO Object
driveSpec = "c"
Set d = fso.GetDrive(driveSpec)                         'Instantiate the Drive Object
```

See the **GetDrive** method under the FileSystemObject Object Model section for additional details on instantiation of the Drive Object.

The Drive object has no Methods, only Properties. These properties are generally read-only and follow the format:

return = *objDrive.Property*
where
return = return value or a returned object
objDrive = the required Drive object ("d" in the examples below)
Property = the Drive object property being accessed

Table G: Drive Object Properties

Property	Description
AvailableSpace	Returns the amount of space available to the user on the specified drive or network share drive
DriveLetter	Returns the drive letter of a physical local drive or a network share drive. Read-only value
DriveType	Returns the value indicating the type of the specified drive.
FileSystem	Returns the type of the file system in use for the specified drive. Return types are FAT, NTFS, CDFS
FreeSpace	Returns the amount of free space (in bytes) available to the user on a specified drive or network share drive. Read-only
IsReady	Returns True if the specified file is ready, otherwise returns False. For removable media drives, returns True only when the media is inserted and ready for access
Path	Returns the path for a specified drive. For Drive letters, the root drive is not included. E.g. the path for the C drive is C:, not C:\
RootFolder	Returns a Folder object representing the root folder of a specified drive. Read only value
SerialNumber	Returns a decimal number used to uniquely identify a disk volume
ShareName	Returns the network share name for a specified drive
TotalSize	Returns the total space, in bytes, of a Drive or network shared drive
VolumeName	Sets or returns the volume name from the specified drive. Read/Write.

Property **AvailableSpace**
Description: Returns the amount of space available to a user on the specified drive or network share drive.
Use: intSpace = *objDrive.*AvailableSpace
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: The read-only value returned by the **AvailableSpace** property is typically the same as that returned by the **FreeSpace** property. Differences may occur between the two for computer systems that support quotas.
Remarks: Read only.
Example:
```
Dim fso, d
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set d = fso.GetDrive(fso.GetDriveName("c:")
MsgBox "Available Space = " & FormatNumber(d.AvailableSpace/1024, 0) & " Kbytes"
```

Property **DriveLetter**
Description: Returns the drive letter of a physical local drive or a network share.
Use: strLetter = *objDrive.*DriveLetter
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: The **DriveLetter** property returns a zero-length string ("") if the specified drive is not associated with a drive letter, for example, a network share that has not been mapped to a drive letter.
Remarks: Read only.
Example:
```
Dim fso, d
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set d = fso.GetDrive(fso.GetDriveName("c:")
MsgBox "Drive " & d.DriveLetter & ":"
```

Property **DriveType**
Description: Returns a value indicating the type of a specified drive.
Use: intType = *objDrive.*DriveType
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: The **DriveType** property a value indication the type of the specified drive. Return values are:
0 – unknown
1 – Removable
2 – Fixed
3 – Network
4 – CD-ROM
5 – RAM Disk
Remarks: Read only.
Example:
```
Dim fso, d, t
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set d = fso.GetDrive(fso.GetDriveName("c:")
Select Case d.DriveType
    Case 0: t = "Unknown"
    Case 1: t = "Removable"
    Case 2: t = "Fixed"
    Case 3: t = "Network"
    Case 4: t = "CD-ROM"
    Case 5: t = "RAM Disk"
 End Select
 MsgBox "Drive " & d.DriveLetter & ": - " & " is a " & t & " drive"
```

Property **FileSystem**
Description: Returns the type of file system in use for the specified drive.
Use: strType = *objDrive.*FileSystem
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: Available return types include FAT, NTFS, and CDFS.
Remarks: Read only.
Example:
```
Dim fso, d
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set d = fso.GetDrive(fso.GetDriveName("c:")
MsgBox "Drive " & d.DriveLetter & "  Files System type =" & d.FileSystem
```

Property **FreeSpace**
Description: Returns the amount of space available to a user on the specified drive or network share drive.
Use: intSpace = *objDrive.*FreeSpace
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: The read-only value returned by the **FreeSpace** property is typically the same as that returned by the **AvailableSpace** property. Differences may occur between the two for computer systems that support quotas.
Remarks: Read only.
Example:
```
Dim fso, d
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set d = fso.GetDrive(fso.GetDriveName("c:")
MsgBox "Free Space = " & d.FreeSpace/1024 & " Kbytes"
```

Property **IsReady**
Description: Indicates whether the specified drive is ready or not
Use: boolReady = *objDrive.*IsReady
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: Returns **True** if the specified drive is ready; **False** if it is not.
Remarks: Read only.
Example:
```
Dim fso, d, s
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
Set d = fso.GetDrive(fso.GetDriveName("c:")
s = "Drive " & d.DriveLetter
If d.IsReady Then
    s = s & " Drive is Ready."
Else
    s = s & " Drive is not Ready."
End If
MsgBox s
```

Property **Path**
Description: Returns the path for a specified drive.
Use: strPath = *objDrive.*Path
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: The path for a specified drive
Remarks: For drive letters, the root drive is not included. For example, the path for the C drive is C:, not C:\.
Example:
```
Dim fso, d
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
Set d = fso.GetDrive(fso.GetDriveName("c:"))
MsgBox "Path = " & UCase(d.Path)                        'Returns c:
```

Property **RootFolder**
Description: Returns a **Folder** object representing the root folder of a specified drive.
Use: objFolder = *objDrive.*RootFolder
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: The path for a specified drive
Remarks: Read-only. All the files and folders contained on the drive can be accessed using the returned **Folder** object.
Example:
```
Dim fso, d
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
Set d = fso.GetDrive("c:")
MsgBox "RootFolder = " & d.RootFolder                   'Returns "c:\"
```

Property **SerialNumber**
Description: Returns the decimal serial number used to uniquely identify a disk volume.
Use: intSerNum = *objDrive.*SerialNumber
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: A decimal serial number that uniquely identifies a disk volume
Remarks: Read-only. You can use the **SerialNumber** property to ensure that the correct disk is inserted in a drive with removable media.
Example:
```
Dim fso, d
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
Set d = fso.GetDrive("c:")
MsgBox "Drive Serial Number = " & d.SerialNumber        'Returns "c:\"
```

Property **ShareName**
Description: Returns the network share name for a specified drive.
Use: strName = *objDrive.*ShareName
Arguments: *objDrive*
Required. The name of a Drive Object previously instantiated.
Return: A string that is the network share name for a specified drive.
Remarks: Read-only. If *object* is not a network drive, the **ShareName** property returns a zero-length string ("").
Example:
```
Dim fso, dc, d
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
Set d = fso.GetDrive("c:")
If d.DriveType = 3 Then                                 'See if a network drive
    MsgBox "Network Shared Drive Name = " & d.ShareName
Else
    MsgBox "Not a Network Shared Drive"
End If
```

Property	**TotalSize**
Description:	Returns the total space, in bytes, of a drive or network shared drive.
Use:	intSize = *objDrive.*TotalSize
Arguments:	*objDrive* Required. The name of a Drive Object previously instantiated.
Return:	Integer. The total space, in bytes, of a drive or network shared drive
Remarks:	Read-only.
Example:	Dim fso, d Set fso = CreateObject("Scripting.FileSystemObject") 'Instantiate the FSO object Set d = fso.GetDrive("c:") MsgBox "Total Drive Size = " & d.TotalSize & " bytes" 'Returns the total size of the drive

Property	**VolumeName**
Description:	Sets or returns the volume name of the specified drive. Read/write.
Use:	*strName* = *objDrive.*VolumeName *objDrive.*VolumeName [= *newname*]
Arguments:	*objDrive* Required. The name of a Drive Object previously instantiated.. *newname* Optional. If provided, *newname* is the new name of the specified object
Return:	String. The volume name of the specified drive.
Remarks:	Read/Write.
Example:	Dim fso, d Set fso = CreateObject("Scripting.FileSystemObject") 'Instantiate the FSO object Set d = fso.GetDrive("c:") MsgBox "Total Drive Size = " & d.TotalSize & " bytes" 'Returns the total size of the drive

Notes:

- The comparable IWS built-in function to the **AvailableSpace** and **FreeSpace** property is **InfoDiskFree**
- There are no comparable IWS built-in functions to the rest of the Drive object Properties.

The Folder Object

The Folder Object allows the programmer refer to a specific folder. Once the Folder object is instantiated, it can be referred to as an object from VBScript and its various Methods and Properties accessed.

The Folder Object is instantiated as follows:

```
Dim fso, f, myPath
Set fso = CreateObject("Scripting.FileSystemObject")   'Instantiate the FSO Object
myPath = $GetAppPath() & "Hst"                         'Define the path to the folder of interest
Set f = fso.GetFolder(myPath)                          'Instantiate the Drive Object
```

See the **GetFolder** method under the FileSystemObject Object Model section for additional details on instantiation of the Folder Object.

The Folder object has both Methods and Properties available.

Table H: Folder Object Methods

Method	Description
Copy	Copies a specified folder from one location to another
CreateTextFile	Creates a specified file name and returns a TextStream object that can be used to read from or write to the file
Delete	Deletes a specified folder
Move	Moves a specified file or folder from one location to another.

Table I: Folder Object Properties

Properties	Description
Attributes	Sets or returns the attributes of files or folders.
DateCreated	Returns the date and time that the specified folder was created.
DateLastAccessed	Returns the date and time that the specified folder was last accessed
DateLastModified	Returns the date and time that the specified folder was last modified
Drive	Returns the drive letter of the drive on which the specified file or folder resides
Files	Returns a Files collection consisting of all File objects contained in the specified folder.
IsRootFolder	Tests to see if the specified folder is the root folder.
Name	Sets or returns the name of a specified file or folder
ParentFolder	Returns the folder object for the parent of the specified folder
Path	Returns the path for a specified folder
ShortName	Returns the short name used by programs that require the earlier 8.3 naming convention.
ShortPath	Returns the short path used by programs that require the earlier 8.3 naming convention.
Size	Returns the size of all the files and subfolders contained in the specified folder
SubFolders	Returns a Folders collection consisting of all folders contained in a specified folder,
Type	Returns information about the type of a folder.

Method **Copy**
Description: Copies a specified folder from one location to another.
Use: *objFolder.*Copy (*destination,* [*overwrite*])
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
destination
Required. Destination where the folder is to be copied. Wildcard characters are not allowed.
overwrite
Optional. Boolean value that is **True** (default) if existing folders are to be overwritten, **False** if they are not.
Return: None
Remarks: The results of the **Copy** method on a **Folder** are identical to operations performed using **FileSystemObject.CopyFolder** where the folder referred to by *object* is passed as an argument. You should note, however, that the alternative method is capable of copying multiple folders.
Example:

```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFolder = $getAppPath() & "Hst"                          'Application Folder for Historical files
Set f = fso.GetFolder (myFolder)
f.Copy (myFolder & "Temp")                                'Creates folder /HstTemp and copies files
```

Notes:
- This **Copy** method only copies an individual folder. The FSO **Copy** method will copy multiple folders.
- IWS does not have a comparable built-in Function

Method: **CreateTextFile**
Description: Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file
Use: Set objFile = *objFolde.***CreateTextFile**(*filename*[, *overwrite*[, *Unicode*]])
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
filename
Required. A string expression that identifies the file to create
overwrite
Optional. Boolean value that indicates whether you can overwrite an existing file. The value is **True** if the file can be overwritten, **False** if it can't be overwritten. If omitted, existing files are not overwritten (default **False**).
unicode
Optional. Boolean value that indicates whether the file is created as a Unicode or ASCII file. If the value is **True,** the file is created as a Unicode file. If the value is **False,** the file is created as an ASCII file. If omitted, an ASCII file is assumed.
Remarks: None
Example:

```
Dim fso, myFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set myFile = fso.CreateTextFile("c:\testfile.txt", True, False)
myFile.WriteLine("This is a test.")
myFile.Close
```

Notes:
- The **CreateTextFile** method allows you to create a text file for UniCode characters. Compare this to the IWS built-in FileWrite function which only supports ASCII files.
- The **CreateTextFile** method is available in either the FSO object or the Folder object
- Although the **CreateTextFile** method indicates that it will support reading, it does not appear to work. For reading to TextStream files, use the **OpenTextFile** or **OpenAsTextStream** methods.

Method: **Delete**
Description: Deletes a specified folder
Use: *objFolder.*Delete (*force*)
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
force
Optional. Boolean value that is **True** if folders with the read-only attribute set are to be deleted; **False** if they are not (default).
Return: None
Remarks: An error occurs if the specified folder does not exist. The results of the **Delete** method on a **Folder** are identical to operations performed using **FileSystemObject.DeleteFolder**. The **Delete** method does not distinguish between folders that have content and those that do not. The specified folder is deleted regardless of whether or not it has content.
Example:

```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFolder = $getAppPath() & "HstTemp"                      'Specify the HstTemp folder in app directory
Set f = fso.GetFolder (myFolder)
f.Delete                                                  'Delete it
```

Note:
- The FSO **DeleteFolder** method allows you to specify wildcard characters in the last path component. The Folder **Move** method and the IWS built-in function **DirDelete** only deletes one Folder at a time.

Method: **Move**
Description: Moves a specified folder from one location to another.
Use: *objFolder.*Move (*destination*)
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
destination
Required. Destination where the folder is to be moved. Wildcard characters are not allowed.
Return: None
Remarks: The results of the **Move** method on a **Folder** is identical to operations performed using **FileSystemObject.MoveFolder**. You should note, however, that the alternative methods are capable of moving multiple folders.
Example:

```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFolder = $getAppPath() & "HstTemp"                      'Specify the HstTemp folder in app directory
Set f = fso.GetFolder (myFolder)
f.move("c:\archive")                                      'Move it into c:\archive folder
```

Notes:
- The comparable IWS built-in function is **DirRename.**
- The FSO **MoveFolder** method allows moving folders between volumes only if supported by the operating system. .
- You can use the Folder Object **Move** method to move an individual folder. The FSO **Move** method supports moving multiple folders.

Property: **Attributes**
Description: Sets or returns the attributes of files or folders.
Use: *objFolder*.Attributes = newAttributes
intAttribute = *objFolder*.Attributes
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
newAttributes
Optional. If provided, newAttributes is the new value for the attributes of the specified *object*. The newattributes argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No Attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.
Hidden	2	Hidden file. Attribute is read/write.
System	4	System file. Attribute is read/write.
Volume	8	Disk drive volume label. Attribute is read-only
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write
Alias	1024	Link or shortcut. Attribute is read-only
Compressed	2048	Compressed file. Attribute is read-only.

Return: Can return an attribute of a file or folder
Remarks: Read/write or read-only, depending on the attribute. The newAttribute can have any valid combination of the above values.
Example:

```
Dim fso, f, attrVal, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
attrVal = f.Attributes
attrVal = attrVal And 16                                'See if a folder
If attrVal = 16 Then
    MsgBox "Object is a folder"
Else
    MsgBox "Object is not a folder"
End If
```

Property: **DateCreated**
Description: Returns the date and time that the specified folder was created.
Use: *objFolder*.DateCreated
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: None
Remarks: Read-only.
Example:

```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "App Directory created on " & f.DateCreated
```

Property: **DateLastAccessed**
Description: Returns the date and time that the specified folder was last accessed
Use: *objFolder.*DateLastAccessed
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: None
Remarks: Read-only.
Example:
```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFolder = $getAppPath()                                 'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "App Directory was last accessed on " & f.DateLastAccessed
```

Property: **DateLastModified**
Description: Returns the date and time that the specified folder was last modified
Use: *objFolder.*DateLastModified
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: None
Remarks: Read-only.
Example:
```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFolder = $getAppPath()                                 'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "App Directory was last modified on " & f.DateLastModified
```

Property: **Drive**
Description: Returns the drive letter of the drive on which the specified folder resides
Use: *objFolder.*Drive
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: None
Remarks: Read-only.
Example:
```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFolder = $getAppPath()                                 'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "App Directory is installed on drive  " & f.Drive    'Installed on drive c:
```

Property: **Files**
Description: Returns a Files collection consisting of all File objects contained in the specified folder.
Use: *objFolder.*Files
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: A file collection.
Remarks: Includes files with hidden and system file attributes set.
Example:

```
Dim fso, f, fc, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
fc = f.files                                            'Return file collection of files in app folder
```

Property: **IsRootFolder**
Description: Tests to see if the specified folder is the root folder.
Use: boolValue = *objFolder.*IsRootFolder
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: **True** if the specified folder is the root folder; **False** if not.
Remarks: Includes files with hidden and system file attributes set.
Example:

```
Dim fso, f, n, s, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
n = 0
If f.IsRootfolder Then
    MsgBox "The app folder is the root folder"
Else
    s = myFolder & vbCrLf
    Do Until f.IsRootFolder
        Set f = f.ParentFolder
        n = n+1
        s = s & "parent folder is " & f.Name & vbCrLf
    Loop
    MsgBox "Folder was nested " & n & " levels" & vbCrLf & s
End If
```

Property: **Name**
Description: Sets or returns the name of a specified folder
Use: *objFolder*.Name = *newName*
strName = *objFolder*.Name
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
newName
Optional. If provided, *newName* is the new name of the specified folder object
Return: The name of the specified folder.
Remarks: Read/write.
Example:

```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "folder name is " & f.Name                       'Returns the folder name
```

Property: **ParentFolder**
Description: Returns the folder object for the parent of the specified folder
Use: objParent = *objFolder.*ParentFolder
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: The folder object for the parent of the specified folder.
Remarks: Read-only
Example:
```
Dim fso, f, pf, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
Set pf = f.ParentFolder                                 'Get the parent folder
MsgBox "Parent Folder name = " & pf.Name
```

Property **Path**
Description: Returns the path for a specified folder
Use: strPath = *objFolder.*Path
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: The path for a specified folder
Remarks: None
Example:
```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "Path = " & UCase(f.Path)                        'Display path to app folder
```

Property **ShortName**
Description: Returns the short name used by programs that require the earlier 8.3 naming convention.
Use: strName = *objFolder.*ShortName
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: The short name for the folder object
Remarks: None
Example:
```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "Short name = " & f.ShortName                    'Display short name of app folder
```

Property **ShortPath**
Description: Returns the short path used by programs that require the earlier 8.3 naming convention.
Use: strPath = *objFolder.*ShortPath
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: The short path for the folder object
Remarks: None
Example:
```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFolder = $getAppPath()                                'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "Short pathname = " & f.ShortPath                'Display short path of app folder
```

Property **Size**
Description: Returns the size of all the files and subfolders contained in the specified folder
Use: intSize = *objFolder.*Size
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: The size of the specified folder
Remarks: Size is in bytes
Example:

```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFolder = $getAppPath()                                  'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "Size = " & f.Size & " bytes"                      'Display size of app folder
```

Property **SubFolders**
Description: Returns a **Folders** collection consisting of all folders contained in a specified folder,
Use: objFC = *objFolder.*SubFolders
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: A folders collection of all subfolders in a specified folder.
Remarks: Includes folders with hidden and system file attributes set.
Example:

```
Dim fso, f, fc, s, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFolder = $getAppPath()                                  'Specify the app directory
Set f = fso.GetFolder (myFolder)
fc = f.Subfolders                                         'Returns collection of (sub)folders
s = ""
For each f1 in fc
    s = s & fc.name & vbCrLf
Next
MsgBox s
```

Property **Type**
Description: Returns information about the type of a folder.
Use: strType = *objFolder.*Type
Arguments: *objFolder*
Required. The name of a Folder Object previously instantiated.
Return: The type of folder.
Remarks: If the object is a folder, "Folder" will be returned.
Example:

```
Dim fso, f, myFolder
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFolder = $getAppPath()                                  'Specify the app directory
Set f = fso.GetFolder (myFolder)
MsgBox "Type = " & f.Type                                 'Dispays "Folder"
```

Notes:

- Many of the Folder object Properties have no corresponding IWS built-in function. Many are, however, of little use in a typical IWS application.
- Note that two of the Folder object Properties return collections: the **SubFolders** property which returns a collection of (sub)folders within specific folder, and the **Files** property which returns a collection of files contained within a specific folder. These collections are not sorted and do not have any mask criteria (e.g. date, type, etc.). Compare this to the IWS built-in function **FindFile** which can return a collection of files has satisfy a file mask criteria. However, collections returned by the **SubFolders** and **Files** properties can be further manipulated in VBScript by examining other attribute properties.

The File Object

The File Object allows the programmer refer to a specific file. Once the File object is instantiated, it can be referred to as an object from VBScript and its various Methods and Properties accessed.

The File Object is instantiated as follows:

```
Dim fso, f, myPath
Set fso = CreateObject("Scripting.FileSystemObject")   'Instantiate the FSO Object
myPath = $GetAppPath() & "notes.txt"                   'Define the path to the file of interest
Set f = fso.GetFile(myPath)                            'Instantiate the Drive Object
```

See the **GetFile** method under the FileSystemObject Object Model section for additional details on instantiation of the File Object.

The File object has both Methods and Properties available.

Table J: File Object Methods

Method	Description
Copy	Copies a specified folder from one location to another
Delete	Deletes a specified folder
Move	Moves a specified file or folder from one location to another.
OpenAsTextStream	Creates a specified file name and returns a TextStream object that can be used to read from or write to the file

Table K: File Object Properties

Properties	Description
Attributes	Sets or returns the attributes of files or folders.
DateCreated	Returns the date and time that the specified folder was created.
DateLastAccessed	Returns the date and time that the specified folder was last accessed
DateLastModified	Returns the date and time that the specified folder was last modified
Drive	Returns the drive letter of the drive on which the specified file or folder resides
Name	Sets or returns the name of a specified file or folder
ParentFolder	Returns the folder object for the parent of the specified folder
Path	Returns the path for a specified folder
ShortName	Returns the short name used by programs that require the earlier 8.3 naming convention.
ShortPath	Returns the short path used by programs that require the earlier 8.3 naming convention.
Size	Returns the size of all the files and subfolders contained in the specified folder
Type	Returns information about the type of a folder.

Method **Copy**
Description: Copies a specified file from one location to another.
Use: *objFile.*Copy (*destination,* [*overwrite*])
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
destination
Required. Destination where the File is to be copied. Wildcard characters are not allowed.
overwrite
Optional. Boolean value that is **True** (default) if existing files are to be overwritten, **False** if they are not.
Return: None
Remarks: The results of the **Copy** method on a **File** are identical to operations performed using **FileSystemObject.CopyFile** where the file referred to by *object* is passed as an argument. You should note, however, that the alternative method is capable of copying multiple files.
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                  'Get the file object
Set f = fso.GetFile (myFile)
f.Copy ("c:\save\recipe1.xml")                          'Save the file
```

Note:
- The comparable IWS built-in function is **FileCopy**
- The FSO **CopyFile** method allows use of wildcards to copy multiple files. The File object **Copy** method only copies a single file.

Method: **Delete**
Description: Deletes a specified file
Use: *objFile.*Delete (*force*)
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
force
Optional. Boolean value that is **True** if files with the read-only attribute set are to be deleted; **False** if they are not (default).
Return: None
Remarks: An error occurs if the specified file does not exist. The results of the **Delete** method on a **File** are identical to operations performed using **FileSystemObject.DeleteFile**. The **Delete** method does not distinguish between files that have content and those that do not. The specified file is deleted regardless of whether or not it has content.
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                  'Specify the HstTemp folder in app directory
Set f = fso.GetFile (myFile)
f.Delete                                                'Delete it
```

Note:
- The comparable IWS built-in function is **FileDelete**
- The FSO **DeleteFile** method allows use of wildcards to delete multiple files. The File object **Delete** method only deletes a single file.

Method: **Move**
Description: Moves a specified file from one location to another.
Use: *objFile.*Move (*destination*)
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
destination
Required. Destination where the file is to be moved. Wildcard characters are not allowed.
Return: None
Remarks: The results of the **Move** method on a **File** is identical to operations performed using **FileSystemObject.MoveFile**. You should note, however, that the alternative methods are capable of moving multiple files.
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                    'Specify the HstTemp folder in app directory
Set f = fso.GetFile (myFile)
f.move("Recipe1Save.xml")                                 'Moves the file
```

Note:
- The comparable IWS built-in function is **FileRename**
- The FSO **MoveFile** method allows use of wildcards to move multiple files. The File object **Move** method only moves a single file.

Method: **OpenAsTextStream**
Description: Opens a specified file name and returns a **TextStream** object that can be used to read from or write to, or append to a file
Use: oTso = *oFile*.OpenAsTextStream([*iomode[,format]]*)
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
iomode
Optional. Indicates the file input/output mode. Can be one of three constants:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file
ForWriting	2	Open a file for reading & writing
ForAppending	8	Open a file and write to the end of the file

format
Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default
TristateTrue	-1	Opens the file as Unicode
TrstateFalse	0	Opens the file as ASCII

Return: A **TextStream** object
Remarks The **OpenAsTextStream** method provides the same functionality as the **OpenTextFile** method of the **FileSystemObject**. In addition, the **OpenAsTextStream** method can be used to write to a file.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, tso
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.GetFile("c:\testfile.txt")                    'Must be an existing file
Set tso = f.OpenAsTextStream(ForWriting, True)            'Unicode file
tso.Write "Hello world!"                                  'Write a line of text to the file
tso.Close
```

Notes:
- To create a TextFile, you need to use the CreateTextFile method from the FSO Object or Folder Object. Additionally, you could use the OpenTextFile method of the FSO object.
- The OpenAsTextStream method only works on existing files.
- As with the CreateTextFile and OpenTextFile methods, the OpenAsTextStream method supports UniCode characters.

Property: **Attributes**
Description: Sets or returns the attributes of files or folders.
Use: *objFile.*Attributes = newAttributes
intAttribute = *objFile*.Attributes
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
newAttributes
Optional. If provided, newAttributes is the new value for the attributes of the specified *object*. The newattributes argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No Attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.
Hidden	2	Hidden file. Attribute is read/write.
System	4	System file. Attribute is read/write.
Volume	8	Disk drive volume label. Attribute is read-only
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write
Alias	1024	Link or shortcut. Attribute is read-only
Compressed	2048	Compressed file. Attribute is read-only.

Return: Can return an attribute of a file or folder
Remarks: Read/write or read-only, depending on the attribute. The newAttribute can have any valid combination of the above values.
Example:

