Scripting Runtime Library

Dictionary Object

Object that stores data key, item pairs.

Remarks

A **Dictionary** object is the equivalent of a PERL associative array. Items can be any form of data, and are stored in the array. Each item is associated with a unique key. The key is used to retrieve an individual item and is usually a integer or a string, but can be anything except an array.

The following code illustrates how to create a **Dictionary** object:

```
[JScript]
var y = new ActiveXObject("Scripting.Dictionary");
y.add ("a", "test");
if (y.Exists("a"))
   document.write("true");
...
[VBScript]
Dim d   ' Create a variable.
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"   ' Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
```

Methods

Add Method (Dictionary) | Exists Method | Items Method | Keys Method | Remove Method | RemoveAll Method

Properties

Count Property | Item Property | Key Property

See Also

FileSystemObject Object | TextStream Object

Scripting Runtime Library

Count Property

Returns the number of items in a collection or **Dictionary** object. Read-only.

object.**Count**

The *object* is always the name of one of the items in the Applies To list.

Remarks

The following code illustrates use of the **Count** property:

```
[JScript]
function CountDemo()
{
   var a, d, i, s;                  // Create some variables.
   d = new ActiveXObject("Scripting.Dictionary");
   d.Add ("a", "Athens");           // Add some keys and items.
   d.Add ("b", "Belgrade");
   d.Add ("c", "Cairo");
   a = (new VBArray(d.Keys()));    // Get the keys.
   s = "";
   for (i = 0; i < d.Count; i++)   //Iterate the dictionary.
   {
      s += a.getItem(i) + " - " + d(a.getItem(i)) + "<br>";
   }
   return(s);                       // Return the results.
}
[VBScript]
Function ShowKeys
   Dim a, d, i, s   ' Create some variables.
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "a", "Athens"   ' Add some keys and items.
   d.Add "b", "Belgrade"
   d.Add "c", "Cairo"
   a = d.Keys   ' Get the keys.
   For i = 0 To d.Count -1 ' Iterate the array.
      s = s & a(i) & "<BR>" ' Create return string.
   Next
   ShowKeys = s
End Function
```

See Also

CompareMode Property | Item Property | Key Property

Applies To: Dictionary Object | Drives Collection | Files Collection | Folders Collection

---

Scripting Runtime Library

Item Property

Sets or returns an *item* for a specified *key* in a **Dictionary** object. For collections, returns an *item* based on the specified *key*. Read/write.

```
object.Item(key)[ = newitem]
```

Arguments

object
Required. Always the name of a collection or **Dictionary** object.

key
Required. *Key* associated with the *item* being retrieved or added.

newitem
Optional. Used for **Dictionary** object only; no application for collections. If provided, *newitem* is the new value associated with the specified *key*.

Remarks

If *key* is not found when changing an *item*, a new *key* is created with the specified *newitem*. If *key* is not found when attempting to return an existing item, a new *key* is created and its corresponding item is left empty.

The following example illustrates the use of the **Item** property.

```
[JScript]
function DicTest(keyword)
{
   var a, d;
   d = new ActiveXObject("Scripting.Dictionary");
   d.Add("a", "Athens");
   d.Add("b", "Belgrade");
   d.Add("c", "Cairo");
   a = d.Item(keyword);
   return(a);
}
[VBScript]
Function ItemDemo
   Dim d   ' Create some variables.
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "a", "Athens"   ' Add some keys and items.
   d.Add "b", "Belgrade"
   d.Add "c", "Cairo"
   ItemDemo = d.Item("c")   ' Get the item.
End Function
```

See Also

CompareMode Property | Count Property | Key Property

Applies To: Dictionary Object | Drives Collection | Files Collection | Folders Collection

---

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Key Property

Sets a *key* in a **Dictionary** object.

```
object.Key(key) = newkey
```

Arguments

object
: Required. Always the name of a **Dictionary** object.

key
: Required. *Key* value being changed.

newkey
: Required. New value that replaces the specified *key*.

Remarks

If *key* is not found when changing a *key*, a new *key* is created and its associated *item* is left empty.

The following example illustrates the use of the **Key** property:

```
[JScript]
var d;
d = new ActiveXObject("Scripting.Dictionary");
```

```
function AddStuff()
{
   var a;
   d.Add("a", "Athens");
   d.Add("b", "Belgrade");
   d.Add("c", "Cairo");
}

function ChangeKey(oldkey, newkey)
{
   var s;
   d.Key("c") = "Ca";
   s = "Key " + oldkey + " changed to " + newkey;
   return(s);
}
[VBScript]
Function DicDemo
   Dim d   ' Create some variables.
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "a", "Athens"   ' Add some keys and items.
   d.Add "b", "Belgrade"
   d.Add "c", "Cairo"
   d.Key("c") = "d"   ' Set key for "c" to "d".
   DicDemo = d.Item("d")   ' Return associate item.
End Function
```

See Also

CompareMode Property | Count Property | Item Property

Applies To: Dictionary Object

Scripting Runtime Library

Add Method (Dictionary)

Adds a key and item pair to a **Dictionary** object.

```
object.Add (key, item)
```

Arguments

object
: Required. Always the name of a **Dictionary** object.

key
: Required. The *key* associated with the *item* being added.

item
: Required. The *item* associated with the *key* being added.

Remarks

An error occurs if the *key* already exists.

The following example illustrates the use of the **Add** method.

```
[JScript]
var d;
d = new ActiveXObject("Scripting.Dictionary");
d.Add("a", "Athens");
d.Add("b", "Belgrade");
d.Add("c", "Cairo");
[VBScript]
Dim d   ' Create a variable.
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"   ' Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
```

See Also

Add Method (Folders) | Exists Method | Items Method | Keys Method | Remove Method | RemoveAll Method

Applies To: Dictionary Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Exists Method

Returns **true** if a specified key exists in the **Dictionary** object, **false** if it does not.

object.**Exists(***key***)**

Arguments

object
  Required. Always the name of a **Dictionary** object.

key
  Required. *Key* value being searched for in the **Dictionary** object.

Remarks

The following example illustrates the use of the **Exists** method.

```
[JScript]
function keyExists(k)
{
   var fso, s = "";
   d = new ActiveXObject("Scripting.Dictionary");
   d.Add("a", "Athens");
   d.Add("b", "Belgrade");
   d.Add("c", "Cairo");
   if (d.Exists(k))
```

```
      s += "Specified key exists.";
   else
      s += "Specified key doesn't exist.";
   return(s);
}
[VBScript]
Function KeyExistsDemo
   Dim d, msg   ' Create some variables.
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "a", "Athens"   ' Add some   keys and items.
   d.Add "b", "Belgrade"
   d.Add "c", "Cairo"
   If d.Exists("c") Then
      msg = "Specified key exists."
   Else
      msg = "Specified key doesn't exist."
   End If
   KeyExistsDemo = msg
End Function
```

See Also

Add Method (Dictionary) | Items Method | Keys Method | Remove Method | RemoveAll Method
Applies To: Dictionary Object

Scripting Runtime Library

Items Method

Returns an array containing all the items in a **Dictionary** object.

```
object.Items( )
```

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **Items** method:

```
[JScript]
function ItemsDemo()
{
   var a, d, i, s;                  // Create some variables.
   d = new ActiveXObject("Scripting.Dictionary");
   d.Add ("a", "Athens");              // Add some keys and items.
   d.Add ("b", "Belgrade");
   d.Add ("c", "Cairo");
   a = (new VBArray(d.Items())).toArray();   // Get the items.
   s = "";
   for (i in a)                  // Iterate the dictionary.
   {
      s += a[i] + "<br>";
   }
   return(s);                     // Return the results.
}
[VBScript]
Function DicDemo
   Dim a, d, i, s   ' Create some variables.
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "a", "Athens"   ' Add some keys and items.
   d.Add "b", "Belgrade"
   d.Add "c", "Cairo"
   a = d.Items   ' Get the items.
   For i = 0 To d.Count -1 ' Iterate the array.
      s = s & a(i) & "<BR>" ' Create return string.
   Next
   DicDemo = s
End Function
```

See Also

Add Method (Dictionary) | Exists Method | Keys Method | Remove Method | RemoveAll Method
Applies To: Dictionary Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Keys Method

Returns an array containing all existing keys in a **Dictionary** object.

```
object.Keys( )
```

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **Keys** method:

```
[JScript]
function KeysDemo()
{
   var a, d, i, s;                  // Create some variables.
   d = new ActiveXObject("Scripting.Dictionary");
   d.Add ("a", "Athens");              // Add some keys and items.
   d.Add ("b", "Belgrade");
   d.Add ("c", "Cairo");
   a = (new VBArray(d.Keys())).toArray();   // Get the keys.
   s = "";
   for (i in a)                  // Iterate the dictionary.
   {
      s += a[i] + " - " + d(a[i]) + "<br>";
   }
   return(s);                     // Return the results.
}
[VBScript]
```

```
Function DicDemo
   Dim a, d, i   ' Create some variables.
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "a", "Athens"   ' Add some keys and items.
   d.Add "b", "Belgrade"
   d.Add "c", "Cairo"
   a = d.Keys   ' Get the keys.
   For i = 0 To d.Count -1 ' Iterate the array.
      s = s & a(i) & "<BR>" ' Return results.
   Next
   DicDemo = s
End Function
```

See Also

Add Method (Dictionary) | Exists Method | Items Method | Remove Method | RemoveAll Method
Applies To: Dictionary Object

Scripting Runtime Library

Remove Method

Removes a key, item pair from a **Dictionary** object.

`object``.`**`Remove(`***`key`***`)`**

Arguments

object
    Required. Always the name of a **Dictionary** object.
key

Required. *Key* associated with the key, item pair you want to remove from the **Dictionary** object.

Remarks

An error occurs if the specified key, item pair does not exist.

The following code illustrates use of the **Remove** method:

```
[JScript]
var a, d, i, s;               // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens");        // Add some keys and items.
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
...
d.Remove("b");                // Remove second pair.
[VBScript]
Dim a, d   ' Create some variables.
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"    ' Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
d.Remove("b")   ' Remove second pair.
```

See Also

Add Method (Dictionary) | Exists Method | Items Method | Keys Method | RemoveAll Method
Applies To: Dictionary Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

RemoveAll Method

The **RemoveAll** method removes all key, item pairs from a **Dictionary** object.

```
object.RemoveAll( )
```

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **RemoveAll** method:

```
[JScript]
var a, d, i;             // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens");   // Add some keys and items.
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
...
d.RemoveAll( );       // Clear the dictionary.
[VBScript]
Dim a, d, i   ' Create some variables.
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"   ' Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
a = d.RemoveAll   ' Clear the dictionary.
```

See Also

Add Method (Dictionary) | Exists Method | Items Method | Keys Method | Remove Method
Applies To: Dictionary Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

FileSystemObject

In This Section

Scripting Run-Time Reference
List of elements that make up Scripting Run-Time Reference.
FileSystemObject Basics
A guide to the fundamentals of the FileSystemObject.

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

FileSystemObject Basics

When writing scripts for Active Server Pages, the Windows Script Host, or other applications where scripting can be used, it's often important to add, move, change, create, or delete folders (directories) and files on the Web server. It may also be necessary to get information about and manipulate drives attached to the Web server.

Scripting allows you to process drives, folders, and files using the **FileSystemObject** (FSO) object model, which is explained in the following sections:

The FileSystemObject Object Model

The FileSystemObject object model allows you to use the familiar *object.method* syntax with a rich set of properties, methods, and events to process folders and files.

FileSystemObject Objects

A list of the objects and collections in FileSystemObject object model.

Programming the FileSystemObject

Description of how to program with the FileSystemObject.

Working with Drives and Folders

Describes how you use the FileSystemObject to copy and move folders, as well as get information about drives and folders.

Working with Files

Describes how you use the FileSystemObject to manipulate files.

FileSystemObject Sample Code

A real-world example that demonstrates many of the features available in the FileSystemObject object model.

Scripting Runtime Library

The FileSystemObject Object Model

The **FileSystemObject** (FSO) object model allows you to use the familiar *object.method* syntax with a rich set of properties, methods, and events to process folders and files.

Use this object-based tool with:

- HTML to create Web pages
- Windows Scripting Host to create batch files for Microsoft Windows
- Script Control to provide a scripting capability to applications developed in other languages

Because use of the FSO on the client side raises serious security issues about providing potentially unwelcome access to a client's local file system, this documentation assumes use of the FSO object model to create scripts executed by Internet Web pages on the server side. Since the server side is used, the Internet Explorer default security settings do not allow client-side use of the **FileSystemObject** object. Overriding those defaults could subject a local computer to unwelcome access to the file system, which could result in total destruction of the file system's integrity, causing loss of data, or worse.

The FSO object model gives your server-side applications the ability to create, alter, move, and delete folders, or to detect if particular folders exist, and if so, where. You can also find out information about folders, such as their names, the date they were created or last modified, and so forth.

The FSO object model also makes it easy to process files. When processing files, the primary goal is to store data in a space- and resource-efficient, easy-to-access format. You need to be able to create files, insert and change the data, and output (read) the data. Since storing data in a database, such as Access or SQL Server, adds a significant amount of overhead to your application, storing your data in a binary or text file may be the most efficient solution. You may prefer not to have this overhead, or your data access requirements may not require all the extra features associated with a full-featured database.

- The FSO object model, which is contained in the Scripting type library (Scrrun.dll), supports text file creation and manipulation through the **TextStream** object. Although it does not yet support the creation or manipulation of binary files, future support of binary files is planned.

Scripting Runtime Library

FileSystemObject Objects

The **FileSystemObject** (FSO) object model contains the following objects and collections.

Object/Collection	Description
FileSystemObject	Main object. Contains methods and properties that allow you to create, delete, gain information about, and generally manipulate drives, folders, and files. Many of the methods associated with this object duplicate those in other FSO objects; they are provided for convenience.
Drive	Object. Contains methods and properties that allow you to gather information about a drive attached to the system, such as its share name and how much room is available. Note that a "drive" isn't necessarily a hard disk, but can be a CD-ROM drive, a RAM disk, and so forth. A drive doesn't need to be physically attached to the system; it can be also be logically connected through a network.
Drives	Collection. Provides a list of the drives attached to the system, either physically or logically. The **Drives** collection includes all drives, regardless of type. Removable-media drives need not have media inserted for them to appear in this collection.
File	Object. Contains methods and properties that allow you to create, delete, or move a file. Also allows you to query the system for a file name, path, and various other properties.
Files	Collection. Provides a list of all files contained within a folder.
Folder	Object. Contains methods and properties that allow you to create, delete, or move folders. Also allows you to query the system for folder names, paths, and various other properties.
Folders	Collection. Provides a list of all the folders within a **Folder**.
TextStream	Object. Allows you to read and write text files.

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Programming the FileSystemObject

To program with the **FileSystemObject** (FSO) object model:

- Use the **CreateObject** method to create a **FileSystemObject** object.
- Use the appropriate method on the newly created object.

- Access the object's properties.

The FSO object model is contained in the Scripting type library, which is located in the Scrrun.dll file. Therefore, you must have Scrrun.dll in the appropriate system directory on your Web server to use the FSO object model.

Creating a FileSystemObject Object

First, create a **FileSystemObject** object by using the **CreateObject** method.

The following code displays how to create an instance of the **FileSystemObject**:

```
[VBScript]
Dim fso
Set fso = CreateObject("Scripting.FileSystemObject")
[JScript]
var fso;
fso = new ActiveXObject("Scripting.FileSystemObject");
```

In both of these examples, **Scripting** is the name of the type library and **FileSystemObject** is the name of the object that you want to create. You can create only one instance of the **FileSystemObject** object, regardless of how many times you try to create another.

Using the Appropriate Method

Second, use the appropriate method of the **FileSystemObject** object. For example, to create a new object, use either **CreateTextFile** or **CreateFolder** (the FSO object model doesn't support the creation or deletion of drives).

To delete objects, use the **DeleteFile** and **DeleteFolder** methods of the **FileSystemObject** object, or the **Delete** method of the **File** and **Folder** objects. You can also copy and move files and folders, by using the appropriate methods.

> **Note** Some functionality in the **FileSystemObject** object model is redundant. For example, you can copy a file using either the **CopyFile** method of the **FileSystemObject** object, or you can use the **Copy** method of the **File** object. The methods work the same; both exist to offer programming flexibility.

Accessing Existing Drives, Files, and Folders

To gain access to an existing drive, file, or folder, use the appropriate "get" method of the **FileSystemObject** object:

- GetDrive
- GetFolder
- GetFile

To gain access to an existing file:

```
[VBScript]
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.GetFile("c:\test.txt")
[JScript]
var fso, f1;
fso = new ActiveXObject("Scripting.FileSystemObject");
f1 = fso.GetFile("c:\\test.txt");
```

Do not use the "get" methods for newly created objects, since the "create" functions already return a handle to that object. For example, if you create a new folder using the **CreateFolder** method, don't use the **GetFolder** method to access its properties, such as **Name**, **Path**, **Size**, and so forth. Just set a variable to the **CreateFolder** function to gain a handle to the newly created folder, then access its properties, methods, and events.

