- Coding using VBScript and JScript,
- The scripting dictionary object,
- File access with ASP,
- Debugging ASP and error-handling.

Coding using VBScript and JScript,

JavaScript	VBScript
// This is a comment. /* Multi-line comment. */	'This is a comment. Rem So is this. No multi-line comments.
Variables, Constants, and Arrays	
var x, y var z = 10	Dim x, y Dim z ' Can't assign on same line z = 10
No way to force variables to be declared	'Force all vars to be declared before being used. Option Explicit
No constants	Const PI = 3.14
var Vector = new Array(10) var Names = new Array()	Dim Vector(9) ' 9 is last slot ' Can't do this – arrays must have upper bound
var Matrix = new Array(4) Matrix[0] = new Array(5)	Dim Matrix(3, 4)
Vector[9] = 1.5 Matrix[1][4] = 2	Vector(9) = 1.5 Matrix(1, 4) = 2
Output	
str = " there!" document.write("hello" + str) document.writeln("good\nbye")	str = " there!" document.write "hello" & str document.writeln "good" & vbCr & "bye"
Response.Write(str) 'Server-side <%= str %> 'Another way	Response.Write str 'Server-side <%= str %> 'Another way
alert("message") // Client-side	MsgBox "message" 'Client-side
Functions	

function Add(a, b)
{
return a + b
}
 	
Variable are always passed by value.
Only objects can be passed by reference.
	Function Add(ByVal x, ByVal y)
	Add = x + y
	End Function
	
	Sub Add2(ByVal x, ByVal y, ByRef ans)
	ans = x + y
	End Sub

sum = Add(1, 2)	sum = Add(1, 2) Add2 1, 2, sum ‘No () for Sub params Call Add2(1, 2, sum) ‘Alternate call
Decisions	
if (score == 100) alert("Great!")	If score = 100 Then _ MsgBox "Great!"

if (a == 1 || a == 5)
{
 vector[a] = b
 c = 1
}
else if (a == 2)
 b--
else
 b *= 2
	If a = 1 Or a = 5 Then
	 vector(a) = b
	 c = 1
	ElseIf a = 2 Then
	 b = b - 1
	Else
	 b = b * 2
	End If

switch(x)
{
 case 1: str = str + "end"
 break
 case 2:
 case 3: y = y * y * y
 break
 default: z = z / 2
}
	Select Case x
	 Case 1
	 str = str & "end"
	 Case 2, 3
	 y = y ^ 3
	 Case 4 To 10
	 ‘4 <= x <= 10
	 Case Is >= 25
	 ‘x >= 25
	 Case Else ‘Default
	 z = z \ 2
	End Select

Loops	
for (i = 1; i <= 5; i++) document.writeln(i)	For i = 1 To 5 document.writeln i Next
for (c = 2; c <= 10; c += 2) a += c	For c = 2 To 10 Step 2 a = a + c Next

for (var index in myArray)
  document.write(myArray[index])
	Dim element
	For Each element in myArray
	  document.write element
	Next

while (a < 10)
  a++
	Do While a < 10
	  a = a + 1
	Loop
	
	Do Until a = 10
	  a = a + 1
	  Loop

do
  a++;
while (a < 10)
	Do
	  a = a + 1
	Loop While a < 10
	
	Do
	  a = a + 1
	  Loop Until a = 10

Arithmetic Operators	
+ - * / % (mod)	+ - * / \ (integer division) Mod ^ (raising to a power)
Relational Operators	
< <= > >= == !=	< <= > >= = <>
Logic Operators	
&& || !	And Not Or Xor Eqv (Equivalent) Imp
Bitwise Operators	
& ~ | ^ << >>	And Not Or Xor (no shift left/right)
Strings	
var x x = "HU" x = x + " is great!"	Dim x x = "HU" x = x & " is great!"