```
Dim fso, f, attrVal, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory and file
Set f = fso.GetFile(myFile)
attrVal = f.Attributes
attrVal = attrVal And 1                                  'See if a normal file
If attrVal = 0 Then
    MsgBox "Object is a normal file"
Else
    MsgBox "Object is not a normal file"
End If
```

Property: **DateCreated**
Description: Returns the date and time that the specified file was created.
Use: *objFile.*DateCreated
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: None
Remarks: Read-only.
Example:
```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "File created on " & f.DateCreated
```

Property: **DateLastAccessed**
Description: Returns the date and time that the specified file was last accessed
Use: *objFile.*DateLastAccessed
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: None
Remarks: Read-only.
Example:
```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "File was last accessed on " & f.DateLastAccessed
```

Property: **DateLastModified**
Description: Returns the date and time that the specified file was last modified
Use: *objFile.*DateLastModified
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: None
Remarks: Read-only.
Example:
```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "File was last modified on " & f.DateLastModified
```

Property: **Drive**
Description: Returns the drive letter of the drive on which the specified file resides
Use: *objFile.*Drive
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: None
Remarks: Read-only.
Example:
```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "File is located on drive  " & f.Drive            'Installed on drive c:
```

Property: **Name**
Description: Sets or returns the name of a specified file
Use: *objFile.*Name = *newName*
strName = *objFile.*Name
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
newName
Optional. If provided, *newName* is the new name of the specified file object
Return: The name of the specified file.
Remarks: Read/write.
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "file name is " & f.Name                          'Returns the file name
```

Property: **ParentFolder**
Description: Returns the folder object for the parent of the specified file
Use: objFolder = *objFile.*ParentFolder
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: The folder object for the parent folder of the specified file.
Remarks: Read-only
Example:

```
Dim fso, f, pf, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
Set pf = f.ParentFolder                                  'Get the parent folder
MsgBox "Parent Folder name = " & pf.Name
```

Property **Path**
Description: Returns the path for a specified file
Use: strPath = *objFile.*Path
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: The path for a specified file
Remarks: None
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "Path = " & UCase(f.Path)                         'Display path to app file
```

Property **ShortName**
Description: Returns the short name used by programs that require the earlier 8.3 naming convention.
Use: strName = *objFile.*ShortName
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: The short name for the file object
Remarks: None
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                  'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "Short name = " & f.ShortName                    'Display short name of app file
```

Property **ShortPath**
Description: Returns the short path used by programs that require the earlier 8.3 naming convention.
Use: strPath = *objFile.*ShortPath
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: The short path for the file object
Remarks: None
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                  'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "Short name = " & f.ShortPath                    'Display short path of app file
```

Property **Size**
Description: Returns the size of the specified file
Use: intSize = *objFile.*Size
Arguments: *objFile*
Required. The name of a File Object previously instantiated.
Return: The size of the specified file
Remarks: Size is in bytes
Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                  'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "Size = " & f.Size & " bytes"                    'Display size of file
```

Property	**Type**
Description:	Returns information about the type of a file.
Use:	strType = *objFile.*Type
Arguments:	*objFile* Required. The name of a File Object previously instantiated.
Return:	The type of file.
Remarks:	E.g. for files ending in .TXT, "Text Document" is returned.

Example:

```
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "recipe1.xml"                   'Specify the app directory & file
Set f = fso.GetFile (myFile)
MsgBox "Type = " & f.Type                                'Dispays "XML Document"
```

Notes:

- Many of the File object Properties have no corresponding IWS built-in function. Many are, however, of little use in a typical IWS application.

The TextStream Object

The TextStream Object allows the programmer to sequentially access a text file. Once the TextStream object is instantiated, it can be referred to as an object from VBScript and its various Methods and Properties accessed.

The TextStream object can be instantiated in three different ways. These are

- Through the **CreateTextFile** method of the FSO object
- Through the **OpenTextFile** method of the FSO object
- Through the **OpenAsTextStream** method of the File Object

There are subtle differences between these methods. The **CreateTextFile** is used to create a file and a TextStream object. This method can optionally overwrite an existing object. The **OpenTextFile** opens an existing file and returns a TextStream object, but can optionally create the filename if it does not exist. The **OpenAsTextStream** object opens an existing file and returns a TextStream object. This method gives an error if the text file does not exist, there is no option to create the file if it does not exist. Another difference is that the **CreateTextFile** method opens a TextStream object for reading and writing, while the **OpenTextFile** and **OpenAsTextStream** methods open a TextStream object for reading, writing or appending.

Examples of the various approaches to instantiating the TextStream object are:

Instantiating a TextStream object with the CreateTextFile Method

```
Dim fso, f, myfile
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.CreateTextFile(myFile, True, True)            'Open as UniCode TextStream object
```

Instantiating a TextStream object with the OpenTextFile Method

```
Constant forReading = 1, forWriting = 2, forAppending = 8
Dim fso, myfile, tso
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set tso = fso.OpenTextFile(myFile, ForWriting, True, True)  'Open as UniCode TextStream object
```

Instantiating a TextStream object with the OpenAsTextStream Method

```
Constant forReading = 1, forWriting = 2, forAppending = 8
Dim fso, f, myfile, tso
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.GetFile(myFile)                               'Instantiate the file object
Set tso = f.OpenAsTextStream(forAppending, True)          'Open as UniCode TextStream object
```

See the **CreateTextFile** and **OpenTextFile** methods under the FileSystemObject Object Model section for additional details on instantiation of the TextStream Object. See the **OpenAsTextStream** method under the File Object section for additional details on instantiation of the TextStream Object

The TextStream object supports either ASCII or UniCode characters, according to the argument settings when calling the method used to instantiate the TextStream object.

The TextStream object has both Methods and Properties available.