To set a variable to the **CreateFolder** function, use this syntax:

```
[VBScript]
Sub CreateFolder
   Dim fso, fldr
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set fldr = fso.CreateFolder("C:\MyTest")
   Response.Write "Created folder: " & fldr.Name
End Sub
[JScript]
function CreateFolder()
{
   var fso, fldr;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fldr = fso.CreateFolder("C:\\MyTest");
   Response.Write("Created folder: " + fldr.Name);
}
```

Accessing the Object's Properties

Once you have a handle to an object, you can access its properties. For example, to get the name of a particular folder, first create an instance of the object, then get a handle to it with the appropriate method (in this case, the **GetFolder** method, since the folder already exists).

Use this code to get a handle to the **GetFolder** method:

```
[VBScript]
Set fldr = fso.GetFolder("c:\")
[JScript]
var fldr = fso.GetFolder("c:\\");
```

Now that you have a handle to a **Folder** object, you can check its **Name** property.

```
[VBScript]
Response.Write "Folder name is: " & fldr.Name
[JScript]
Response.Write("Folder name is: " + fldr.Name);
```

To find out the last time a file was modified, use the following syntax:

```
[VBScript]
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
' Get a File object to query.
Set f1 = fso.GetFile("c:\detlog.txt")
' Print information.
Response.Write "File last modified: " & f1.DateLastModified
[JScript]
var fso, f1;
fso = new ActiveXObject("Scripting.FileSystemObject");
// Get a File object to query.
f1 = fso.GetFile("c:\\detlog.txt");
// Print information.
Response.Write("File last modified: " + f1.DateLastModified);
```

Scripting Runtime Library

Working with Drives and Folders

With the **FileSystemObject** (FSO) object model, you can work with drives and folders programmatically just as you can in the Windows Explorer interactively. You can copy and move folders, get information about drives and folders, and so forth.

Getting Information About Drives

The **Drive** object allows you to gain information about the various drives attached to a system, either physically or over a network. Its properties allow you to obtain information about:

- The total size of the drive in bytes (**TotalSize** property)
- How much space is available on the drive in bytes (**AvailableSpace** or **FreeSpace** properties)
- What letter is assigned to the drive (**DriveLetter** property)
- What type of drive it is, such as removable, fixed, network, CD-ROM, or RAM disk (**DriveType** property)
- The drive's serial number (**SerialNumber** property)
- The type of file system the drive uses, such as FAT, FAT32, NTFS, and so forth (**FileSystem** property)
- Whether a drive is available for use (**IsReady** property)
- The name of the share and/or volume (**ShareName** and **VolumeName** properties)
- The path or root folder of the drive (**Path** and **RootFolder** properties)

View the sample code to see how these properties are used in **FileSystemObject**.

Example Usage of the Drive Object

Use the **Drive** object to gather information about a drive. You won't see a reference to an actual **Drive** object in the following code; instead, use the **GetDrive** method to get a reference to an existing **Drive** object (in this case, drv).

The following example demonstrates how to use the **Drive** object:

```
[VBScript]
Sub ShowDriveInfo(drvPath)
   Dim fso, drv, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set drv = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & drv.VolumeName & "<br>"
   s = s & "Total Space: " & FormatNumber(drv.TotalSize / 1024, 0)
   s = s & " Kb" & "<br>"
   s = s & "Free Space: " & FormatNumber(drv.FreeSpace / 1024, 0)
   s = s & " Kb" & "<br>"
   Response.Write s
End Sub
[JScript]
function ShowDriveInfo1(drvPath)
{
   var fso, drv, s ="";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   drv = fso.GetDrive(fso.GetDriveName(drvPath));
   s += "Drive " + drvPath.toUpperCase()+ " - ";
   s += drv.VolumeName + "<br>";
   s += "Total Space: " + drv.TotalSize / 1024;
   s += " Kb" + "<br>";
   s += "Free Space: " + drv.FreeSpace / 1024;
   s += " Kb" + "<br>";
   Response.Write(s);
}
```

Working with Folders

Common folder tasks and the methods for performing them are described in the following table.

Task	Method
Create a folder.	FileSystemObject.CreateFolder
Delete a folder.	Folder.Delete or FileSystemObject.DeleteFolder
Move a folder.	Folder.Move or FileSystemObject.MoveFolder
Copy a folder.	Folder.Copy or FileSystemObject.CopyFolder
Retrieve the name of a folder.	Folder.Name
Find out if a folder exists on a drive.	FileSystemObject.FolderExists
Get an instance of an existing **Folder** object.	FileSystemObject.GetFolder
Find out the name of a folder's parent folder.	FileSystemObject.GetParentFolderName
Find out the path of system folders.	FileSystemObject.GetSpecialFolder

View the sample code to see how many of these methods and properties are used in **FileSystemObject**.

The following example demonstrates how to use the **Folder** and **FileSystemObject** objects to manipulate folders and gain information about them.

```
[VBScript]
Sub ShowFolderInfo()
   Dim fso, fldr, s
   ' Get instance of FileSystemObject.
   Set fso = CreateObject("Scripting.FileSystemObject")
   ' Get Drive object.
   Set fldr = fso.GetFolder("c:")
   ' Print parent folder name.
   Response.Write "Parent folder name is: " & fldr & "<br>"
   ' Print drive name.
   Response.Write "Contained on drive " & fldr.Drive & "<br>"
   ' Print root file name.
   If fldr.IsRootFolder = True Then
      Response.Write "This is the root folder." & ""<br>"<br>"
   Else
      Response.Write "This folder isn't a root folder." & "<br><br>"
   End If
   ' Create a new folder with the FileSystemObject object.
   fso.CreateFolder ("C:\Bogus")
   Response.Write "Created folder C:\Bogus" & "<br>"
   ' Print the base name of the folder.
   Response.Write "Basename = " & fso.GetBaseName("c:\bogus") & "<br>"
   ' Delete the newly created folder.
   fso.DeleteFolder ("C:\Bogus")
   Response.Write "Deleted folder C:\Bogus" & "<br>"
End Sub
[JScript]
function ShowFolderInfo()
{
   var fso, fldr, s = "";
   // Get instance of FileSystemObject.
   fso = new ActiveXObject("Scripting.FileSystemObject");
   // Get Drive object.
   fldr = fso.GetFolder("c:");
   // Print parent folder name.
   Response.Write("Parent folder name is: " + fldr + "<br>");
   // Print drive name.
   Response.Write("Contained on drive " + fldr.Drive + "<br>");
   // Print root file name.
   if (fldr.IsRootFolder)
```

```
      Response.Write("This is the root folder.");
    else
      Response.Write("This folder isn't a root folder.");
    Response.Write("<br><br>");
    // Create a new folder with the FileSystemObject object.
    fso.CreateFolder ("C:\\Bogus");
    Response.Write("Created folder C:\\Bogus" + "<br>");
    // Print the base name of the folder.
    Response.Write("Basename = " + fso.GetBaseName("c:\\bogus") + "<br>");
    // Delete the newly created folder.
    fso.DeleteFolder ("C:\\Bogus");
    Response.Write("Deleted folder C:\\Bogus" + "<br>");
}
```

Scripting Runtime Library

Working with Files

There are two major categories of file manipulation:

- Creating, adding, or removing data, and reading files
- Moving, copying, and deleting files

Creating Files

There are three ways to create an empty text file (sometimes referred to as a "text stream").

The first way is to use the **CreateTextFile** method. The following example demonstrates how to create a text file using the **CreateTextFileMethod** method.

```
[VBScript]
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
[JScript]
var fso, f1;
fso = new ActiveXObject("Scripting.FileSystemObject");
f1 = fso.CreateTextFile("c:\\testfile.txt", true);
```

The second way to create a text file is to use the **OpenTextFile** method of the **FileSystemObject** object with the **ForWriting** flag set.

```
[VBScript]
Dim fso, ts
Const ForWriting = 2
Set fso = CreateObject("Scripting. FileSystemObject")
Set ts = fso.OpenTextFile("c:\test.txt", ForWriting, True)
[JScript]
var fso, ts;
var ForWriting= 2;
fso = new ActiveXObject("Scripting.FileSystemObject");
ts = fso.OpenTextFile("c:\\test.txt", ForWriting, true);
```

A third way to create a text file is to use the **OpenAsTextStream** method with the **ForWriting** flag set.

```
[VBScript]
Dim fso, f1, ts
Const ForWriting = 2
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CreateTextFile ("c:\test1.txt")
Set f1 = fso.GetFile("c:\test1.txt")
Set ts = f1.OpenAsTextStream(ForWriting, True)
[JScript]
var fso, f1, ts;
var ForWriting = 2;
fso = new ActiveXObject("Scripting.FileSystemObject");
fso.CreateTextFile ("c:\\test1.txt");
f1 = fso.GetFile("c:\\test1.txt");
ts = f1.OpenAsTextStream(ForWriting, true);
```

Adding Data to the File

Once the text file is created, add data to the file using the following three steps:

Open the text file.

Write the data.

Close the file.

To open an existing file, use either the **OpenTextFile** method of the **FileSystemObject** object or the **OpenAsTextStream** method of the **File** object.

To write data to the open text file, use the **Write**, **WriteLine**, or **WriteBlankLines** methods of the **TextStream** object, according to the tasks outlined in the following table.

Task	**Method**
Write data to an open text file without a trailing newline character.	Write
Write data to an open text file with a trailing newline character.	WriteLine
Write one or more blank lines to an open text file.	WriteBlankLines

To close an open file, use the **Close** method of the **TextStream** object.

> **Note** The newline character contains a character or characters (depending on the operating system) to advance the cursor to the beginning of the next line (carriage return/line feed). Be aware that the end of some strings may already have such nonprinting characters.

The following example demonstrates how to open a file, use all three write methods to add data to the file, and then close the file:

```
[VBScript]
Sub CreateFile()
   Dim fso, tf
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set tf = fso.CreateTextFile("c:\testfile.txt", True)
   ' Write a line with a newline character.
   tf.WriteLine("Testing 1, 2, 3.")
   ' Write three newline characters to the file.
   tf.WriteBlankLines(3)
   ' Write a line.
   tf.Write ("This is a test.")
   tf.Close
End Sub
[JScript]
function CreateFile()
{
   var fso, tf;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   tf = fso.CreateTextFile("c:\\testfile.txt", true);
   // Write a line with a newline character.
   tf.WriteLine("Testing 1, 2, 3.") ;
   // Write three newline characters to the file.
   tf.WriteBlankLines(3) ;
   // Write a line.
   tf.Write ("This is a test.");
   tf.Close();
}
```

Reading Files

To read data from a text file, use the **Read**, **ReadLine**, or **ReadAll** method of the **TextStream** object. The following table describes which method to use for various tasks.

Task	Method
Read a specified number of characters from a file.	Read
Read an entire line (up to, but not including, the newline character).	ReadLine
Read the entire contents of a text file.	ReadAll

If you use the **Read** or **ReadLine** method and want to skip to a particular portion of data, use the **Skip** or **SkipLine** method. The resulting text of the read methods is stored in a string which can be displayed in a control, parsed by string functions (such as **Left**, **Right**, and **Mid**), concatenated, and so forth.

The following example demonstrates how to open a file, write to it, and then read from it:

```
[VBScript]
Sub ReadFiles
   Dim fso, f1, ts, s
   Const ForReading = 1
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
   ' Write a line.
   Response.Write "Writing file <br>"
   f1.WriteLine "Hello World"
   f1.WriteBlankLines(1)
   f1.Close
   ' Read the contents of the file.
   Response.Write "Reading file <br>"
   Set ts = fso.OpenTextFile("c:\testfile.txt", ForReading)
   s = ts.ReadLine
   Response.Write "File contents = '" & s & "'"
   ts.Close
End Sub
[JScript]
function ReadFiles()
{
   var fso, f1, ts, s;
   var ForReading = 1;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f1 = fso.CreateTextFile("c:\\testfile.txt", true);
   // Write a line.
   Response.Write("Writing file <br>");
   f1.WriteLine("Hello World");
   f1.WriteBlankLines(1);
   f1.Close();
   // Read the contents of the file.
   Response.Write("Reading file <br>");
   ts = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   s = ts.ReadLine();
   Response.Write("File contents = '" + s + "'");
   ts.Close();
}
```

Moving, Copying, and Deleting Files

The FSO object model has two methods each for moving, copying, and deleting files, as described in the following table.

Task	Method
Move a file	File.Move or FileSystemObject.MoveFile
Copy a file	File.Copy or FileSystemObject.CopyFile
Delete a file	File.Delete or FileSystemObject.DeleteFile

The following example creates a text file in the root directory of drive C, writes some information to it, moves it to a directory called \tmp, makes a copy of it in a directory called \temp, then deletes the copies from both directories.

To run the following example, create directories named \tmp and \temp in the root directory of drive C:

```
[VBScript]
Sub ManipFiles
   Dim fso, f1, f2, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
   Response.Write "Writing file <br>"
   ' Write a line.
   f1.Write ("This is a test.")
   ' Close the file to writing.
   f1.Close
   Response.Write "Moving file to c:\tmp <br>"
   ' Get a handle to the file in root of C:\.
   Set f2 = fso.GetFile("c:\testfile.txt")
   ' Move the file to \tmp directory.
   f2.Move ("c:\tmp\testfile.txt")
   Response.Write "Copying file to c:\temp <br>"
   ' Copy the file to \temp.
   f2.Copy ("c:\temp\testfile.txt")
   Response.Write "Deleting files <br>"
   ' Get handles to files' current location.
   Set f2 = fso.GetFile("c:\tmp\testfile.txt")
   Set f3 = fso.GetFile("c:\temp\testfile.txt")
   ' Delete the files.
   f2.Delete
   f3.Delete
   Response.Write "All done!"
End Sub
[JScript]
function ManipFiles()
{
   var fso, f1, f2, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f1 = fso.CreateTextFile("c:\\testfile.txt", true);
   Response.Write("Writing file <br>");
   // Write a line.
   f1.Write("This is a test.");
   // Close the file to writing.
   f1.Close();
   Response.Write("Moving file to c:\\tmp <br>");
   // Get a handle to the file in root of C:\.
   f2 = fso.GetFile("c:\\testfile.txt");
   // Move the file to \tmp directory.
   f2.Move ("c:\\tmp\\testfile.txt");
   Response.Write("Copying file to c:\\temp <br>");
   // Copy the file to \temp.
   f2.Copy ("c:\\temp\\testfile.txt");
   Response.Write("Deleting files <br>");
   // Get handles to files' current location.
   f2 = fso.GetFile("c:\\tmp\\testfile.txt");
   f3 = fso.GetFile("c:\\temp\\testfile.txt");
   // Delete the files.
   f2.Delete();
   f3.Delete();
   Response.Write("All done!");
}
```

Scripting Runtime Library

FileSystemObject Sample Code

The sample code described in this section provides a real-world example that demonstrates many of the features available in the **FileSystemObject** object model. This code shows how all the features of the object model work together, and how to use those features effectively in your own code.

Notice that since this code is fairly generic, some additional code and a little tweaking are needed to make this code actually run on your machine. These changes are necessary because of the different ways input and output to the user is handled between Active Server Pages and the Windows Scripting Host.

To run this code on an Active Server Page, use the following steps:

Create a standard web page with an .asp extension.

Copy the following sample code into that file between the <BODY>...</BODY> tags.

Enclose all the code within <%...%> tags.

Move the **Option Explicit** statement from its current position in the code to the very top of your HTML page, positioning it even before the opening <HTML> tag.

Place <%...%> tags around the **Option Explicit** statement to ensure that it's run on the server side.