VBScript String Functions

Function	Description	Example
StrConv	Used primarily to change the case of letters in the string. Can also do UNICODE and other format changes.	x = StrConv("Test", vbUpperCase) --> “TEST”
UCase, LCase	Returns string converted to uppercase/lowercase.	x = UCase(“Test”) --> “TEST”
Len	Returns the string length.	length = Len(“a”) + Len(x) --> 1 + 4
LSet, RSet	Justifies chars in fixed-length string to left or right side.	RSet y = “right” --> “ right”

Left, Right	Returns the specified number of chars from left-hand or right-hand side of string.	x = Left(“Harding”, 4) --> “Hard”
Mid	Returns substring from a search string.	x = Mid(“I love you.”, 3, 4) --> “love”
LTrim, RTrim, Trim	Returns a copy of the string with leading, trailing, or both leading and trailing spaces removed.	x = LTrim(“ <-trim-> “) -->“<-trim-> ” x = Trim(“ <-trim-> “) --> “<-trim->”
StrComp	Compares 2 strings. Returns –1 if str1 < str2, 1 if str1 > str2, 0 if strings are equal.	r = StrComp(“HU”, “ACU”) --> 1
InStr	Searches a string for a substring and returns the integer position if found, 0 if not found.	r = InStr(1, “To be or not to be”, “be”) --> 4
Asc, AscW	Asc returns ASCII value of given string. AscW returns UNICODE value.	n = Asc(“A”) --> 65 n = AscW(“A”) --> 65
Chr	Returns string equivalent of given ASCII value.	c = Chr(65) --> “A“
Str	Converts a number to a string.	x = Str(459) --> " 459"
Split	Returns an array of substrings produced from splitting a string based on a delimeter.	myArray = Split("IxLovexVBScript!", "x") ‘myArray(0) --> “I” ‘myArray(1) --> “Love” ‘myArray(2) --> “VBScript!”
StrReverse	Returns a reversed string.	x = StrReverse("VBScript") --> "tpircSBV"

The scripting dictionary object

The Dictionary object stores information in name/value pairs.

More Examples

- Does a specified key exist?
  How to create a Dictionary object, and then use the Exists method to check if a specified key exists.
- Return an array of all items
  How to use the Items method to return an array of all the items.
- Return an array of all keys
  How to use the Keys method to return an array of all the keys.
- Return the value of an item
  How to use the Item property to return the value of an item.
- Set a key
  How to use the Key property to set a key in a Dictionary object.
- Return the number of key/item pairs
  How to use the Count property to return the number of key/item pairs.

The Dictionary Object

The Dictionary object is used to store information in name/value pairs (referred to as key and item). The Dictionary object might seem similar to Arrays, however, the Dictionary object is a more desirable solution to manipulate related data.

Comparing Dictionaries and Arrays:

- Keys are used to identify the items in a Dictionary object
- You do not have to call ReDim to change the size of the Dictionary object
- When deleting an item from a Dictionary, the remaining items will automatically shift up
- Dictionaries cannot be multidimensional, Arrays can
- Dictionaries have more built-in functions than Arrays
- Dictionaries work better than arrays on accessing random elements frequently
- Dictionaries work better than arrays on locating items by their content

The following example creates a Dictionary object, adds some key/item pairs to it, and retrieves the item value for the key gr:

```
<%
Dim d
Set d=Server.CreateObject("Scripting.Dictionary")
d.Add "re","Red"
d.Add "gr","Green"
d.Add "bl","Blue"
d.Add "pi","Pink"
Response.Write("The value of key gr is: " & d.Item("gr"))
%>
```

Output:

```
The value of key gr is: Green
```

The Dictionary object's properties and methods are described below:

Properties

Property	Description
CompareMode	Sets or returns the comparison mode for comparing keys in a Dictionary object
Count	Returns the number of key/item pairs in a Dictionary object
Item	Sets or returns the value of an item in a Dictionary object
Key	Sets a new key value for an existing key value in a Dictionary object

Methods

Method	Description
Add	Adds a new key/item pair to a Dictionary object
Exists	Returns a Boolean value that indicates whether a specified key exists in the Dictionary object
Items	Returns an array of all the items in a Dictionary object
Keys	Returns an array of all the keys in a Dictionary object
Remove	Removes one specified key/item pair from the Dictionary object

RemoveAll	Removes all the key/item pairs in the Dictionary object

File access with ASP,

ASP FileSystemObject Object

The FileSystemObject object is used to access the file system on a server.