Note:
- Although Microsoft documentation indicates that the **CreateTextFile** method supports text file reading, no examples of this are documented and all read methods for an object created by the **CreateTextFile** method fail. It is recommended to use either the **OpenTextFile** or **OpenAsTextStream** method for a text file read method.

Table L: TextStream Object Methods

Method	Description
Close	Closes an open TextStream file
Read	Reads a specified number of characters from a TextStream file and returns a resulting string.
ReadAll	Reads an entire TextStream file and returns a resulting string. Note: this is an inefficient way to read a file. Use ReadLine Method instead.
ReadLine	Reads an entire line from a TextStream file and returns a resulting string. Reads up to but not including the newline character.
Skip	Skips a specified number of characters when reading a TextStream file. Skipped characters are discarded.
SkipLine	Skips the next line when reading a TextStream file. This method actually reads then discards all characters in a line up to and including the next newline character. An Error occurs if the file is not open for reading.
Write	Writes a specified string to a TextStream file. The specified string is written with no intervening spaces or characters between each string written. To write lines of text, use either a string that ends with a newline character or use the WriteLine method.
WriteLine	Writes a specified string and new line character to a TextStream file.
WriteBlankLines	Writes a specified number of newline characters to a TextStream file.

Table M: TextStream Object Properties

Property	Description
AtEndOfLine	Returns a value of True of the file pointer immediately preceeds the end of line marker in a TextStream file. Otherwise returns a value of False.
AtEndOfStream	Returns a value of True if the file pointer is at the end of the File, otherwise returns False
Column	Depending on the
Line	

Notes:
- When reading or writing files, remember that files can only be read or written to sequentially
- A file object cannot be open simultaneously for both reading and writing. However, you can use two objects, one for reading and one for writing, to a file. For example:

```
Const ForReading = 1, ForWriting = 2, ForAppending = 8
Dim f, f1, fso, tso, myFile, s
Set fso = CreateObject("Scripting.FileSystemObject")
myFile = $getAppPath() & "notes.txt"          'Specify the app directory & file"
Set f = fso.OpenTextFile(myFile, ForReading)  'Use OpenTextFile method for reading
Set f1 = fso.GetFile(myFile)                  'Instantiate a file object
Set tso = f1.OpenAsTextStream(ForAppending)   'Instantiate a TextStream object for writing
s = f.ReadAll
MsgBox "Line count = " & f.Line & vbCrLf & s
tso.WriteLine "this is a line of appended data"
s = f.ReadAll
MsgBox "Line count = " & f.Line & vbCrLf & s  'Will only display the line of appended data
```

Method: **Close**
Description: Closes an open TextStream file
Use: *objTso.*Close
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
Return: None
Remarks: The Close method closes the file, but still need to set the object variable to Nothing to release memory. (e.g. **"**Set objTso = Nothing"
Example:

```
Dim fso, f, myfile
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.CreateTextFile(myFile, True)
f.WriteLine ("this is a note")
f.Close                                                   'Close the document
```

Method: **Read**
Description: Reads a specified number of characters from a **TextStream** file and returns the resulting string.
Use: strChars = *objTso.*Read(*numCharacters*)
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
numCharacters
Required. The number of characters you want to read from the file
Return: A specified number of characters from the file
Remarks: None
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading)
s = f.Read(10)                                            'Read 10 characters
MsgBox "First 10 characters = " & s                       'Display
f.Close                                                   'Close the document
```

Method: **ReadAll**
Description: Reads the entire **TextStream** file and returns the resulting string.
Use: strChars = *objTso.*ReadAll
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
Return: The entire TextStream file.
Remarks: VBScript does not have a limit on the resultant character string length other than the available memory.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading)
s = f.ReadAll                                             'Read entire file
MsgBox "File contents = " & s                             'Display it
f.Close
```

Notes:
- The **ReadAll** method inefficiently uses memory for large text files. Other methods, such as **ReadLine** are recommended instead.

Method: **ReadLine**

Description: Reads an entire line (up to, but not including, the newline character) from a **TextStream** file and returns the resulting string.

Use: strChars = *objTso.*ReadLine

Arguments: *objTso*

Required. The name of a TextStream Object previously instantiated.

Return: An entire line from a TextStream file

Remarks: Does not include the newline character. Successive calls to the ReadLine method do not return any newline character(s). For display purposes, you must add a newline character

Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s, linecount
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading)
linecount = 0
s = ""
Do While f.AtEndOfStream <> True
    linecount = linecount +1
    s = s & "line " & linecount & "  " & f.ReadLine & vbCrLf        'Read a line at a time
Loop
MsgBox s                                                  'Display it
f.Close
```

Note:

- Another (simpler) approach to this example would be to use the following:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s, linecount
Set fso = CreateObject("Scripting.FileSystemObject")              'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                              'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading)
s = f.ReadAll
linecount = f.Line
MsgBox "# of lines = " & linecount & VbCrLf & "Data" & VbCrLf & s         'Display information
f.Close
```

Notes:

- IWS includes a built-in function **GetLine** which searches for a specific string, then returns the whole line.
- IWS has a limit of 256 characters for a string tag, which is where the line of text from a file is stored. VBScript by comparison, has no limitation on the size of the character string other than the available system memory.
- The TextStream object **Read, ReadAll** and **ReadLine** methods read a character (or number of characters) at a time, a line at a time, or the whole file at once. Following whichever Read method is used, VBScript's character operations can search the string for a specific character sequence.