Add the following code to the end of the sample code:

```
Sub Print(x)
   Response.Write "<PRE>&ltFONT FACE=""Courier New"" SIZE=""1"">"
   Response.Write x
   Response.Write "</FONT></PRE>"
End Sub
Main
```

The previous code adds a print procedure that will run on the server side, but display results on the client side. To run this code on the Windows Scripting Host, add the following code to the end of the sample code:

```
Sub Print(x)
   WScript.Echo x
End Sub
Main
```

The code is contained in the following section:

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' FileSystemObject Sample Code
' Copyright 1998 Microsoft Corporation.   All Rights Reserved.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Regarding code quality:
' 1) The following code does a lot of string manipulation by
'    concatenating short strings together with the "&" operator.
'    Since string concatenation is expensive, this is a very
'    inefficient way to write code. However, it is a very
'    maintainable way to write code, and is used here because this
'    program performs extensive disk operations, and because the
'    disk is much slower than the memory operations required to
'    concatenate the strings. Keep in mind that this is demonstration
'    code, not production code.
'
' 2) "Option Explicit" is used, because declared variable access is
'    slightly faster than undeclared variable access. It also prevents
'    bugs from creeping into your code, such as when you misspell
'    DriveTypeCDROM as DriveTypeCDORM.
'
' 3) Error handling is absent from this code, to make the code more
'    readable. Although precautions have been taken to ensure that the
'    code will not error in common cases, file systems can be
'    unpredictable. In production code, use On Error Resume Next and
'    the Err object to trap possible errors.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Some handy global variables
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim TabStop
Dim NewLine

Const TestDrive = "C"
Const TestFilePath = "C:\Test"

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Constants returned by Drive.DriveType
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const DriveTypeRemovable = 1
Const DriveTypeFixed = 2
Const DriveTypeNetwork = 3
Const DriveTypeCDROM = 4
Const DriveTypeRAMDisk = 5

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Constants returned by File.Attributes
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const FileAttrNormal   = 0
Const FileAttrReadOnly = 1
Const FileAttrHidden = 2
Const FileAttrSystem = 4
Const FileAttrVolume = 8
Const FileAttrDirectory = 16
Const FileAttrArchive = 32
Const FileAttrAlias = 64
Const FileAttrCompressed = 128

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Constants for opening files
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const OpenFileForReading = 1
Const OpenFileForWriting = 2
Const OpenFileForAppending = 8

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ShowDriveType
' Purpose:
'    Generates a string describing the drive type of a given Drive object.
' Demonstrates the following
'  - Drive.DriveType
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function ShowDriveType(Drive)

   Dim S

   Select Case Drive.DriveType
   Case DriveTypeRemovable
      S = "Removable"
   Case DriveTypeFixed
      S = "Fixed"
   Case DriveTypeNetwork
      S = "Network"
   Case DriveTypeCDROM
      S = "CD-ROM"
   Case DriveTypeRAMDisk
      S = "RAM Disk"
   Case Else
      S = "Unknown"
   End Select

   ShowDriveType = S

End Function
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ShowFileAttr
' Purpose:
'    Generates a string describing the attributes of a file or folder.
' Demonstrates the following
'  - File.Attributes
'  - Folder.Attributes
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function ShowFileAttr(File) ' File can be a file or folder

   Dim S
   Dim Attr

   Attr = File.Attributes

   If Attr = 0 Then
      ShowFileAttr = "Normal"
      Exit Function
   End If

   If Attr And FileAttrDirectory Then S = S & "Directory "
   If Attr And FileAttrReadOnly Then S = S & "Read-Only "
   If Attr And FileAttrHidden Then S = S & "Hidden "
   If Attr And FileAttrSystem Then S = S & "System "
   If Attr And FileAttrVolume Then S = S & "Volume "
   If Attr And FileAttrArchive Then S = S & "Archive "
   If Attr And FileAttrAlias Then S = S & "Alias "
   If Attr And FileAttrCompressed Then S = S & "Compressed "

   ShowFileAttr = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GenerateDriveInformation
' Purpose:
'    Generates a string describing the current state of the
'    available drives.
' Demonstrates the following
'  - FileSystemObject.Drives
'  - Iterating the Drives collection
'  - Drives.Count
'  - Drive.AvailableSpace
'  - Drive.DriveLetter
'  - Drive.DriveType
'  - Drive.FileSystem
'  - Drive.FreeSpace
'  - Drive.IsReady
'  - Drive.Path
'  - Drive.SerialNumber
'  - Drive.ShareName
'  - Drive.TotalSize
'  - Drive.VolumeName
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateDriveInformation(FSO)

   Dim Drives
   Dim Drive
   Dim S

   Set Drives = FSO.Drives
   S = "Number of drives:" & TabStop & Drives.Count & NewLine & NewLine

   ' Construct 1st line of report.
   S = S & String(2, TabStop) & "Drive"
   S = S & String(3, TabStop) & "File"
   S = S & TabStop & "Total"
   S = S & TabStop & "Free"
   S = S & TabStop & "Available"
   S = S & TabStop & "Serial" & NewLine

   ' Construct 2nd line of report.
   S = S & "Letter"
   S = S & TabStop & "Path"
   S = S & TabStop & "Type"
   S = S & TabStop & "Ready?"
   S = S & TabStop & "Name"
   S = S & TabStop & "System"
   S = S & TabStop & "Space"
   S = S & TabStop & "Space"
   S = S & TabStop & "Space"
   S = S & TabStop & "Number" & NewLine

   ' Separator line.
   S = S & String(105, "-") & NewLine
   For Each Drive In Drives
      S = S & Drive.DriveLetter
      S = S & TabStop & Drive.Path
      S = S & TabStop & ShowDriveType(Drive)
      S = S & TabStop & Drive.IsReady

      If Drive.IsReady Then
         If DriveTypeNetwork = Drive.DriveType Then
            S = S & TabStop & Drive.ShareName
         Else
            S = S & TabStop & Drive.VolumeName
         End If
         S = S & TabStop & Drive.FileSystem
         S = S & TabStop & Drive.TotalSize
         S = S & TabStop & Drive.FreeSpace
         S = S & TabStop & Drive.AvailableSpace
         S = S & TabStop & Hex(Drive.SerialNumber)
      End If

      S = S & NewLine

   Next

   GenerateDriveInformation = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GenerateFileInformation
' Purpose:
'    Generates a string describing the current state of a file.
' Demonstrates the following
'  - File.Path
'  - File.Name
'  - File.Type
'  - File.DateCreated
'  - File.DateLastAccessed
'  - File.DateLastModified
'  - File.Size
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateFileInformation(File)
```

```
   Dim S

   S = NewLine & "Path:" & TabStop & File.Path
   S = S & NewLine & "Name:" & TabStop & File.Name
   S = S & NewLine & "Type:" & TabStop & File.Type
   S = S & NewLine & "Attribs:" & TabStop & ShowFileAttr(File)
   S = S & NewLine & "Created:" & TabStop & File.DateCreated
   S = S & NewLine & "Accessed:" & TabStop & File.DateLastAccessed
   S = S & NewLine & "Modified:" & TabStop & File.DateLastModified
   S = S & NewLine & "Size" & TabStop & File.Size & NewLine

   GenerateFileInformation = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GenerateFolderInformation
' Purpose:
'    Generates a string describing the current state of a folder.
' Demonstrates the following
'  - Folder.Path
'  - Folder.Name
'  - Folder.DateCreated
'  - Folder.DateLastAccessed
'  - Folder.DateLastModified
'  - Folder.Size
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateFolderInformation(Folder)

   Dim S

   S = "Path:" & TabStop & Folder.Path
   S = S & NewLine & "Name:" & TabStop & Folder.Name
   S = S & NewLine & "Attribs:" & TabStop & ShowFileAttr(Folder)
   S = S & NewLine & "Created:" & TabStop & Folder.DateCreated
   S = S & NewLine & "Accessed:" & TabStop & Folder.DateLastAccessed
   S = S & NewLine & "Modified:" & TabStop & Folder.DateLastModified
   S = S & NewLine & "Size:" & TabStop & Folder.Size & NewLine

   GenerateFolderInformation = S

End Function
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GenerateAllFolderInformation
' Purpose:
'    Generates a string describing the current state of a
'    folder and all files and subfolders.
' Demonstrates the following
'  - Folder.Path
'  - Folder.SubFolders
'  - Folders.Count
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateAllFolderInformation(Folder)

   Dim S
   Dim SubFolders
   Dim SubFolder
   Dim Files
   Dim File

   S = "Folder:" & TabStop & Folder.Path & NewLine & NewLine
   Set Files = Folder.Files

   If 1 = Files.Count Then
      S = S & "There is 1 file" & NewLine
   Else
      S = S & "There are " & Files.Count & " files" & NewLine
   End If

   If Files.Count <> 0 Then
      For Each File In Files
         S = S & GenerateFileInformation(File)
      Next
   End If

   Set SubFolders = Folder.SubFolders

   If 1 = SubFolders.Count Then
      S = S & NewLine & "There is 1 sub folder" & NewLine & NewLine
   Else
      S = S & NewLine & "There are " & SubFolders.Count & " sub folders" _
      NewLine & NewLine
   End If

   If SubFolders.Count <> 0 Then
```

```
      For Each SubFolder In SubFolders
         S = S & GenerateFolderInformation(SubFolder)
      Next
      S = S & NewLine
      For Each SubFolder In SubFolders
         S = S & GenerateAllFolderInformation(SubFolder)
      Next
   End If

   GenerateAllFolderInformation = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GenerateTestInformation
' Purpose:
'    Generates a string describing the current state of the C:\Test
'    folder and all files and subfolders.
' Demonstrates the following
'  - FileSystemObject.DriveExists
'  - FileSystemObject.FolderExists
'  - FileSystemObject.GetFolder
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateTestInformation(FSO)

   Dim TestFolder
   Dim S

   If Not FSO.DriveExists(TestDrive) Then Exit Function
   If Not FSO.FolderExists(TestFilePath) Then Exit Function

   Set TestFolder = FSO.GetFolder(TestFilePath)

   GenerateTestInformation = GenerateAllFolderInformation(TestFolder)

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' DeleteTestDirectory
' Purpose:
'    Cleans up the test directory.
' Demonstrates the following
'  - FileSystemObject.GetFolder
'  - FileSystemObject.DeleteFile
'  - FileSystemObject.DeleteFolder
'  - Folder.Delete
'  - File.Delete
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sub DeleteTestDirectory(FSO)

   Dim TestFolder
   Dim SubFolder
   Dim File


   ' Two ways to delete a file:

   FSO.DeleteFile(TestFilePath & "\Beatles\OctopusGarden.txt")

   Set File = FSO.GetFile(TestFilePath & "\Beatles\BathroomWindow.txt")
   File.Delete

   ' Two ways to delete a folder:
   FSO.DeleteFolder(TestFilePath & "\Beatles")
   FSO.DeleteFile(TestFilePath & "\ReadMe.txt")
   Set TestFolder = FSO.GetFolder(TestFilePath)
   TestFolder.Delete

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' CreateLyrics
' Purpose:
'    Builds a couple of text files in a folder.
' Demonstrates the following
'  - FileSystemObject.CreateTextFile
'  - TextStream.WriteLine
'  - TextStream.Write
'  - TextStream.WriteBlankLines
'  - TextStream.Close
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sub CreateLyrics(Folder)

   Dim TextStream
   Set TextStream = Folder.CreateTextFile("OctopusGarden.txt")


   ' Note that this does not add a line feed to the file.
   TextStream.Write("Octopus' Garden ")
   TextStream.WriteLine("(by Ringo Starr)")
   TextStream.WriteBlankLines(1)
   TextStream.WriteLine("I'd like to be under the sea in an octopus' garden in the shade,")
   TextStream.WriteLine("He'd let us in, knows where we've been -- in his octopus' garden in the shade.")
   TextStream.WriteBlankLines(2)

   TextStream.Close

   Set TextStream = Folder.CreateTextFile("BathroomWindow.txt")
   TextStream.WriteLine("She Came In Through The Bathroom Window (by Lennon/McCartney)")
   TextStream.WriteLine("")
   TextStream.WriteLine("She came in through the bathroom window protected by a silver spoon")
   TextStream.WriteLine("But now she sucks her thumb and wanders by the banks of her own lagoon")
   TextStream.WriteBlankLines(2)
   TextStream.Close

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GetLyrics
' Purpose:
'    Displays the contents of the lyrics files.
' Demonstrates the following
'  - FileSystemObject.OpenTextFile
'  - FileSystemObject.GetFile
'  - TextStream.ReadAll
'  - TextStream.Close
'  - File.OpenAsTextStream
'  - TextStream.AtEndOfStream
'  - TextStream.ReadLine
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GetLyrics(FSO)

   Dim TextStream
   Dim S
   Dim File
   ' There are several ways to open a text file, and several
   ' ways to read the data out of a file. Here's two ways
   ' to do each:

   Set TextStream = FSO.OpenTextFile(TestFilePath & "\Beatles\OctopusGarden.txt", OpenFileForReading)


   S = TextStream.ReadAll & NewLine & NewLine
   TextStream.Close

   Set File = FSO.GetFile(TestFilePath & "\Beatles\BathroomWindow.txt")
   Set TextStream = File.OpenAsTextStream(OpenFileForReading)
   Do    While Not TextStream.AtEndOfStream
      S = S & TextStream.ReadLine & NewLine
   Loop
   TextStream.Close

   GetLyrics = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' BuildTestDirectory
' Purpose:
'    Builds a directory hierarchy to demonstrate the FileSystemObject.
'    We'll build a hierarchy in this order:
'       C:\Test
'       C:\Test\ReadMe.txt
'       C:\Test\Beatles
'       C:\Test\Beatles\OctopusGarden.txt
'       C:\Test\Beatles\BathroomWindow.txt
' Demonstrates the following
'  - FileSystemObject.DriveExists
'  - FileSystemObject.FolderExists
'  - FileSystemObject.CreateFolder
'  - FileSystemObject.CreateTextFile
'  - Folders.Add
'  - Folder.CreateTextFile
'  - TextStream.WriteLine
'  - TextStream.Close
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function BuildTestDirectory(FSO)
```

```
   Dim TestFolder
   Dim SubFolders
   Dim SubFolder
   Dim TextStream

   ' Bail out if (a) the drive does not exist, or if (b) the directory is being built
   ' already exists.

   If Not FSO.DriveExists(TestDrive) Then
      BuildTestDirectory = False
      Exit Function
   End If

   If FSO.FolderExists(TestFilePath) Then
      BuildTestDirectory = False
      Exit Function
   End If

   Set TestFolder = FSO.CreateFolder(TestFilePath)

   Set TextStream = FSO.CreateTextFile(TestFilePath & "\ReadMe.txt")
   TextStream.WriteLine("My song lyrics collection")
   TextStream.Close

   Set SubFolders = TestFolder.SubFolders
   Set SubFolder = SubFolders.Add("Beatles")
   CreateLyrics SubFolder
   BuildTestDirectory = True

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' The main routine
' First, it creates a test directory, along with some subfolders
' and files. Then, it dumps some information about the available
' disk drives and about the test directory, and then cleans
' everything up again.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sub Main

   Dim FSO

   ' Set up global data.
   TabStop = Chr(9)
   NewLine = Chr(10)


   Set FSO = CreateObject("Scripting.FileSystemObject")

   If Not BuildTestDirectory(FSO) Then
      Print "Test directory already exists or cannot be created.   Cannot continue."
      Exit Sub
   End If

   Print GenerateDriveInformation(FSO) & NewLine & NewLine
   Print GenerateTestInformation(FSO) & NewLine & NewLine
   Print GetLyrics(FSO) & NewLine & NewLine
   DeleteTestDirectory(FSO)

End Sub
```

Scripting Runtime Library

Scripting Run-Time Reference

In This Section

FileSystemObject Properties
  List of properties you can use with the FileSystemObject object model.
FileSystemObject Methods
  List of methods you can use with the FileSystemObject object model.
FileSystemObject Objects
  List of objects you can use with the FileSystemObject object model.
FileSystemObject Collections

List of collections you can use with the FileSystemObject object model.

Related Section

FileSystemObject Basics
A guide to the fundamentals of the FileSystemObject.

Scripting Runtime Library

FileSystemObject Properties

In This Section

AtEndOfLine Property
Returns true if the file pointer is positioned immediately before the end-of-line marker in a **TextStream** file; false if it is not.
AtEndOfStream Property
Returns true if the file pointer is at the end of a **TextStream** file; false if it is not.
Attributes Property
Sets or returns the attributes of files or folders.
AvailableSpace Property
Returns the amount of space available to a user on the specified drive or network share.
Column Property
Returns the column number of the current character position in a **TextStream** file.
CompareMode Property
Sets and returns the comparison mode for comparing string keys in a **Dictionary** object.
Count Property
Returns the number of items in a collection or **Dictionary** object.
DateCreated Property
Returns the date and time that the specified file or folder was created. Read-only.

DateLastAccessed Property
: Returns the date and time that the specified file or folder was last accessed.

DateLastModified Property
: Returns the date and time that the specified file or folder was last modified.

Drive Property
: Returns the drive letter of the drive on which the specified file or folder resides.

DriveLetter Property
: Returns the drive letter of a physical local drive or a network share.

Drives Property
: Returns a **Drives** collection consisting of all **Drive** objects available on the local machine.

DriveType Property
: Returns a value indicating the type of a specified drive.

Files Property
: Returns a **Files** collection consisting of all **File** objects contained in the specified folder, including those with hidden and system file attributes set.

FileSystemProperty
: Returns the type of file system in use for the specified drive.

FreeSpace Property
: Returns the amount of free space available to a user on the specified drive or network share.

IsReady Property
: Returns true if the specified drive is ready; false if it is not.

IsRootFolder Property
: Returns true if the specified folder is the root folder; false if it is not.

Item Property
: Sets or returns an *item* for a specified *key* in a **Dictionary** object. For collections, returns an *item* based on the specified *key*.

Key Property
: Sets a *key* in a **Dictionary** object.

Line Property
: Returns the current line number in a **TextStream** file.

Name Property
: Sets or returns the name of a specified file or folder.

ParentFolder Property
: Returns the folder object for the parent of the specified file or folder.

Path Property
: Returns the path for a specified file, folder, or drive.

RootFolder Property
: Returns a **Folder** object representing the root folder of a specified drive.

SerialNumber Property
: Returns the decimal serial number used to uniquely identify a disk volume.

ShareName Property
: Returns the network share name for a specified drive.

ShortName Property
: Returns the short name used by programs that require the earlier 8.3 naming convention.

ShortPath Property
: Returns the short path used by programs that require the earlier 8.3 file naming convention.

Size Property
: For files, returns the size, in bytes, of the specified file. For folders, returns the size, in bytes, of all files and subfolders contained in the folder.

SubFolders Property
: Returns a **Folders** collection consisting of all folders contained in a specified folder, including those with hidden and system file attributes set.

TotalSize Property
: Returns the total space, in bytes, of a drive or network share.

Type Property
: Returns information about the type of a file or folder.

VolumeName Property
: Sets or returns the volume name of the specified drive.

Related Sections

Scripting Run-Time Reference
: List of elements that make up Scripting Run-Time Reference.

FileSystemObject Basics
: A guide to the fundamentals of the FileSystemObject.

Scripting Runtime Library

AtEndOfLine Property

Returns **true** if the file pointer is positioned immediately before the end-of-line marker in a **TextStream** file; **false** if it is not. Read-only.