More Examples

- Does a specified file exist?
  How to check if a file exists.
- Does a specified folder exist?
  How to check if a folder exists.
- Does a specified drive exist?
  How to check if a drive exists.
- Get the name of a specified drive
  How to get the name of a specified drive.
- Get the name of the parent folder of a specified path
  How to get the name of the parent folder of a specified path.
- Get file name
  How to get the file name of the last component in a specified path.
- Get the file extension
  How to get the file extension of the last component in a specified path.
- Get the base name of a file or folder
  How to get the base name of a file or folder, in a specified path.

The FileSystemObject Object

The FileSystemObject object is used to access the file system on a server.

This object can manipulate files, folders, and directory paths. It is also possible to retrieve file system information with this object.

The following code creates a text file (c:\test.txt) and then writes some text to the file:

```
<%
dim fs,fname
set fs=Server.CreateObject("Scripting.FileSystemObject")
set fname=fs.CreateTextFile("c:\test.txt",true)
fname.WriteLine("Hello World!")
fname.Close
set fname=nothing
set fs=nothing
%>
```

The FileSystemObject object's properties and methods are described below:

Properties

Property	Description
Drives	Returns a collection of all Drive objects on the computer

Methods

Method	Description
BuildPath	Appends a name to an existing path
CopyFile	Copies one or more files from one location to another
CopyFolder	Copies one or more folders from one location to another
CreateFolder	Creates a new folder

CreateTextFile	Creates a text file and returns a TextStream object that can be used to read from, or write to the file
DeleteFile	Deletes one or more specified files
DeleteFolder	Deletes one or more specified folders
DriveExists	Checks if a specified drive exists
FileExists	Checks if a specified file exists
FolderExists	Checks if a specified folder exists
GetAbsolutePathName	Returns the complete path from the root of the drive for the specified path
GetBaseName	Returns the base name of a specified file or folder
GetDrive	Returns a Drive object corresponding to the drive in a specified path
GetDriveName	Returns the drive name of a specified path
GetExtensionName	Returns the file extension name for the last component in a specified path
GetFile	Returns a File object for a specified path
GetFileName	Returns the file name or folder name for the last component in a specified path
GetFolder	Returns a Folder object for a specified path
GetParentFolderName	Returns the name of the parent folder of the last component in a specified path
GetSpecialFolder	Returns the path to some of Windows' special folders
GetTempName	Returns a randomly generated temporary file or folder
MoveFile	Moves one or more files from one location to another
MoveFolder	Moves one or more folders from one location to another
OpenTextFile	Opens a file and returns a TextStream object that can be used to access the file

ASP TextStream Object

The TextStream object is used to access the contents of a text file.

More Examples

- Read textfile
  How to read from a text file.
- Read only a part of a textfile
  How to only read a part of a TextStream file.
- Read one line of a textfile
  How to read one line from a TextStream file.
- Read all lines from a textfile
  How to read all the lines from a TextStream file.
- Skip a part of a textfile
  How to skip a specified number of characters when reading the TextStream file.
- Skip a line of a textfile
  How to skip a line when reading the TextStream file.
- Return line-number
  How to return the current line number in a TextStream file.
- Get column number
  How to get the column number of the current character in a file.

The TextStream Object

The TextStream object is used to access the contents of text files.

The following code creates a text file (c:\test.txt) and then writes some text to the file (the variable f is an instance of the TextStream object):

```
<%
dim fs,f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.CreateTextFile("c:\test.txt",true)
f.WriteLine("Hello World!")
f.Close
set f=nothing
```

```
set fs=nothing
%>
```

To create an instance of the TextStream object you can use the CreateTextFile or OpenTextFile methods of the FileSystemObject object, or you can use the OpenAsTextStream method of the File object.