Method: **Skip**
Description: Skips a specified number of characters when reading a **TextStream** file
Use: *objTso.Skip*(*numCharacters*)
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
numCharacters
Required. The number of characters you want to skip when reading a file
Return: None
Remarks: Skipped characters are discarded.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading)
f.Skip(5)                                                 'Skip 5 characters
MsgBox f.ReadLine                                         'Read the rest of the line
f.Close                                                   'Close the document
```

Notes:

- If you use the **Skip** method followed by a **ReadLine** method, the remained of a line (up to, but not including, the newline character will be read)

Method: **SkipLine**
Description: Skips the next line when reading from a **TextStream** file.
Use: *objTso.*SkipLine
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
Return: None
Remarks: The skipped line is discarded.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s
Set fso = CreateObject("Scripting.FileSystemObject")      'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                      'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading)
f.SkipLine                                                'Skip the first line
s=f.ReadLine
MsgBox s                                                  'Display the second line
f.Close
```

Method: **Write**
Description: Writes a specified string to a **TextStream** file.
Use: *objTso.*Write(*string)*
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
string
Required. The text you want to write to the file.
Return: None
Remarks: Specified strings are written to the file with no intervening spaces or characters between each string. Use the **WriteLine** method to write a newline character or a string that ends with a newline character.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myFile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                     'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForWriting, True)
f.Write "This is a new string of data"                   'Write a string
Set f = fso.OpenTextFile(myFile, ForReading)
MsgBox "File contents = " & f.ReadLine                   'Display line of data
f.Close
```

Notes:
- The corresponding IWS built-in function is **FileWrite**.
- The IWS **FileWrite** function includes a parameter specifying whether to overwrite or appending text. With the TextStream object, the choice of overwriting or appending text is specified in the **OpenTextFile** or **OpenAsTextStream** method.

Method: **WriteBlankLines**
Description: Writes a specified number of newline characters to a **TextStream** file.
Use: *objTso.*WriteBlankLines(*numLines*)
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
numLines
Required. The number of newline characters you want to write to the file.
Return: None
Remarks: None
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                     'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForWriting, True)
f.WriteBlankLines(3)                                     'Write 3 blank lines
f.WriteLine "This is a new line of data"                 'Write data on the 4th line
f.Close
```

Method: **WriteLine**
Description: Writes a specified string and newline character to a **TextStream** file.
Use: *objTso.*WriteLine([*string*])
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
string
Optional. The text you want to write to the file.
Return: None
Remarks: If you omit the *string*, a newline character is written to the file.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                     'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForWriting, True)
f.WriteLine "This is a line of data"                     'Write a line of data
f.WriteLine                                              'Write a blank line
f.Close
```

Property: **AtEndOfLine**
Description: Indicates whether the file pointer is positioned immediately before the end-of-line marker in a **TextStream** file.
Use: *objTso.*AtEndOfLine
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
Return: Returns **True** if the file pointer is positioned immediately before the end-of-line marker in a **TextStream** file; **False** if it is not.
Remarks: The **AtEndOfLine** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s
Set fso = CreateObject("Scripting.FileSystemObject")     'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                     'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading, False)
s =""
Do While f.AtEndOfLine <> True
    s=f.read(1)                                          'Read one character at a time
Loop
MsgBox "A line of text = " & s
f.Close
```

Property: **AtEndOfStream**
Description: Indicates whether the file pointer is positioned at the end of a **TextStream** file.
Use: *objTso.*AtEndOfStream
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
Return: Returns **True** if the file pointer is positioned at the end of a **TextStream** file; **False** if it is not.
Remarks: The **AtEndOfStream** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                    'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading, False)
s =""
Do While f.AtEndOfLine <> True
    s = s & f.ReadLine                                  'Read file one line at a time
Loop
MsgBox s                                                'Display text
f.Close
```

Property: **Column**
Description: Returns the column number of the current character position in a **TextStream** file.
Use: intColumnPos = *objTso.*Column
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
Return: An integer column number
Remarks: Read-only. After a newline character has been written, but before any other character is written, **Column** is equal to 1.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s, colNum
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                    'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading, False)
s = f.ReadLine                                          'Read a line
colNum = f.Column                                       'Get the column position
f.Close
```

Property: **Line**
Description: Returns the current line number in a **TextStream** file.
Use: intLineNum = *objTso.*Line
Arguments: *objTso*
Required. The name of a TextStream Object previously instantiated.
Return: An integer line number
Remarks: Read-only. After a file is initially opened and before anything is written, **Line** is equal to 1.
Example:

```
Const ForReading=1, Const ForWriting=2, ForAppending=8
Dim fso, f, myfile, s, lineNum
Set fso = CreateObject("Scripting.FileSystemObject")    'Instantiate the FSO object
myFile = $getAppPath() & "notes.txt"                    'Specify the app directory & file
Set f = fso.OpenTextFile(myFile, ForReading, False)
s = f.ReadAll                                           'Read the entire file
lineNum = f.Line                                        'Get the last line number
f.Close
```

Note:

- Since IWS does not have a comparable TextStream object, many of the TextStream object Properties have no corresponding function in IWS.

Summary

IWS has a number of specific built-in functions that work well for typical IWS applications. The FSO object model provides several objects, methods and properties that allow generic manipulation of drives, folder, files and text files to support a wide range of applications where VBScript is used. As shown in this Application Note and Application Note AN-00-005, the choice of whether to use an IWS built-in function or a FSO function depends on the operation to be performed. In addition, the combination of calling IWS built-in functions from VBScript to be used as parameters for a FSO method or property is quite powerful, performing operations that neither approach alone would easily support.