```
object.AtEndOfLine
```

The *object* is always the name of a **TextStream** object.

Remarks

The **AtEndOfLine** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfLine** property:

```
[JScript]
function GetALine(filespec)
{
   var fso, a, s, ForReading;
   ForReading = 1, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   a = fso.OpenTextFile(filespec, ForReading, false);
   while (!a.AtEndOfLine)
   {
      s += a.Read(1);
   }
   a.Close( );
   return(s);
}
[VBScript]
Function ReadEntireFile(filespec)
   Const ForReading = 1
   Dim fso, theFile, retstring
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set theFile = fso.OpenTextFile(filespec, ForReading, False)
   Do While theFile.AtEndOfLine <> True
      retstring = theFile.Read(1)
   Loop
   theFile.Close
   ReadEntireFile = retstring
End Function
```

See Also

AtEndOfStream Property

Applies To: TextStream Object

Scripting Runtime Library

AtEndOfStream Property

Returns **true** if the file pointer is at the end of a **TextStream** file; **false** if it is not. Read-only.

`*object*.AtEndOfStream`

The *object* is always the name of a **TextStream** object.

Remarks

The **AtEndOfStream** property applies only to **TextStream** files that are open for reading, otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfStream** property:

```
[JScript]
function GetALine(filespec)
{
   var fso, f, s, ForReading;
   ForReading = 1, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.OpenTextFile(filespec, ForReading, false);
   while (!f.AtEndOfStream)
```

```
      s += f.ReadLine( );
   f.Close( );
   return(s);
}
[VBScript]
Function ReadEntireFile(filespec)
   Const ForReading = 1
   Dim fso, theFile, retstring
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set theFile = fso.OpenTextFile(filespec, ForReading, False)
   Do While theFile.AtEndOfStream <> True
      retstring = theFile.ReadLine
   Loop
   theFile.Close
   ReadEntireFile = retstring
End Function
```

See Also

AtEndOfLine Property

Applies To: TextStream Object

Scripting Runtime Library

Attributes Property

Sets or returns the attributes of files or folders. Read/write or read-only, depending on the attribute.

```
object.Attributes [= newattributes]
```

Arguments

object
Required. Always the name of a **File** or **Folder** object.

newattributes
Optional. If provided, *newattributes* is the new value for the attributes of the specified *object*.

Settings

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	64	Link or shortcut. Attribute is read-only.
Compressed	128	Compressed file. Attribute is read-only.

Remarks

The following code illustrates the use of the **Attributes** property with a file:

```
[JScript]
function ToggleArchiveBit(filespec)
{
   var fso, f, r, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec)
   if (f.attributes && 32)
   {
      f.attributes = f.attributes - 32;
      s = "Archive bit is cleared.";
   }
   else
   {
      f.attributes = f.attributes + 32;
      s =   "Archive bit is set.";
   }
   return(s);
}
[VBScript]
Function ToggleArchiveBit(filespec)
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   If f.attributes and 32 Then
      f.attributes = f.attributes - 32
      ToggleArchiveBit = "Archive bit is cleared."
   Else
      f.attributes = f.attributes + 32
      ToggleArchiveBit = "Archive bit is set."
   End If
End Function
```

See Also

DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

AvailableSpace Property

Returns the amount of space available to a user on the specified drive or network share.

```
object.AvailableSpace
```

The *object* is always a **Drive** object.

Remarks

The 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.

The following code illustrates the use of the **AvailableSpace** property:

```
[JScript]
function ShowAvailableSpace(drvPath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + drvPath.toUpperCase() + " - ";
   s += d.VolumeName + "<br>";
   s += "Available Space: " + d.AvailableSpace/1024 + " Kbytes";
   return(s);
}
[VBScript]
Function ShowAvailableSpace(drvPath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & d.VolumeName   & "<BR>"
   s = s & "Available Space: " & FormatNumber(d.AvailableSpace/1024, 0)
   s = s & " Kbytes"
   ShowAvailableSpace = s
End Function
```

See Also

DriveLetter Property | DriveType Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Column Property

Read-only property that returns the column number of the current character position in a **TextStream** file.

`object.Column`

The *object* is always the name of a **TextStream** object.

Remarks

After a newline character has been written, but before any other character is written, **Column** is equal to 1.

The following examples illustrates the use of the **Column** property:

```
[JScript]
function GetColumn()
{
   var fso, f, m;
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true);
   f.Write("Hello World!");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   m = f.ReadLine();
   return(f.Column);
}
```

```
[VBScript]
Function GetColumn
   Const ForReading = 1, ForWriting = 2
   Dim fso, f, m
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.Write "Hello world!"
   f.Close
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   m =   f.ReadLine
   GetColumn = f.Column
End Function
```

See Also

Line Property

Applies To: TextStream Object

Scripting Runtime Library

CompareMode Property

Sets and returns the comparison mode for comparing string keys in a **Dictionary** object.

object.**CompareMode**[= *compare*]

Arguments

object

Required. Always the name of a **Dictionary** object.

compare

Optional. If provided, *compare* is a value representing the comparison mode. Acceptable values are 0 (Binary), 1 (Text), 2 (Database). Values greater than 2 can be used to refer to comparisons using specific Locale IDs (LCID).

Remarks

An error occurs if you try to change the comparison mode of a **Dictionary** object that already contains data.

The following example illustrates the use of the **CompareMode** property:

```
[JScript]function TestCompareMode(key)
{
   // Create some variables.
   var a, d;
   var BinaryCompare = 0, TextCompare = 1;
   d = new ActiveXObject("Scripting.Dictionary");
   // Set Compare mode to Text.
   d.CompareMode = TextCompare;
   // Add some keys and items.
   d.Add("a", "Athens");
   d.Add("b", "Belgrade");
   d.Add("c", "Cairo");
   return(d.Item(key));
}
[VBScript
Dim d
Set d = CreateObject("Scripting.Dictionary")

d.CompareMode = vbTextCompare
d.Add "a", "Athens"    ' Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
d.Add "B", "Baltimore"    ' Add method fails on this line because the
                          ' letter b already exists in the Dictionary.
```

See Also

Key Property

Applies To: Dictionary Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

DateLastAccessed Property

Returns the date and time that the specified file or folder was last accessed. Read-only.

object.DateLastAccessed

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateLastAccessed** property with a file:

```
[JScript]
function ShowFileAccessInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = filespec.toUpperCase() + "<br>";
   s += "Created: " + f.DateCreated + "<br>";
   s += "Last Accessed: " + f.DateLastAccessed + "<br>";
   s += "Last Modified: " + f.DateLastModified;
   return(s);
}
[VBScript]
Function ShowFileAccessInfo(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = UCase(filespec) & "<BR>"
   s = s & "Created: " & f.DateCreated & "<BR>"
   s = s & "Last Accessed: " & f.DateLastAccessed & "<BR>"
   s = s & "Last Modified: " & f.DateLastModified
   ShowFileAccessInfo = s
End Function
```

> **Note** This method depends on the underlying operating system for its behavior. If the operating system does not support providing time information, none will be returned.

See Also

Attributes Property | DateCreated Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

DateLastModified Property

Returns the date and time that the specified file or folder was last modified. Read-only.

object`.DateLastModified`

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateLastModified** property with a file:

```
[JScript]
function ShowFileAccessInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = filespec.toUpperCase() + "<br>";
   s += "Created: " + f.DateCreated + "<br>";
   s += "Last Accessed: " + f.DateLastAccessed + "<br>";
   s += "Last Modified: " + f.DateLastModified;
   return(s);
}
[VBScript]
Function ShowFileAccessInfo(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = UCase(filespec) & "<BR>"
   s = s & "Created: " & f.DateCreated & "<BR>"
   s = s & "Last Accessed: " & f.DateLastAccessed & "<BR>"
   s = s & "Last Modified: " & f.DateLastModified
   ShowFileAccessInfo = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Drive Property

Returns the drive letter of the drive on which the specified file or folder resides. Read-only.

object.**Drive**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Drive** property:

```
[JScript]
function ShowFileAccessInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = f.Name + " on Drive " + f.Drive + "<br>";
   s += "Created: " + f.DateCreated + "<br>";
   s += "Last Accessed: " + f.DateLastAccessed + "<br>";
   s += "Last Modified: " + f.DateLastModified;
   return(s);
}
[VBScript]
Function ShowFileAccessInfo(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = f.Name & " on Drive " & UCase(f.Drive) & "<BR>"
   s = s & "Created: " & f.DateCreated & "<BR>"
   s = s & "Last Accessed: " & f.DateLastAccessed & "<BR>"
   s = s & "Last Modified: " & f.DateLastModified
   ShowFileAccessInfo = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Scripting Runtime Library

DriveLetter Property

Returns the drive letter of a physical local drive or a network share. Read-only.

```
object.DriveLetter
```

The *object* is always a **Drive** object.

Remarks

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.

The following code illustrates the use of the **DriveLetter** property:

```
[JScript]
function ShowDriveLetter(drvPath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + d.DriveLetter.toUpperCase( ) + ": - ";
   s += d.VolumeName + "<br>";
   s += "Available Space: " + d.AvailableSpace/1024 + " Kbytes";
   return(s);
}
[VBScript]
Function ShowDriveLetter(drvPath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & d.DriveLetter & ": - "
   s = s & d.VolumeName & "<BR>"
   s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
   s = s & " Kbytes"
   ShowDriveLetter = s
End Function
```

See Also

AvailableSpace Property | DriveType Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Scripting Runtime Library

Drives Property

Returns a **Drives** collection consisting of all **Drive** objects available on the local machine.

object.**Drives**

The *object* is always a **FileSystemObject**.

Remarks

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

[JScript]

You can iterate the members of the **Drives** collection using the **Enumerator** object and the **for** statement:

```
[JScript]
function ShowDriveList()
{
   var fso, s, n, e, x;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   e = new Enumerator(fso.Drives);
   s = "";
   for (; !e.atEnd(); e.moveNext())
   {
      x = e.item();
      s = s + x.DriveLetter;
      s += " - ";
      if (x.DriveType == 3)
         n = x.ShareName;
      else if (x.IsReady)
         n = x.VolumeName;
      else
         n = "[Drive not ready]";
      s +=   n + "<br>";
   }
   return(s);
}
```

[VBScript]

You can iterate the members of the **Drives** collection using a **For Each...Next** construct as illustrated in the following code:

```
[VBScript]
```

```
Function ShowDriveList
   Dim fso, d, dc, s, n
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set dc = fso.Drives
   For Each d in dc
      n = ""
      s = s & d.DriveLetter & " - "
      If d.DriveType = 3 Then
         n = d.ShareName
      ElseIf d.IsReady Then
         n = d.VolumeName
      End If
      s = s & n & "<BR>"
   Next
   ShowDriveList = s
End Function
```

See Also

Drives Collection | Files Property | SubFolders Property

Applies To: FileSystemObject Object

Scripting Runtime Library

DriveType Property

Returns a value indicating the type of a specified drive.

```
object.DriveType
```

The *object* is always a **Drive** object.

Remarks

The following code illustrates the use of the **DriveType** property:

```
[JScript]
function ShowDriveType(drvpath)
{
   var fso, d, s, t;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(drvpath);
   switch (d.DriveType)
   {
      case 0: t = "Unknown"; break;
      case 1: t = "Removable"; break;
      case 2: t = "Fixed"; break;
      case 3: t = "Network"; break;
      case 4: t = "CD-ROM"; break;
      case 5: t = "RAM Disk"; break;
   }
   s = "Drive " + d.DriveLetter + ": - " + t;
   return(s);
}
[VBScript]
Function ShowDriveType(drvpath)
   Dim fso, d, t
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(drvpath)
   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
   ShowDriveType = "Drive " & d.DriveLetter & ": - " & t
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Scripting Runtime Library

FileSystem Property

Returns the type of file system in use for the specified drive.

```
object.FileSystem
```

The *object* is always a **Drive** object.

Remarks

Available return types include FAT, NTFS, and CDFS.

The following code illustrates the use of the **FileSystem** property:

```
[JScript]
function ShowFileSystemType(drvPath)
{
   var fso,d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(drvPath);
   s = d.FileSystem;
   return(s);
}
```

```
[VBScript]
Function ShowFileSystemType(drvspec)
   Dim fso,d
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(drvspec)
   ShowFileSystemType = d.FileSystem
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | DriveType Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Scripting Runtime Library

FreeSpace Property

Returns the amount of free space available to a user on the specified drive or network share. Read-only.

```
object.FreeSpace
```

The *object* is always a **Drive** object.

Remarks

The 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.

The following code illustrates the use of the **FreeSpace** property:

```
[JScript]
function ShowFreeSpace(drvPath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + drvPath.toUpperCase( ) + " - ";
   s += d.VolumeName + "<br>";
   s += "Free Space: " + d.FreeSpace/1024 + " Kbytes";
   return(s);
}
[VBScript]
Function ShowFreeSpace(drvPath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & d.VolumeName   & "<BR>"
   s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
   s = s & " Kbytes"
   ShowFreeSpace = s
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | DriveType Property | FileSystem Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

IsReady Property

Returns **True** if the specified drive is ready; **False** if it is not.

object.**IsReady**

The *object* is always a **Drive** object.

Remarks

For removable-media drives and CD-ROM drives, **IsReady** returns **True** only when the appropriate media is inserted and ready for access.

The following code illustrates the use of the **IsReady** property:

```
[JScript]
function ShowDriveInfo(drvpath)
{
   var fso, d, s, t;
   fso = new ActiveXObject("Scripting.FileSystemObject")
   d = fso.GetDrive(drvpath)
   switch (d.DriveType)
   {
      case 0: t = "Unknown"; break;
      case 1: t = "Removable"; break;
      case 2: t = "Fixed"; break;
      case 3: t = "Network"; break;
      case 4: t = "CD-ROM"; break;
      case 5: t = "RAM Disk"; break;
   }
   s = "Drive " + d.DriveLetter + ": - " + t;
   if (d.IsReady)
      s += "<br>" + "Drive is Ready.";
   else
      s += "<br>" + "Drive is not Ready.";
   return(s);
}
[VBScript]
Function ShowDriveInfo(drvpath)
   Dim fso, d, s, t
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(drvpath)
   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
   s = "Drive " & d.DriveLetter & ": - " & t
   If d.IsReady Then
      s = s & "<BR>" & "Drive is Ready."
   Else
      s = s & "<BR>" & "Drive is not Ready."
   End If
   ShowDriveInfo = s
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | DriveType Property | FileSystem Property | FreeSpace Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Scripting Runtime Library

IsRootFolder Property

Returns **True** if the specified folder is the root folder; **False** if it is not.

```
object.IsRootFolder
```

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **IsRootFolder** property:

```
[JScript]
function DisplayLevelDepth(pathspec)
{
   var fso, f, n, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(pathspec);
   n = 0;
   if (f.IsRootFolder)
      s = "The specified folder is the root folder."
   else
   {
      do
      {
         f = f.ParentFolder;
         n++;
      }
      while (!f.IsRootFolder)
      s = "The specified folder is nested " + n + " levels deep."
   }
   return(s);
}
[VBScript]
Function DisplayLevelDepth(pathspec)
   Dim fso, f, n
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(pathspec)
   If f.IsRootFolder Then
      DisplayLevelDepth = "The specified folder is the root folder."
   Else
      Do Until f.IsRootFolder
         Set f = f.ParentFolder
         n = n + 1
      Loop
      DisplayLevelDepth = "The specified folder is nested " & n & " levels deep."
   End If
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: Folder Object

Scripting Runtime Library

Line Property

Read-only property that returns the current line number in a **TextStream** file.

```
object.Line
```

The *object* is always the name of a **TextStream** object.

Remarks

After a file is initially opened and before anything is written, **Line** is equal to 1.

The following example illustrates the use of the **Line** property:

```
[JScript]
function GetLine()
```

```
{
   var fso, f, r
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject")
   f = fso.OpenTextFile("c:\\textfile.txt", ForWriting, true)
   f.WriteLine("Hello world!");
   f.WriteLine("JScript is fun");
   f.Close();
   f = fso.OpenTextFile("c:\\textfile.txt", ForReading);
   r =   f.ReadAll();
   return(f.Line);
}
[VBScript]
Function GetLine
   Const ForReading = 1, ForWriting = 2
   Dim fso, f, ra
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.Write "Hello world!" & vbCrLf & "VB Script is fun!" & vbCrLf
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   ra =   f.ReadAll
   GetLine = f.Line
End Function
```

See Also

Column Property

Applies To: TextStream Object

Name Property

Sets or returns the name of a specified file or folder. Read/write.

```
object.Name [= newname]
```

Arguments

object
Required. Always the name of a **File** or **Folder** object.

newname
Optional. If provided, *newname* is the new name of the specified *object*.

Remarks

The following code illustrates the use of the **Name** property:

```
[JScript]
function ShowFileAccessInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = f.Name + " on Drive " + f.Drive + "<br>";
   s += "Created: " + f.DateCreated + "<br>";
   s += "Last Accessed: " + f.DateLastAccessed + "<br>";
   s += "Last Modified: " + f.DateLastModified;
   return(s);
}
[VBScript]
Function ShowFileAccessInfo(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = f.Name & " on Drive " & UCase(f.Drive) & "<BR>"
   s = s & "Created: " & f.DateCreated & "<BR>"
   s = s & "Last Accessed: " & f.DateLastAccessed & "<BR>"
   s = s & "Last Modified: " & f.DateLastModified
   ShowFileAccessInfo = s
```

```
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Scripting Runtime Library

ParentFolder Property

Returns the folder object for the parent of the specified file or folder. Read-only.

object.ParentFolder

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ParentFolder** property with a file:

```
[JScript]
function ShowFileAccessInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = f.Name + " in " + f.ParentFolder + "<br>";
   s += "Created: " + f.DateCreated + "<br>";
   s += "Last Accessed: " + f.DateLastAccessed + "<br>";
   s += "Last Modified: " + f.DateLastModified;
   return(s);
}
[VBScript]
Function ShowFileAccessInfo(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = UCase(f.Name) & " in " & UCase(f.ParentFolder) & "<BR>"
   s = s & "Created: " & f.DateCreated & "<BR>"
   s = s & "Last Accessed: " & f.DateLastAccessed & "<BR>"
   s = s & "Last Modified: " & f.DateLastModified
   ShowFileAccessInfo = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Scripting Runtime Library

Path Property

Returns the path for a specified file, folder, or drive.

object.**Path**

The *object* is always a **File**, **Folder**, or **Drive** object.

Remarks

For drive letters, the root drive is not included. For example, the path for the C drive is C:, not C:\.

The following code illustrates the use of the **Path** property with a **File** object:

```
[JScript]
function ShowFileAccessInfo(filespec)
{
   var fso, d, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = f.Path.toUpperCase() + "<br>";
   s += "Created: " + f.DateCreated + "<br>";
   s += "Last Accessed: " + f.DateLastAccessed + "<br>";
   s += "Last Modified: " + f.DateLastModified
   return(s);
}
[VBScript]
Function ShowFileAccessInfo(filespec)
   Dim fso, d, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = UCase(f.Path) & "<BR>"
   s = s & "Created: " & f.DateCreated & "<BR>"
   s = s & "Last Accessed: " & f.DateLastAccessed & "<BR>"
   s = s & "Last Modified: " & f.DateLastModified
   ShowFileAccessInfo = s
End Function
```

See Also

Attributes Property | AvailableSpace Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | DriveLetter Property | DriveType Property | Files Property | FileSystem Property | FreeSpace Property | IsReady Property | IsRootFolder Property | Name Property | ParentFolder Property | RootFolder Property | SerialNumber Property | ShareName Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | TotalSize Property | Type Property | VolumeName Property

Applies To: Drive Object | File Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

RootFolder Property

Returns a **Folder** object representing the root folder of a specified drive. Read-only.

object.RootFolder

The *object* is always a **Drive** object.

Remarks

All the files and folders contained on the drive can be accessed using the returned **Folder** object.

The following example illustrates the use of the **RootFolder** property:

```
[JScript]
function GetRootFolder(drv)
{
   var fso,d;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   if (fso.DriveExists(drv))
      {
         d = fso.GetDrive(drv);
         return(d.RootFolder);
      }
   else
      return(false);
}
```

```
[VBScript]
Function ShowRootFolder(drvspec)
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetDrive(drvspec)
   ShowRootFolder = f.RootFolder
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | DriveType Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

SerialNumber Property

Returns the decimal serial number used to uniquely identify a disk volume.

object.**SerialNumber**

The *object* is always a **Drive** object.

Remarks

You can use the **SerialNumber** property to ensure that the correct disk is inserted in a drive with removable media.

The following code illustrates the use of the **SerialNumber** property:

```
[JScript]
function ShowDriveInfo(drvpath){
  var fso, d, s, t;
  fso = new ActiveXObject("Scripting.FileSystemObject");
  d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)));
  switch (d.DriveType)
  {
    case 0: t = "Unknown"; break;
    case 1: t = "Removable"; break;
    case 2: t = "Fixed"; break;
    case 3: t = "Network"; break;
    case 4: t = "CD-ROM"; break;
    case 5: t = "RAM Disk"; break;
  }
  s = "Drive " + d.DriveLetter + ": - " + t;
  s += "<br>" + "SN: " + d.SerialNumber;
  return(s);
}
[VBScript]
Function ShowDriveInfo(drvpath)
   Dim fso, d, s, t
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
   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
   s = "Drive " & d.DriveLetter & ": - " & t
   s = s & "<BR>" & "SN: " & d.SerialNumber
   ShowDriveInfo = s
End Function
```

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

ShareName Property

Returns the network share name for a specified drive.

object.ShareName

The *object* is always a **Drive** object.

Remarks

If *object* is not a network drive, the **ShareName** property returns a zero-length string ("").

The following code illustrates the use of the **ShareName** property:

```
[JScript]
function ShowDriveInfo(drvpath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)));
   s = "Drive " + d.DriveLetter + ": - " + d.ShareName;
   return(s);
}
[VBScript]
Function ShowDriveInfo(drvpath)
   Dim fso, d
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
   ShowDriveInfo = "Drive " & d.DriveLetter & ": - " & d.ShareName
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | DriveType Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | TotalSize Property | VolumeName Property

Applies To: Drive Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

ShortName Property

Returns the short name used by programs that require the earlier 8.3 naming convention.

`object.ShortName`

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ShortName** property with a **File** object:

```
[JScript]
function ShowShortName(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = "The short name for " + "" + f.Name;
   s += "" + "<br>";
   s += "is: " + "" + f.ShortName + "";
   return(s);
}
[VBScript]
Function ShowShortName(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = "The short name for "   & UCase(f.Name) & "<BR>"
   s = s & "is: " & f.ShortName
   ShowShortName = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

ShortPath Property

Returns the short path used by programs that require the earlier 8.3 file naming convention.

```
object.ShortPath
```

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ShortName** property with a **File** object:

```
[JScript]
function ShowShortPath(filespec)
```

```
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = "The short path for " + "" + f.Name;
   s += "" + "<br>";
   s += "is: " + "" + f.ShortPath + "";
   return(s);
}
[VBScript]
Function ShowShortName(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = "The short name for "   & UCase(f.Name) & "<BR>"
   s = s & "is: " & f.ShortName
   ShowShortName = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Scripting Runtime Library

Size Property

For files, returns the size, in bytes, of the specified file. For folders, returns the size, in bytes, of all files and subfolders contained in the folder.

object.**Size**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Size** property with a **Folder** object:

```
[JScript]
function ShowFolderSize(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(filespec);
   s = f.Name + " uses " + f.size + " bytes.";
   return(s);
}
[VBScript]
 Function ShowFolderSize(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(filespec)
   s = UCase(f.Name) & " uses " & f.size & " bytes."
   ShowFolderSize = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

SubFolders Property

Returns a **Folders** collection consisting of all folders contained in a specified folder, including those with hidden and system file attributes set.

object.SubFolders

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **SubFolders** property:

```
[JScript]
function ShowFolderList(folderspec)
{
   var fso, f, fc, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(folderspec);
   fc = new Enumerator(f.SubFolders);
   s = "";
   for (;!fc.atEnd(); fc.moveNext())
      {
         s += fc.item();
         s += "<br>";
      }
      return(s);
}
[VBScript]
Function ShowFolderList(folderspec)
   Dim fso, f, f1, s, sf
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(folderspec)
   Set sf = f.SubFolders
   For Each f1 in sf
      s = s & f1.name
      s = s & "<BR>"
   Next
   ShowFolderList = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | Type Property

Applies To: Folder Object

Scripting Runtime Library

TotalSize Property

Returns the total space, in bytes, of a drive or network share.

`object.TotalSize`

The *object* is always a **Drive** object.

Remarks

The following code illustrates the use of the **TotalSize** property:

```
[JScript]
function SpaceReport(drvPath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + drvPath + " - ";
   s += d.VolumeName + "<br>";
   s += "Total Space: "+ d.TotalSize/1024 + " Kbytes <br>";
   s += "Free Space:   " + d.FreeSpace/1024 + " Kbytes";
   return(s);
}
[VBScript]
Function ShowSpaceInfo(drvpath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
   s = "Drive " & d.DriveLetter & ":"
   s = s & vbCrLf
   s = s & "Total Size: " & FormatNumber(d.TotalSize/1024, 0) & " Kbytes"
   s = s & vbCrLf
   s = s & "Available: " & FormatNumber(d.AvailableSpace/1024, 0) & " Kbytes"
   ShowSpaceInfo = s
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | DriveType Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | VolumeName Property

Applies To: Drive Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Type Property

Returns information about the type of a file or folder. For example, for files ending in .TXT, "Text Document" is returned.

object.**Type**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Type** property to return a folder type. In this example, try providing the path of the Recycle Bin or other unique folder to the procedure.

```
[JScript]
function ShowFileType(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   if (fso.FolderExists(filespec))
      f = fso.GetFolder(filespec);
   else if (fso.FileExists(filespec))
      f = fso.GetFile(filespec);
   else
      s = "File or Folder does not exist.";
   s = f.Name + " is a " + f.Type;
   return(s);
}
[VBScript]
Function ShowFolderType(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(filespec)
   s = UCase(f.Name) & " is a " & f.Type
   ShowFolderType = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property |

IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property

Applies To: File Object | Folder Object

Scripting Runtime Library

VolumeName Property

Sets or returns the volume name of the specified drive. Read/write.

object.**VolumeName** [= *newname*]

Arguments

object
: Required. Always the name of a **Drive** object.

newname
: Optional. If provided, *newname* is the new name of the specified *object*.

Remarks

The following code illustrates the use of the **VolumeName** property:

```
[JScript]
function SpaceReport(drvPath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + drvPath + " - ";
   s += d.VolumeName + "<br>";
   s += "Total Space: "+ d.TotalSize/1024 + " Kbytes <br>";
   s += "Free Space:   " + d.FreeSpace/1024 + " Kbytes";
   return(s);
}
[VBScript]
Function ShowVolumeInfo(drvpath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
   s = "Drive " & d.DriveLetter & ": - " & d.VolumeName
   ShowVolumeInfo = s
End Function
```

See Also

AvailableSpace Property | DriveLetter Property | DriveType Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property

Applies To: Drive Object

Scripting Runtime Library

FileSystemObject Methods

In This Section

Add Method (Dictionary)
    Adds a key and item pair to a **Dictionary** object.

Add Method (Folders)
Adds a new folder to a **Folders** collection.
BuildPath Method
Appends a name to an existing path.
Close Method
Closes an open **TextStream** file.
Copy Method
Copies a specified file or folder from one location to another.
CopyFile Method
Copies one or more files from one location to another.
CopyFolder Method
Recursively copies a folder from one location to another.
CreateFolder Method
Creates a folder.
CreateTextFile Method
Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.
Delete Method
Deletes a specified file or folder.
DeleteFile Method
Deletes a specified file.
DeleteFolder Method
Deletes a specified folder and its contents.
DrivesExists Method
Returns true if the specified drive exists; false if it does not.
Exists Method
Returns true if a specified key exists in the **Dictionary** object, false if it does not.
FileExists Method
Returns true if a specified file exists; false if it does not.
FolderExists Method
Returns true if a specified folder exists; false if it does not.
GetAbsolutePathName Method
Returns a complete and unambiguous path from a provided path specification.
GetBaseName Method
Returns a string containing the base name of the last component, less any file extension, in a path.
GetDrive Method
Returns a **Drive** object corresponding to the drive in a specified path.
GetDriveName Method

Returns a string containing the name of the drive for a specified path.

GetExtensionName Method

Returns a string containing the extension name for the last component in a path.

GetFile Method

Returns a **File** object corresponding to the file in a specified path.

GetFileName Method

Returns the last component of specified path that is not part of the drive specification.

GetFileVersion Method

Returns the version number of a specified file.

GetFolder Method

Returns a **Folder** object corresponding to the folder in a specified path.

GetParentFolderName Method

Returns a string containing the name of the parent folder of the last component in a specified path.

GetSpecialFolder Method

Returns the special folder object specified.

GetTempName Method

Returns a randomly generated temporary file or folder name that is useful for performing operations that require a temporary file or folder.

Items Method

Returns an array containing all the items in a **Dictionary** object.

Keys Method

Returns an array containing all existing keys in a **Dictionary** object.

Move Method

Moves a specified file or folder from one location to another.

MoveFile Method

Moves one or more files from one location to another.

MoveFolder Method

Moves one or more folders from one location to another.

OpenAsTextStream Method

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

OpenTextFile Method

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

Read Method

Reads a specified number of characters from a **TextStream** file and returns the resulting string.

ReadAll Method

Reads an entire **TextStream** file and returns the resulting string.

ReadLine Method

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

Remove Method
  Removes a key, item pair from a **Dictionary** object.

RemoveAll Method
  Removes all key, item pairs from a **Dictionary** object.

Skip Method
  Skips a specified number of characters when reading a **TextStream** file.

SkipLine
  Skips the next line when reading a **TextStream** file.

Write Method
  Writes a specified string to a **TextStream** file.

WriteBlankLines Method
  Writes a specified number of newline characters to a **TextStream** file.

WriteLine Method
  Writes a specified string and newline character to a **TextStream** file.

Related Sections

Scripting Run-Time Reference
  List of elements that make up Scripting Run-Time Reference.

FileSystemObject Basics
  A guide to the fundamentals of the FileSystemObject.

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Add Method (Folders)

Adds a new folder to a **Folders** collection.

`object.`**`Add (`**`folderName`**`)`**

Arguments

object
Required. Always the name of a **Folders** collection.

folderName
Required. The name of the new **Folder** being added.

Remarks

The following example illustrates the use of the **Add** method to create a new folder.

```
[JScript]
function AddNewFolder(path,folderName)
{
   var fso, f, fc, nf;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(path);
   fc = f.SubFolders;
   if (folderName != "" )
      nf = fc.Add(folderName);
   else
      nf = fc.Add("New Folder");
}
[VBScript]
Sub AddNewFolder(path, folderName)
   Dim fso, f, fc, nf
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(path)
   Set fc = f.SubFolders
   If folderName <> "" Then
      Set nf = fc.Add(folderName)
   Else
      Set nf = fc.Add("New Folder")
   End If
End Sub
```

An error occurs if the *folderName* already exists.

See Also

Add Method (Dictionary)
Applies To: Folders Collection

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

BuildPath Method

Appends a name to an existing path.

object.**BuildPath(***path*, *name***)**

Arguments

object
Required. Always the name of a **FileSystemObject**.

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*.

Remarks

The **BuildPath** method inserts an additional path separator between the existing path and the name, only if necessary.

The following example illustrates use of the **BuildPath** method.

```
[JScript]
function GetBuildPath(path)
{
   var fso, newpath;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   newpath = fso.BuildPath(path, "New   Folder");
   return(newpath);
}
[VBScript]
Function GetBuildPath(path)
   Dim fso, newpath
   Set fso = CreateObject("Scripting.FileSystemObject")
   newpath = fso.BuildPath(path, "Sub Folder")
   GetBuildPath = newpath
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDriveName Method | GetExtensionName Method | GetFileName Method | GetParentFolderName Method | GetTempName Method
Applies To: FileSystemObject Object

Scripting Runtime Library

Close Method

Closes an open **TextStream** file.

`object.Close( );`

The *object* is always the name of a **TextStream** object.

Remarks

The following example illustrates use of the **Close** method.

```
[JScript]
var fso;
fso = new ActiveXObject("Scripting.FileSystemObject");
a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
[VBScript]
Sub CreateAFile
   Dim fso, MyFile
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
   MyFile.WriteLine("This is a test.")
   MyFile.Close
End Sub
```

Scripting Runtime Library

Copy Method

Copies a specified file or folder from one location to another.

```
object.Copy( destination[, overwrite] );
```

Arguments

object
Required. Always the name of a **File** or **Folder** object.
destination
Required. Destination where the file or folder is to be copied. Wildcard characters are not allowed.

overwrite
Optional. Boolean value that is **True** (default) if existing files or folders are to be overwritten; **False** if they are not.

Remarks

The results of the **Copy** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.CopyFile** or **FileSystemObject.CopyFolder** where the file or folder referred to by *object* is passed as an argument. You should note, however, that the alternative methods are capable of copying multiple files or folders.

Example

The following example illustrates the use of the **Copy** method.

```
[JScript]
var fso, f;
fso = new ActiveXObject("Scripting.FileSystemObject");
f = fso.CreateTextFile("c:\\testfile.txt", true);
f.WriteLine("This is a test.");
f.Close();
f = fso.GetFile("c:\\testfile.txt");
f.Copy("c:\\windows\\desktop\\test2.txt");
[VBScript]
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
MyFile.WriteLine("This is a test.")
MyFile.Close
Set MyFile = fso.GetFile("c:\testfile.txt")
MyFile.Copy ("c:\windows\desktop\test2.txt")
```

See Also

CopyFile Method | CopyFolder Method | Delete Method | Move Method | OpenAsTextStream Method
Applies To: File Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

CopyFile Method

Copies one or more files from one location to another.

```
object.CopyFile ( source, destination[, overwrite] )
```

Arguments

object
: Required. The *object* is always the name of a **FileSystemObject**.

source
: Required. 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.

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.*

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
[JScript]
fso = new ActiveXObject("Scripting.FileSystemObject");
fso.CopyFile ("c:\\mydocuments\\letters\\*.doc", "c:\\tempfolder\\")
[VBScript]
FileSystemObject.CopyFile "c:\mydocuments\letters\*.doc", "c:\tempfolder\"
```

But you cannot use:

```
[JScript]
fso = new ActiveXObject("Scripting.FileSystemObject");
fso.CopyFile ("c:\\mydocuments\\*\\R1???97.xls", "c:\\tempfolder")
```

```
[VBScript]
FileSystemObject.CopyFile "c:\mydocuments\*\R1???97.xls", "c:\tempfolder"
```

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 an individual 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.

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.

See Also

Copy Method | CopyFolder Method | CreateTextFile Method | DeleteFile Method | MoveFile Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

CopyFolder Method

Recursively copies a folder from one location to another.

```
object.CopyFolder ( source, destination[, overwrite] );
```

Arguments

object
: Required. Always the name of a **FileSystemObject**.

source
: Required. Character string folder specification, which can include wildcard characters, for one or more folders to be copied.

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

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**.

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
[JScript]
fso = new ActiveXObject("Scripting.FileSystemObject");
fso.CopyFolder ("c:\\mydocuments\\letters\\*", "c:\\tempfolder\\")
[VBScript]
FileSystemObject.CopyFolder "c:\mydocuments\letters\*", "c:\tempfolder\"
```

But you cannot use:

```
[JScript]
fso = new ActiveXObject("Scripting.FileSystemObject");
fso.CopyFolder ("c:\\mydocuments\\*\\*", "c:\\tempfolder\\")
[VBScript]
FileSystemObject.CopyFolder "c:\mydocuments\*\*", "c:\tempfolder\"
```

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 any changes made before an error occurs.

See Also

CopyFile Method | Copy Method | CreateFolder Method | DeleteFolder Method | MoveFolder Method
Applies To: FileSystemObject Object

Scripting Runtime Library

CreateFolder Method

Creates a folder.

```
object.CreateFolder(foldername)
```

Arguments

object
Required. Always the name of a **FileSystemObject**.
foldername
Required. String expression that identifies the folder to create.

Remarks

An error occurs if the specified folder already exists.

The following code illustrates how to use the **CreateFolder** method to create a folder.

```
[JScript]
var fso = new ActiveXObject("Scripting.FileSystemObject");
var a = fso.CreateFolder("c:\\new folder");
[VBScript]
Function CreateFolderDemo
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.CreateFolder("c:\New Folder")
   CreateFolderDemo = f.Path
End Function
```

See Also

CopyFolder Method | DeleteFolder Method | MoveFolder Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

CreateTextFile Method

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

Arguments

object
    Required. Always the name of a **FileSystemObject** or **Folder** object.
filename
    Required. 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.
unicode
    Optional. Boolean value that indicates whether the file is created as a Unicode or ASCII file. The value is **true** if the file is created as a Unicode file, **false** if it's created as an ASCII file. If omitted, an ASCII file is assumed.

Remarks

The following code illustrates how to use the **CreateTextFile** method to create and open a text file.

```
[JScript]
var fso = new ActiveXObject("Scripting.FileSystemObject");
var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
[VBScript]
Sub CreateAfile
   Dim fso, MyFile
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
   MyFile.WriteLine("This is a test.")
   MyFile.Close
End Sub
```

If the *overwrite* argument is **false**, or is not provided, for a *filename* that already exists, an error occurs.

See Also

CreateFolder Method | OpenAsTextStream Method | OpenTextFile Method
Applies To: FileSystemObject Object | Folder Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

DateCreated Property

Returns the date and time that the specified file or folder was created. Read-only.

```
object.DateCreated
```

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateCreated** property with a file:

```
[JScript]
function ShowFileInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = "Created: " + f.DateCreated;
   return(s);
}
[VBScript]
Function ShowFileInfo(filespec)
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   ShowFileInfo = "Created: " & f.DateCreated
End Function
```

See Also

Attributes Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name

Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: File Object | Folder Object

Scripting Runtime Library

Delete Method

Deletes a specified file or folder.

```
object.Delete( force );
```

Arguments

object
Required. Always the name of a **File** or **Folder** object.

force
Optional. Boolean value that is **True** if files or folders with the read-only attribute set are to be deleted; **False** (default) if they are not.

Remarks

An error occurs if the specified file or folder does not exist.

The results of the **Delete** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.DeleteFile** or **FileSystemObject.DeleteFolder**.

The **Delete** 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.

The following example illustrates the use of the **Delete** method.

```
[JScript]
var fso, f;
fso = new ActiveXObject("Scripting.FileSystemObject");
f = fso.CreateTextFile("c:\\testfile.txt", true);
f.WriteLine("This is a test.");
f.Close();
f = fso.GetFile("c:\\testfile.txt");
f.Delete();
[VBScript]
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
MyFile.WriteLine("This is a test.")
MyFile.Close
Set MyFile = fso.GetFile("c:\testfile.txt")
MyFile.Delete
```

See Also

Copy Method | DeleteFile Method | DeleteFolder Method | Move Method | OpenAsTextStream Method
Applies To: File Object | Folder Object

Scripting Runtime Library

DeleteFile Method

Deletes a specified file.

object.**DeleteFile (** *filespec*[, *force*] **);**

Arguments

object
: Required. Always the name of a **FileSystemObject**.

filespec
: Required. The name of the file to delete. The *filespec* can contain wildcard characters in the last path component.

force
: Optional. Boolean value that is **true** if files with the read-only attribute set are to be deleted; **false** (default) if they are not.

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.

The following example illustrates the use of the **DeleteFile** method.

```
[JScript]
function DeleteFile(filespec)
{
   var fso;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fso.DeleteFile(filespec);
}
[VBScript]
Sub DeleteAFile(filespec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   fso.DeleteFile(filespec)
End Sub
```

See Also

CopyFile Method | CreateTextFile Method | Delete Method | DeleteFolder Method | MoveFile Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

DeleteFolder Method

Deletes a specified folder and its contents.

object.**DeleteFolder (** *folderspec*[, *force*] **);**

Arguments

object
: Required. Always the name of a **FileSystemObject**.

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** if folders with the read-only attribute set are to be deleted; **false** (default) if they are not.

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.

The following example illustrates the use of the **DeleteFolder** method.

```
[JScript]
function DeleteFolder(folderspec)
{
   var fso;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fso.DeleteFolder(folderspec);
}
[VBScript]
```

```
Sub DeleteAFolder(filespec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   fso.DeleteFolder(filespec)
End Sub
```

See Also

CopyFolder Method | CreateFolder Method | Delete Method | DeleteFile Method | MoveFolder Method
Applies To: FileSystemObject Object

Scripting Runtime Library

DriveExists Method

Returns **True** if the specified drive exists; **False** if it does not.

object.**DriveExists(***drivespec***)**

Arguments

object
: Required. Always the name of a **FileSystemObject**.

drivespec
: Required. A drive letter or a complete path specification.

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.

The following example illustrates the use of the **DriveExists** method.

```
[JScript]
function ReportDriveStatus(drv)
{
   var fso, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   if (fso.DriveExists(drv))
      s += "Drive " + drv + " exists.";
   else
      s += "Drive " + drv + " doesn't exist.";
   return(s);
}
[VBScript]
Function ReportDriveStatus(drv)
   Dim fso, msg
   Set fso = CreateObject("Scripting.FileSystemObject")
   If fso.DriveExists(drv) Then
      msg = ("Drive " & UCase(drv) & " exists.")
   Else
      msg = ("Drive " & UCase(drv) & " doesn't exist.")
   End If
   ReportDriveStatus = msg
End Function
```

See Also

Drive Object | Drives Collection | FileExists Method | FolderExists Method | GetDrive Method | GetDriveName Method | IsReady Property
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

FileExists Method

Returns **True** if a specified file exists; **False** if it does not.

object.**FileExists(***filespec***)**

Arguments

object
: Required. Always the name of a **FileSystemObject**.

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.

The following example illustrates the use of the **FileExists** method.

```
[JScript]
function ReportFileStatus(filespec)
{
   var fso, s = filespec;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   if (fso.FileExists(filespec))
      s += " exists.";
   else
      s += " doesn't exist.";
   return(s);
}
[VBScript]
Function ReportFileStatus(filespec)
   Dim fso, msg
   Set fso = CreateObject("Scripting.FileSystemObject")
   If (fso.FileExists(filespec)) Then
      msg = filespec & " exists."
   Else
      msg = filespec & " doesn't exist."
   End If
   ReportFileStatus = msg
End Function
```

See Also

DriveExists Method | FolderExists Method | GetFile Method | GetFileName Method
Applies To: FileSystemObject Object

Scripting Runtime Library

Files Property

Returns a **Files** collection consisting of all **File** objects contained in the specified folder, including those with hidden and system file attributes set.

object.**Files**

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **Files** property:

```
[JScript]
function ShowFolderFileList(folderspec)
{
   var fso, f, fc, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(folderspec);
   fc = new Enumerator(f.files);
   s = "";
   for (; !fc.atEnd(); fc.moveNext())
   {
      s += fc.item();
      s += "<br>";
   }
   return(s);
}
[VBScript]Function ShowFileList(folderspec)
   Dim fso, f, f1, fc, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(folderspec)
   Set fc = f.Files
   For Each f1 in fc
      s = s & f1.name
      s = s &   "<BR>"
   Next
   ShowFileList = s
End Function
```

See Also

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

Applies To: Folder Object

Scripting Runtime Library

FolderExists Method

Returns **True** if a specified folder exists; **False** if it does not.

object.**FolderExists(***folderspec***)**

Arguments

object
: Required. Always the name of a **FileSystemObject**.

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.

The following example illustrates the use of the **FileExists** method.

```
[JScript]
function ReportFolderStatus(fldr)
{
   var fso, s = fldr;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   if (fso.FolderExists(fldr))
      s += " exists.";
   else
      s += " doesn't exist.";
   return(s);
}
[VBScript]
Function ReportFolderStatus(fldr)
   Dim fso, msg
   Set fso = CreateObject("Scripting.FileSystemObject")
   If (fso.FolderExists(fldr)) Then
      msg = fldr & " exists."
   Else
      msg = fldr & " doesn't exist."
   End If
   ReportFolderStatus = msg
End Function
```

See Also

DriveExists Method | FileExists Method | GetFolder Method | GetParentFolderName Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

GetAbsolutePathName Method

Returns a complete and unambiguous path from a provided path specification.

```
object.GetAbsolutePathName(pathspec)
```

Arguments

object
Required. Always the name of a **FileSystemObject**.

pathspec
Required. Path specification to change to a complete and unambiguous path.

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"

The following example illustrates the use of the **GetAbsolutePathName** method.

```
function ShowAbsolutePath(path)
{
   var fso, s= "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   s += fso.GetAbsolutePathName(path);
   return(s);
}
```

See Also

GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Scripting Runtime Library

GetBaseName Method

Returns a string containing the base name of the last component, less any file extension, in a path.

```
object.GetBaseName(path)
```

Arguments

object
Required. Always the name of a **FileSystemObject**.

path
Required. The path specification for the component whose base name is to be returned.

Remarks

The **GetBaseName** method returns a zero-length string ("") if no component matches the *path* argument.

> **Note** 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 following example illustrates the use of the **GetBaseName** method.

```
[JScript]
function ShowBaseName(filespec)
{
   var fso, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   s += fso.GetBaseName(filespec);
   return(s);
}
[VBScript]
Function GetTheBase(filespec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   GetTheBase = fso.GetBaseName(filespec)
End Function
```

See Also

GetAbsolutePathName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

GetDrive Method

Returns a **Drive** object corresponding to the drive in a specified path.

object.**GetDrive (** *drivespec* **)**;

Arguments

object
: Required. Always the name of a **FileSystemObject**.

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).

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.

To call the **GetDrive** method on a normal path string, use the following sequence to get a string that is suitable for use as *drivespec*:

```
[JScript]
DriveSpec = GetDriveName(GetAbsolutePathName(Path))
```

[JScript]

The following example illustrates the use of the **GetDrive** method.

```
[JScript]
function ShowFreeSpace(drvPath)
{
   var fso, d, s ="";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + drvPath.toUpperCase( ) + " - ";
   s += d.VolumeName + "<br>";
   s += "Free Space: " + d.FreeSpace/1024 + " Kbytes";
   return(s);
}
[VBScript]
DriveSpec = GetDriveName(GetAbsolutePathName(Path))
```

[VBScript]

The following example illustrates use of the **GetDrive** method:

```
[VBScript]
Function ShowFreeSpace(drvPath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & d.VolumeName   & "<BR>"
   s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
   s = s & " Kbytes"
   ShowFreeSpace = s
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Scripting Runtime Library

GetDriveName Method

Returns a string containing the name of the drive for a specified path.

`object.GetDriveName(path)`

Arguments

object
Required. Always the name of a **FileSystemObject**.

path
Required. The path specification for the component whose drive name is to be returned.

Remarks

The **GetDriveName** method returns a zero-length string ("") if the drive can't be determined.

> **Note** 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 following example illustrates the use of the **GetDriveName** method.

```
[JScript]
function GetDriveLetter(path)
{
   var fso, s ="";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   s += fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(path)));
   return(s);
}
[VBScript]
Function GetAName(DriveSpec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   GetAName = fso.GetDriveName(Drivespec)
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

GetExtensionName Method

Returns a string containing the extension name for the last component in a path.

`*object*.GetExtensionName(*path*)`

Arguments

object
  Required. Always the name of a **FileSystemObject**.
path
  Required. The path specification for the component whose extension name is to be returned.

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.

The following example illustrates the use of the **GetExtensionName** method.

```
[JScript]
function ShowExtensionName(filespec)
```

```
{
   var fso, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   s += fso.GetExtensionName(filespec);
   return(s);
}
[VBScript]
Function GetAnExtension(DriveSpec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   GetAnExtension = fso.GetExtensionName(Drivespec)
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

GetFile Method

Returns a **File** object corresponding to the file in a specified path.

`object.GetFile(filespec)`

Arguments

object
    Required. Always the name of a **FileSystemObject**.

filespec

Required. The *filespec* is the path (absolute or relative) to a specific file.

Remarks

An error occurs if the specified file does not exist.

The following example illustrates the use of the **GetFile** method.

```
[JScript]
function ShowFileAccessInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = f.Path.toUpperCase() + "<br>";
   s += "Created: " + f.DateCreated + "<br>";
   s += "Last Accessed: " + f.DateLastAccessed + "<br>";
   s += "Last Modified: " + f.DateLastModified
   return(s);
}
[VBScript]
Function ShowFileAccessInfo(filespec)
   Dim fso, f, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   s = f.Path & "<br>"
   s = s & "Created: " & f.DateCreated & "<br>"
   s = s & "Last Accessed: " & f.DateLastAccessed & "<br>"
   s = s & "Last Modified: " & f.DateLastModified
   ShowFileAccessInfo = s
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

GetFileName Method

Returns the last component of specified path that is not part of the drive specification.

object.**GetFileName(***pathspec***)**

Arguments

object
  Required. Always the name of a **FileSystemObject**.
pathspec
  Required. The path (absolute or relative) to a specific file.

Remarks

The **GetFileName** method returns a zero-length string ("") if *pathspec* does not end with the named component.

> **Note** 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 following example illustrates the use of the **GetFileName** method.

```
[JScript]
function ShowFileName(filespec)
{
   var fso, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   s += fso.GetFileName(filespec);
   return(s);
}
[VBScript]
Function GetAName(DriveSpec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   GetAName = fso.GetFileName(DriveSpec)
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Scripting Runtime Library

GetFileVersion Method

Returns the version number of a specified file.

***object*.GetFileVersion(*pathspec*)**

Arguments

object
    Required. Always the name of a **FileSystemObject**.
pathspec
    Required. The path (absolute or relative) to a specific file.

Remarks

The **GetFileVersion** method returns a zero-length string ("") if *pathspec* does not end with the named component.

> **Note** 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 following example illustrates the use of the **GetFileVersion** method.

```
[JScript]
function ShowFileName(filespec){
   var fso, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   s += fso.GetFileVersion(filespec);
   return(s);
}
[VBScript]
Function GetVersion(DriveSpec)
   Dim fso, temp
   Set fso = CreateObject("Scripting.FileSystemObject")
   temp = fso.GetFileVersion(pathspec)
   If Len(temp) Then
      GetVersion = temp
   Else
      GetVersion = "No version information available."
   End If
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

GetFolder Method

Returns a **Folder** object corresponding to the folder in a specified path.

```
object.GetFolder(folderspec)
```

Arguments

object
    Required. Always the name of a **FileSystemObject**.
folderspec
    Required. The *folderspec* is the path (absolute or relative) to a specific folder.

Remarks

An error occurs if the specified folder does not exist.

The following example illustrates the use of the **GetFolder** method.

```
[JScript]
function ShowFolderList(folderspec)
{
   var fso, f, fc, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(folderspec);
   fc = new Enumerator(f.SubFolders);
   s = "";
   for (; !fc.atEnd(); fc.moveNext())
   {
      s += fc.item();
      s += "<br>";
   }
   return(s);
}
[VBScript]
Sub AddNewFolder(path, folderName)
```

```
   Dim fso, f, fc, nf
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(path)
   Set fc = f.SubFolders
   If folderName <> "" Then
      Set nf = fc.Add(folderName)
   Else
      Set nf = fc.Add("New Folder")
   End If
End Sub
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Scripting Runtime Library

GetParentFolderName Method

Returns a string containing the name of the parent folder of the last component in a specified path.

object.GetParentFolderName(*path*)

Arguments

object
: Required. Always the name of a **FileSystemObject**.

path
Required. The path specification for the component whose parent folder name is to be returned.

Remarks

The **GetParentFolderName** method returns a zero-length string ("") if there is no parent folder for the component specified in the *path* argument.

> **Note** 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 following example illustrates the use of the **GetParentFolderName** method.

```
[JScript]
function ShowParentFolderName(filespec)
{
   var fso, s = "";
   fso = new ActiveXObject("Scripting.FileSystemObject");
   s += fso.GetParentFolderName(filespec);
   return(s);
}
[VBScript]
Function GetTheParent(DriveSpec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   GetTheParent = fso.GetParentFolderName(Drivespec)
End Function
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetSpecialFolder Method | GetTempName Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

GetSpecialFolder Method

Returns the special folder object specified.

object.**GetSpecialFolder(***folderspec***)**

Arguments

object
Required. Always the name of a **FileSystemObject**.

folderspec
Required. The name of the special folder to be returned. Can be any of the constants shown in the Settings section.

Settings

The *folderspec* argument can have any of the following values:

Constant	Value	Description
WindowsFolder	0	The Windows folder contains files installed by the Windows operating system.
SystemFolder	1	The System folder contains 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.

The following example illustrates the use of the **GetSpecialFolder** method.

```
[JScript]
var fso, tempfile;
fso = new ActiveXObject("Scripting.FileSystemObject");

function CreateTempFile()
{
   var tfolder, tfile, tname, fname, TemporaryFolder = 2;
   tfolder = fso.GetSpecialFolder(TemporaryFolder);
   tname = fso.GetTempName();
   tfile = tfolder.CreateTextFile(tname);
   return(tfile);
}
tempfile = CreateTempFile();
tempfile.writeline("Hello World");
tempfile.close();
[VBScript]
Dim fso, tempfile
Set fso = CreateObject("Scripting.FileSystemObject")

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

Set tempfile = CreateTempFile
tempfile.WriteLine "Hello World"
tempfile.Close
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetTempName Method
Applies To: FileSystemObject Object

Scripting Runtime Library

GetTempName Method

Returns a randomly generated temporary file or folder name that is useful for performing operations that require a temporary file or folder.

```
object.GetTempName ( );
```

The optional *object* is always the name of a **FileSystemObject**.

Remarks

The **GetTempName** method does not create a file. It provides only a temporary file name that can be used with **CreateTextFile** to create a file.

The following example illustrates the use of the **GetTempName** method.

```
[JScript]
var fso, tempfile;
fso = new ActiveXObject("Scripting.FileSystemObject");

function CreateTempFile()
{
   var tfolder, tfile, tname, fname, TemporaryFolder = 2;
   tfolder = fso.GetSpecialFolder(TemporaryFolder);
   tname = fso.GetTempName();
   tfile = tfolder.CreateTextFile(tname);
   return(tfile);
}
tempfile = CreateTempFile();
tempfile.writeline("Hello World");
tempfile.close();
[VBScript]
Dim fso, tempfile
Set fso = CreateObject("Scripting.FileSystemObject")

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

Set tempfile = CreateTempFile
tempfile.WriteLine "Hello World"
tempfile.Close
```

See Also

GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFileVersion Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method
Applies To: FileSystemObject Object

Scripting Runtime Library

Move Method

Moves a specified file or folder from one location to another.

`object.Move( destination );`

Arguments

object
    Required. Always the name of a **File** or **Folder** object.
destination
    Required. Destination where the file or folder is to be moved. Wildcard characters are not allowed.

Remarks

The results of the **Move** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.MoveFile** or **FileSystemObject.MoveFolder**. You should note, however, that the alternative methods are capable of moving multiple files or folders.

See Also

Copy Method | Delete Method | MoveFile Method | MoveFolder Method | OpenAsTextStream Method
Applies To: File Object | Folder Object

Scripting Runtime Library

MoveFile Method

Moves one or more files from one location to another.

```
object.MoveFile ( source, destination );
```

Arguments

object
    Required. Always the name of a **FileSystemObject**.
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.

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.

Note This method allows moving files between volumes only if supported by the operating system.

The following example illustrates the use of the **MoveFile** method:

```
[JScript]
function MoveFile2Desktop(filespec)
{
   var fso;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fso.MoveFile(filespec, "c:\\windows\\desktop\\");
}
[VBScript]
Sub MoveAFile(Drivespec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   fso.MoveFile Drivespec, "c:\windows\desktop\"
End Sub
```

See Also

CopyFile Method | DeleteFile Method | GetFile Method | GetFileName Method | Move Method | MoveFolder Method | OpenTextFile Method
Applies To: FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

MoveFolder Method

Moves one or more folders from one location to another.

```
object.MoveFolder ( source, destination );
```

Arguments

object
Required. Always the name of a **FileSystemObject**.

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.

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 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.

Important This method allows moving folders between volumes only if supported by the operating system.

The following example illustrates the use of the **MoveFolder** method:

```
[JScript]
```

```
function MoveFldr2Desktop(fldrspec)
{
   var fso;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fso.MoveFolder(fldrspec, "c:\\windows\\desktop\\");
}
[VBScript]
Sub MoveAFolder(Drivespec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   fso.MoveFolder Drivespec, "c:\windows\desktop\"
End Sub
```

See Also

CopyFile Method | DeleteFile Method | GetFile Method | GetFileName Method | Move Method | MoveFile Method | OpenTextFile Method
Applies To: FileSystemObject Object

Scripting Runtime Library

OpenAsTextStream Method

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

object.**OpenAsTextStream(**[*iomode*, [*format*]]**)**

Arguments

object
    Required. Always the name of a **File** object.
iomode

Optional. Indicates input/output mode. Can be one of three constants: **ForReading**, **ForWriting**, or **ForAppending**.

format

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

Settings

The *iomode* argument can have any of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

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.

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.

The following code illustrates the use of the **OpenAsTextStream** method:

```
[JScript]
function TextStreamTest( )
{
   var fso, f, ts, s;
   var ForReading = 1, ForWriting = 2, ForAppending = 8;
   var TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fso.CreateTextFile( "test1.txt" );           // Create a file.
   f = fso.GetFile("test1.txt");
   ts = f.OpenAsTextStream(ForWriting, TristateUseDefault);
   ts.Write( "Hello World" );
   ts.Close( );
   ts = f.OpenAsTextStream(ForReading, TristateUseDefault);
   s = ts.ReadLine( );
   ts.Close( );
   return(s);
}
[VBScript]
Function TextStreamTest
   Const ForReading = 1, ForWriting = 2, ForAppending = 8
   Const TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0
   Dim fso, f, ts
   Set fso = CreateObject("Scripting.FileSystemObject")
   fso.CreateTextFile "test1.txt"    ' Create a file.
   Set f = fso.GetFile("test1.txt")
   Set ts = f.OpenAsTextStream(ForWriting, TristateUseDefault)
   ts.Write "Hello World"
   ts.Close
   Set ts = f.OpenAsTextStream(ForReading, TristateUseDefault)
   TextStreamTest = ts.ReadLine
   ts.Close
End Function
```

See Also

Copy Method | CreateTextFile Method | Delete Method | Move Method | OpenTextFile Method
Applies To: File Object

Scripting Runtime Library

OpenTextFile Method

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

```
object.OpenTextFile(filename[, iomode[, create[, format]]])
```

Arguments

object
Required. *Object* is always the name of a **FileSystemObject**.
filename
Required. *String expression* that identifies the file to open.
iomode
Optional. Can be one of three constants: **ForReading**, **ForWriting**, or **ForAppending**.
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 created, **False** if it isn't created. If omitted, a new file isn't created.
format
Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Settings

The *iomode* argument can have any of the following settings:

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

The *format* argument can have any of the following settings:

Value	Description
TristateTrue	Open the file as Unicode.
TristateFalse	Open the file as ASCII.
TristateUseDefault	Open the file using the system default.

Remarks

The following code illustrates the use of the **OpenTextFile** method to open a file for appending text:

```
[JScript]
var fs, a, ForAppending;
ForAppending = 8;
fs = new ActiveXObject("Scripting.FileSystemObject");
a = fs.OpenTextFile("c:\\testfile.txt", ForAppending, false);
...
a.Close();
[VBScript]
Sub OpenTextFileTest
   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
End Sub
```

See Also

CreateTextFile Method | OpenAsTextStream Method
Applies To: FileSystemObject Object

Scripting Runtime Library

Read Method

Reads a specified number of characters from a **TextStream** file and returns the resulting string.

```
object.Read(characters)
```

Arguments

object
: Required. Always the name of a **TextStream** object.

characters
: Required. Number of characters you want to read from the file.

The following example illustrates how to use the **Read** method to read a six character header from a file and return the resulting string:

```
[JScript]
function GetHeader()
{
   var fso, f;
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true);
   f.Write("Header");
   f.Write("1234567890987654321");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   return(f.Read(6));
}
[VBScript]
Function ReadTextFileTest
   Const ForReading = 1, ForWriting = 2, ForAppending = 8
   Dim fso, f, Msg
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.Write "Hello world!"
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   ReadTextFileTest = f.Read(5)
End Function
```

See Also

ReadAll Method | ReadLine Method | Skip Method | SkipLine Method
Applies To: TextStream Object

Scripting Runtime Library

ReadAll Method

Reads an entire **TextStream** file and returns the resulting string.

```
object.ReadAll( );
```

The *object* is always the name of a **TextStream** object.

Remarks

For large files, using the **ReadAll** method wastes memory resources. Other techniques should be used to input a file, such as reading a file line by line.

The following example illustrates the use of the **ReadAll** method:

```
[JScript]
function GetEverything()
{
   var fso, f;
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true);
   f.Write("Header");
   f.Write("1234567890987654321");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   return(f.ReadAll());
}
[VBScript]
Function ReadAllTextFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.Write "Hello world!"
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   ReadAllTextFile =   f.ReadAll
End Function
```

See Also

Read Method | ReadLine Method | Skip Method | SkipLine Method
Applies To: TextStream Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

ReadLine Method

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

`object.ReadLine( )`

The *object* argument is always the name of a **TextStream** object.

Remarks

The following example illustrates the use of the **Line** property:

```
[JScript]
function GetLine()
{
   var fso, f, r;
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true);
   f.WriteLine("Hello world!");
   f.WriteLine("JScript is fun");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   r =  f.ReadLine();
   return(r);
}
[VBScript]
Function ReadLineTextFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, MyFile
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set MyFile = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   MyFile.WriteLine "Hello world!"
   MyFile.WriteLine "The quick brown fox"
   MyFile.Close
   Set MyFile = fso.OpenTextFile("c:\testfile.txt", ForReading)
   ReadLineTextFile = MyFile.ReadLine    ' Returns "Hello world!"
End Function
```

See Also

Read Method | ReadAll Method | Skip Method | SkipLine Method
Applies To: TextStream Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Skip Method

Skips a specified number of characters when reading a **TextStream** file.

`object.`**`Skip(`**`characters`**`)`**

Arguments

object
Required. Always the name of a **TextStream** object.

characters
Required. Number of characters to skip when reading a file.

Remarks

Skipped characters are discarded.

The following example illustrates the use of the **Skip** method:

```
[JScript]
function SkipDemo()
{
   var fso, f, r;
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject")
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true);
   f.WriteLine("Hello world!");
   f.WriteLine("JScript is fun");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   f.Skip(6);
   r = f.ReadLine();
   return(r);
}
[VBScript]
Function SkipTextFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.Write "Hello world!"
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   f.Skip(6)
   SkipTextFile =   f.ReadLine
End Function
```

See Also

Close Method | Read Method | ReadAll Method | ReadLine Method | SkipLine Method | Write Method | WriteLine Method | WriteBlankLines Method
Applies To: TextStream Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

SkipLine Method

Skips the next line when reading a **TextStream** file.

```
object.SkipLine( )
```

The *object* is always the name of a **TextStream** object.

Remarks

The following examples illustrates the use of the **SkipLine** method:

```
[JScript]
function SkipLineDemo()
{
   var fso, f, r
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject")
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true)
   f.WriteLine("Hello world!");
   f.WriteLine("JScript is fun");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   f.SkipLine();
   r = f.ReadLine();
   return(r);
}
[VBScript]
Function SkipLineInFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.Write "Hello world!" & vbCrLf & "VB Script is fun!"
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   f.SkipLine
   SkipLineInFile = f.ReadLine
End Function
```

See Also

Read Method | ReadAll Method | ReadLine Method | Skip Method
Applies To: TextStream Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Write Method

Writes a specified string to a **TextStream** file.

`object.Write(string)`

Arguments

object
Required. Always the name of a **TextStream** object.

string
Required. The text you want to write to the file.

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.

The following example illustrates the use of the **Write** method:

```
[JScript]
function WriteDemo()
{
   var fso, f, r
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject")
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true)
   f.Write("Hello world!");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   r = f.ReadLine();
   return(r);
}
[VBScript]
Function WriteToFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.Write "Hello world!"
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   WriteToFile =   f.ReadLine
End Function
```

See Also

WriteBlankLines Method | WriteLine Method
Applies To: TextStream Object

Scripting Runtime Library

WriteBlankLines Method

Writes a specified number of newline characters to a **TextStream** file.

```
object.WriteBlankLines(lines)
```

Arguments

object
Required. Always the name of a **TextStream** object.

lines
Required. Number of newline characters you want to write to the file.

Remarks

The following example illustrates the use of the **WriteBlankLines** method:

```
[JScript]
function WriteBlanksDemo()
{
   var fso, f, r;
   var ForReading = 1, ForWriting = 2;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.OpenTextFile("c:\\testfile.txt", ForWriting, true);
   f.Write("Hello world!");
   f.WriteBlankLines(2);
   f.Write("JScript is fun!");
   f.Close();
   f = fso.OpenTextFile("c:\\testfile.txt", ForReading);
   r = f.ReadAll();
   return(r);
}
[VBScript]
Function WriteBlankLinesToFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.WriteBlankLines 2
   f.WriteLine "Hello World!"
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   WriteBlankLinesToFile = f.ReadAll
End Function
```

See Also

Write Method | WriteLine Method
Applies To: TextStream Object

Scripting Runtime Library

WriteLine Method

Writes a specified string and newline character to a **TextStream** file.

object.**WriteLine(**[*string*]**)**

Arguments

object
: Required. Always the name of a **TextStream** object.

string
: Optional. The text you want to write to the file. If omitted, a newline character is written to the file.

Remarks

The following example illustrates use of the **WriteLine** method:

```
[JScript]
var fso, f;
fso = new ActiveXObject("Scripting.FileSystemObject");
f = fso.CreateTextFile("c:\\testfile.txt", true);
f.WriteLine("This is a test.");
f.Close();
[VBScript]
Function WriteLineToFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   f.WriteLine "Hello world!"
   f.WriteLine "VBScript is fun!"
   Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
   WriteLineToFile = f.ReadAll
End Function
```

See Also

Write Method | WriteBlankLines Method
Applies To: TextStream Object

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

FileSystemObject Objects

In This Section

Dictionary Object
	Object that stores data key, item pairs.
Drive Object
	Provides access to the properties of a particular disk drive or network share.
File Object
	Provides access to all the properties of a file.
FileSystemObject Object
	Provides access to a computer's file system.
Folder Object
	Provides access to all the properties of a folder.
TextStream Object
	Facilitates sequential access to file.

Related Sections

Scripting Run-Time Reference
	List of elements that make up Scripting Run-Time Reference.
FileSystemObject Basics
	A guide to the fundamentals of the FileSystemObject.
FileSystemObject Collections
	List of collections you can use with the FileSystemObject object model.

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Drive Object

Provides access to the properties of a particular disk drive or network share.

Remarks

The following code illustrates the use of the **Drive** object to access drive properties:

```
[JScript]
function ShowFreeSpace(drvPath)
{
   var fso, d, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   d = fso.GetDrive(fso.GetDriveName(drvPath));
   s = "Drive " + drvPath + " - " ;
   s += d.VolumeName + "<br>";
   s += "Free Space: " + d.FreeSpace/1024 + " Kbytes";
   return(s);
}
[VBScript]
Function ShowFreeSpace(drvPath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & d.VolumeName   & "<BR>"
   s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
   s = s & " Kbytes"
   ShowFreeSpace = s
End Function
```

Methods

The **Drive** object has no methods.

Properties

AvailableSpace Property | DriveLetter Property | DriveType Property | FileSystem Property | FreeSpace Property | IsReady Property | Path Property | RootFolder Property | SerialNumber Property | ShareName Property | TotalSize Property | VolumeName Property

See Also

Drives Collection | File Object | Files Collection | Folder Object | Folders Collection | GetDrive Method

---

Scripting Runtime Library

File Object

Provides access to all the properties of a file.

Remarks

The following code illustrates how to obtain a **File** object and how to view one of its properties.

```
[JScript]
function ShowFileInfo(filespec)
{
   var fso, f, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFile(filespec);
   s = f.DateCreated;
   return(s);
}
[VBScript]
Function ShowDateCreated(filespec)
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFile(filespec)
   ShowDateCreated = f.DateCreated
End Function
```

Methods

Copy Method | Delete Method | Move Method | OpenAsTextStream Method

Properties

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | Type Property

See Also

Drive Object | Drives Collection | Files Collection | Folder Object | Folders Collection

---

Scripting Runtime Library

FileSystemObject Object

Provides access to a computer's file system.

Remarks

[JScript]

The following code illustrates how the **FileSystemObject** is used to return a **TextStream** object that can be read from or written to:

```
[JScript]
```

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

[JScript]

In the example code, the **ActiveXObject** object is assigned to the **FileSystemObject** (fso). The **CreateTextFile** method then creates the file as a **TextStream** object (a), and the **WriteLine** method writes a line of text to the created text file. The **Close** method flushes the buffer and closes the file.

[VBScript]

The following code illustrates how the **FileSystemObject** is used to return a **TextStream** object that can be read from or written to:

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

[VBScript]

In the preceding code, the **CreateObject** function returns the **FileSystemObject** (`fso`). The **CreateTextFile** method then creates the file as a **TextStream** object (`a`) and the **WriteLine** method writes a line of text to the created text file. The **Close** method flushes the buffer and closes the file.

Methods

BuildPath Method | CopyFile Method | CopyFolder Method | CreateFolder Method | CreateTextFile Method | DeleteFile Method | DeleteFolder Method | DriveExists Method | FileExists Method | FolderExists Method | GetAbsolutePathName Method | GetBaseName Method | GetDrive Method | GetDriveName Method | GetExtensionName Method | GetFile Method | GetFileName Method | GetFolder Method | GetParentFolderName Method | GetSpecialFolder Method | GetTempName Method | MoveFile Method | MoveFolder Method | OpenTextFile Method

Properties

Drives Property

See Also

Dictionary Object | Drive Object | Drives Collection | File Object | Files Collection | Folder Object | Folders Collection | TextStream Object

---

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Folder Object

Provides access to all the properties of a folder.

Remarks

The following code illustrates how to obtain a **Folder** object and how to return one of its properties:

```
[JScript]
function ShowFolderInfo(folderspec)
{
   var fso, folder, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   folder = fso.GetFolder(folderspec);
   s = folder.DateCreated;
   return(s);
}
[VBScript]
Function ShowDateCreated(folderspec)
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(folderspec)
   ShowDateCreated = f.DateCreated
```

```
End Function
```

Methods

Copy Method | Delete Method | Move Method | OpenAsTextStream Method

Properties

Attributes Property | DateCreated Property | DateLastAccessed Property | DateLastModified Property | Drive Property | Files Property | IsRootFolder Property | Name Property | ParentFolder Property | Path Property | ShortName Property | ShortPath Property | Size Property | SubFolders Property | Type Property

See Also

Drive Object | Drives Collection | File Object | Files Collection | Folders Collection

---

Scripting Runtime Library

TextStream Object

Facilitates sequential access to file.

```
TextStream.{property  | method( )}
```

The *property* and *method* arguments can be any of the properties and methods associated with the **TextStream** object. Note that in actual usage, **TextStream** is replaced by a variable placeholder representing the **TextStream** object returned from the **FileSystemObject**.

Remarks

In the following code, a is the **TextStream** object returned by the **CreateTextFile** method on the **FileSystemObject**:

```
[JScript]
var fso = new ActiveXObject("Scripting.FileSystemObject");
var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
[VBScript]
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile= fso.CreateTextFile("c:\testfile.txt", True)
MyFile.WriteLine("This is a test.")
MyFile.Close
```

WriteLine and **Close** are two methods of the **TextStream** object.

Methods

Close Method | Read Method | ReadAll Method | ReadLine Method | Skip Method | SkipLine Method | Write Method | WriteBlankLines Method | WriteLine Method

Properties

AtEndOfLine Property | AtEndOfStream Property | Column Property | Line Property

See Also

Dictionary Object | FileSystemObject Object

FileSystemObject Collections

In This Section

Drives Collection
    Read-only collection of all available drives.
Files Collection
    Collection of all **File** objects within a folder.
Folders Collection
    Collection of all **Folder** objects contained within a **Folder** object.

Related Sections

Scripting Run-Time Reference
    List of elements that make up Scripting Run-Time Reference.
FileSystemObject Basics
    A guide to the fundamentals of the FileSystemObject.

Scripting Runtime Library

Drives Collection

Read-only collection of all available drives.

Remarks

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

[JScript]

The following example illustrates how to get the **Drives** collection using the **Drives** property and iterate the collection using the **Enumerator** object:

```
[JScript]
function ShowDriveList()
{
   var fso, s, n, e, x;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   e = new Enumerator(fso.Drives);
   s = "";
   for (; !e.atEnd(); e.moveNext())
   {
      x = e.item();
      s = s + x.DriveLetter;
      s += " - ";
      if (x.DriveType == 3)
         n = x.ShareName;
      else if (x.IsReady)
         n = x.VolumeName;
      else
         n = "[Drive not ready]";
      s +=   n + "<br>";
   }
   return(s);
}
```

[VBScript]

The following code illustrates how to get the **Drives** collection and iterate the collection using the **For Each...Next** statement:

```
[VBScript]
Function ShowDriveList
   Dim fso, d, dc, s, n
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set dc = fso.Drives
   For Each d in dc
      n = ""
      s = s & d.DriveLetter & " - "
      If d.DriveType = Remote Then
         n = d.ShareName
      ElseIf d.IsReady Then
         n = d.VolumeName
      End If
      s = s & n & "<BR>"
   Next
   ShowDriveList = s
End Function
```

Methods

The **Drives** collection has no methods.

Properties

Count Property | Item Property

See Also

Drive Object | Drives Property | File Object | Files Collection | Folder Object | Folders Collection

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Files Collection

Collection of all **File** objects within a folder.

Remarks

[JScript]

The following example illustrates how to get a **Files** collection and iterate the collection using the **Enumerator** object and the **for** statement:

```
[JScript]
function ShowFolderFileList(folderspec)
{
   var fso, f, f1, fc, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(folderspec);
   fc = new Enumerator(f.files);
   s = "";
   for (; !fc.atEnd(); fc.moveNext())
   {
      s += fc.item();
      s += "<br>";
   }
   return(s);
}
```

[VBScript]

The following code illustrates how to get a **Files** collection and iterate the collection using the **For Each...Next** statement:

```
[VBScript]
Function ShowFolderList(folderspec)
   Dim fso, f, f1, fc, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(folderspec)
   Set fc = f.Files
   For Each f1 in fc
      s = s & f1.name
      s = s & "<BR>"
   Next
   ShowFolderList = s
End Function
```

Methods

The **Files** collection has no methods.

Properties

Count Property | Item Property

See Also

Drive Object | Drives Collection | File Object | Folder Object | Folders Collection

---

Scripting Runtime Library

Folders Collection

Collection of all **Folder** objects contained within a **Folder** object.

Remarks

[JScript]

The following example illustrates how to get a **Folders** collection and how to iterate the collection using the **Enumerator** object and the **for** statement:

```
[JScript]
function ShowFolderList(folderspec)
{
   var fso, f, fc, s;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   f = fso.GetFolder(folderspec);
   fc = new Enumerator(f.SubFolders);
   s = "";
   for (; !fc.atEnd(); fc.moveNext())
   {
      s += fc.item();
      s += "<br>";
   }
   return(s);
}
```

[VBScript]

The following code illustrates how to get a **Folders** collection and how to iterate the collection using the **For Each...Next** statement:

```
[VBScript]
Function ShowFolderList(folderspec)
   Dim fso, f, f1, fc, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.GetFolder(folderspec)
   Set fc = f.SubFolders
   For Each f1 in fc
      s = s & f1.name
      s = s &   "<BR>"
   Next
   ShowFolderList = s
End Function
```

Methods

Add Method (Folders)

Properties

Count Property | Item Property

See Also

Drive Object | Drives Collection | File Object | Files Collection | Folder Object | SubFolders Property

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Script Encoder Overview

Script Encoder is a simple command-line tool that enables script designers to encode their final script so that Web hosts and Web clients cannot view or modify their source. Note that this encoding only prevents casual viewing of your code; it will not prevent the determined hacker from seeing what you've done and how.

Web designers use scripting on Web pages and server-side active server pages (.ASP) to add virtually every kind of feature you can imagine. In addition, scripting is used by the Windows® Scripting Host (WSH) and in a number of other applications with equally impressive results.

Up to now, one of the shortcomings of using scripts is that they afford no protection of the intellectual property contained within, nor do they provide any assurance that what users get is what you created. Clever algorithms and carefully designed scripts were always completely visible because they were stored as plain text. As a result, script users at every level could see the script designer's code and could then take it, modify it, and make it their own. Obviously, this is not good if you're trying to get an edge in a very competitive environment.

With the introduction of scriptlets, protecting the source code becomes even more important. Script designers want to use this simple component architecture, but they don't necessarily want to share their source code with the world. After a script is encoded, changing any part of the resulting file will render it inoperable, thus ensuring the absolute integrity of your encoded script.

See Also

Using Script Encoder | Script Encoder Syntax | Script Encoding Sample

---

Build: Topic Version 5.6.9309.1546

Scripting Runtime Library

Using Script Encoder

The Script Encoder encodes only scripting code, with all other file content left untouched to appear as plain text. To use the Script Encoder, develop and debug your script in the usual manner, then use this utility to encode your final script. The Script Encoder uses markers within your source code to identify where encoding should begin.

For Visual Basic® Scripting Edition (VBScript), the following example illustrates how the encoding marker is used to expose a plain-text copyright message:

```
<SCRIPT LANGUAGE="VBScript">
'Copyright© 1998. XYZ Productions. All rights reserved.
'**Start Encode**
' Your code goes here.
</SCRIPT>
```

In JScript®, the encoding marker looks like this:

```
<SCRIPT LANGUAGE="JScript">
//Copyright© 1998. ZYX Productions. All rights reserved.
//**Start Encode**
// Your code goes here.
</SCRIPT>
```

When the Script Encoder is invoked, anything in the script block before the start marker is left unencoded, while everything else in the script block is encoded. Therefore, if the start marker is omitted, the entire scripting block is encoded, but if the start marker is at the end of the scripting block, nothing is encoded.

After the encoding takes place, you should be aware that the language designator in the <SCRIPT> tag has changed. For VBScript, the new designator looks like this:

```
<SCRIPT LANGUAGE="VBScript.Encode">
```

For JScript, the new designator looks like this:

```
<SCRIPT LANGUAGE="JScript.Encode">
```

The Script Encoder is invoked on the MS-DOS command line or in the **Run** dialog box as follows:

SRCENC [switches] inputfile outputfile

See Also

Script Encoder Overview | Script Encoder Syntax | Script Encoding Sample

---

Scripting Runtime Library

Script Encoder Syntax

Encodes scripting source code so it cannot be easily viewed or modified by users.

Syntax

SCRENC [/s] [/f] [/xl] [/l defLanguage] [**/e** defExtension] input file output file

The Script Encoder syntax has these parts:

Part	Description
/s	Optional. Switch that specifies that the Script Encoder is to work silently, that is, produce no screen output. If omitted, the default is to provide verbose output.
/f	Optional. Specifies that the input file is to be overwritten by the output file. Note that this option destroys your original input source file. If omitted, the output file is not overwritten.
/xl	Optional. Specifies that the *@language* directive is not added at the top of .ASP files. If omitted, *@language* directive is added for all .ASP files.

/l *defLanguage*	Optional. Specifies the default scripting language (JScript® or VBScript) to use during encoding. Script blocks within the file being encoded that do not contain a language attribute are assumed to be of this specified language. If omitted, JScript is the default language for HTML pages and scriptlets, while VBScript is the default for active server pages. For plain text files, the file extension (either .js or .vbs) determines the default scripting language.
/e *defExtension*	Optional. Associates the input file with a specific file type. Use this switch when the input file's extension doesn't make the file type obvious, that is, when the input file extension is not one of the recognized extensions, but the file content does fall into one of the recognized types. There is no default for this option. If a file with an unrecognized extension is encountered and this option is not specified, the Script Encoder fails for that unrecognized file. Recognized file extensions are asa, asp, cdx, htm, html, js, sct, and vbs.
inputfile	Required. The name of the input file to be encoded, including any necessary path information relative to the current directory.
outputfile	Required. The name of the output file to be produced, including any necessary path information relative to the current directory.

Remarks

There are four kinds of files than can be processed by the Script Encoder. They are:

- ASP. This format consists of a text active server page containing valid HTML and embedded scripting blocks within <SCRIPT> ... </SCRIPT> tags or <% ... %> tags. Applications that use this format include Microsoft® Internet Information Services (IIS). Recognized file extensions are .asp, .asa, and .cdx.
- HTML. This format consists of a text file that contains valid HTML along with embedded script blocks. Applications using this scripting format include Microsoft FrontPage®, Microsoft® Visual InterDev™ and virtually all Web designers and browsers. Recognized file extensions are .htm and .html.
- Plain text. This format consists of text file that contains only script with no surrounding tags. Applications using scripting format include Windows® Scripting Host (WSH) and Microsoft® Outlook®. Recognized file extensions are .js and .vbs, which are changed to .jse and .vbe, respectively, after encoding.
- Scriptlet. This format consists of a text file that contains valid scriptlet code within <SCRIPT> ... </SCRIPT> tags. Recognized file extension is .sct and .wsh.

Examples

The following are examples of the use of the Script Encoder and a brief explanation of the results:

To encode input file test.html and produce output file encode.html, use:

```
screnc test.html encode.html
```

To encode input file test.htm and overwrite the input file with the encoded output file, use:

```
screnc /f test.htm
```

To encode all .ASP files in the current directory and place the encoded output files in c:\temp, use:

```
screnc *.asp c:\temp
```

To encode all files in the current directory as .ASP files and place them in c:\temp, use: screnc /e asp *.* c:\temp

To encode input file test.htm and produce output file encode.htm, ensuring that all script blocks that don't have a language attribute specified use VBScript, use:

```
screnc /l vbscript test.htm encode.htm
```

To encode all scriptlet files in the current directory and overwrite them with encoded files, while displaying no message, use:

```
screnc /s /f *.sct
```

See Also

Script Encoder Overview | Using Script Encoder | Script Encoding Sample

---

Scripting Runtime Library

Script Encoding Sample

Here is a short example of a Web page that includes some JScript code that needs protecting:

```
<HTML>
<HEAD>
<TITLE>Script Encoder Sample Page</TITLE>
<SCRIPT LANGUAGE="JScript">
<!--//
//Copyright© 1998 Microsoft Corporation. All Rights Reserved.
//**Start Encode**
function verifyCorrectBrowser(){
  if(navigator.appName == "Microsoft Internet Explorer")
    if (navigator.appVersion.indexOf ("5.") >= 0)
      return(true);
    else
      return(false);
}
function getAppropriatePage(){
  var str1 = "Had this been an actual Web site, a page compatible with ";
  var str2 = "browsers other than ";
  var str3 = "Microsoft Internet Explorer 5.0 ";
  var str4 = "would have been loaded.";
  if (verifyCorrectBrowser())
    document.write(str1 + str3 + str4);
  else
    document.write(str1 + str2 + str3 + str4);
}
//-->
</SCRIPT>
</HEAD>
<BODY onload="getAppropriatePage()">
</BODY>
</HTML>
Here's the same page as it appears after being run through the Script Encoder:
<HTML>
<HEAD>
<TITLE>Script Encoder Sample Page</TITLE>
<SCRIPT LANGUAGE="JScript.Encode">
<!--//
//Copyright© 1998 Microsoft Corporation. All Rights Reserved.
//**Start Encode**#@~^QwIAAA==@#@&0;mDkWP7nDb0zZKD.n1YAMGhk+Dvb`@#@&P,kW`UC7kLlDGDcl22gl:n~{'~Jtr1DGkW6YP&xDnD+OPA
```

```
//-->
</SCRIPT>
</HEAD>
<BODY onload="getAppropriatePage()">
</BODY>
</HTML>
```

> **Note** After encoding, if you change even one character in the encoded text, the integrity of the entire script is lost and it can no longer be used.

See Also

Script Encoder Overview | Using Script Encoder | Script Encoder Syntax

---