The TextStream object's properties and methods are described below:

Properties

Property	Description
AtEndOfLine	Returns true if the file pointer is positioned immediately before the end-of-line marker in a TextStream file, and false if not
AtEndOfStream	Returns true if the file pointer is at the end of a TextStream file, and false if not
Column	Returns the column number of the current character position in an input stream
Line	Returns the current line number in a TextStream file

Methods

Method	Description
Close	Closes an open TextStream file
Read	Reads a specified number of characters from a TextStream file and returns the result
ReadAll	Reads an entire TextStream file and returns the result
ReadLine	Reads one line from a TextStream file and returns the result
Skip	Skips a specified number of characters when reading a TextStream file
SkipLine	Skips the next line when reading a TextStream file
Write	Writes a specified text to a TextStream file
WriteLine	Writes a specified text and a new-line character to a TextStream file
WriteBlankLines	Writes a specified number of new-line character to a TextStream file

ASP Drive Object

The Drive object is used to get information about a local disk drive or a network share.

More Examples

- Get the total size of a specified drive
  How to get the total size of a specified drive.
- Get the available space of a specified drive
  How to get the available space of a specified drive.
- Get the free space of a specified drive
  How to get the free space of a specified drive.
- Get the drive letter of a specified drive
  How to get the drive letter of a specified drive.
- Get the drive type of a specified drive
  How to get the drive type of a specified drive.
- Get the file system of a specified drive
  How to get the file system of a specified drive.
- Is the drive ready?
  How to check whether a specified drive is ready.
- Get the path of a specified drive
  How to get the path of a specified drive.
- Get the root folder of a specified drive
  How to get the root folder of a specified drive.
- Get the serialnumber of a specified drive
  How to get the serialnumber of a specified drive.

The Drive Object

The Drive object is used to return information about a local disk drive or a network share. The Drive object can return information about a drive's type of file system, free space, serial number, volume name, and more.

Note: You cannot return information about a drive's content with the Drive object. For this purpose you will have to use the Folder object.

To work with the properties of the Drive object, you will have to create an instance of the Drive object through the FileSystemObject object. First; create a FileSystemObject object and then instantiate the Drive object through the GetDrive method or the Drives property of the FileSystemObject object.

The Drive object's properties are described below:

Properties

Property	Description
AvailableSpace	Returns the amount of available space to a user on a specified drive or network share
DriveLetter	Returns one uppercase letter that identifies the local drive or a network share
DriveType	Returns the type of a specified drive
FileSystem	Returns the file system in use for a specified drive
FreeSpace	Returns the amount of free space to a user on a specified drive or network share
IsReady	Returns true if the specified drive is ready and false if not
Path	Returns an uppercase letter followed by a colon that indicates the path name for a specified drive
RootFolder	Returns a Folder object that represents the root folder of a specified drive
SerialNumber	Returns the serial number of a specified drive
ShareName	Returns the network share name for a specified drive
TotalSize	Returns the total size of a specified drive or network share
VolumeName	Sets or returns the volume name of a specified drive

ASP File Object

The File object is used to return information about a specified file.

More Examples

- When was the file last modified?
  How to get the date and time a specified file was last modified.
- When was the file last accessed?
  How to get the date and time a specified file was last accessed.
- Return the attributes of a specified file
  How to return the attributes of a specified file.

The File Object

The File object is used to return information about a specified file.

To work with the properties and methods of the File object, you will have to create an instance of the File object through the FileSystemObject object. First; create a FileSystemObject object and then instantiate the File object through the GetFile method of the FileSystemObject object or through the Files property of the Folder object.

The following code uses the GetFile method of the FileSystemObject object to instantiate the File object and the DateCreated property to return the date when the specified file was created:

Example

```
<%
Dim fs,f
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set f=fs.CreateTextFile("c:\test.txt")
Response.Write("File created: " & f.DateCreated)
set f=nothing
set fs=nothing
%>
```

ASP Folder Object

The Folder Object is used to return information about a specified folder.

The Folder Object

The Folder object is used to return information about a specified folder.

To work with the properties and methods of the Folder object, you will have to create an instance of the Folder object through the FileSystemObject object. First; create a FileSystemObject object and then instantiate the Folder object through the GetFolder method of the FileSystemObject object.

The following code uses the GetFolder method of the FileSystemObject object to instantiate the Folder object and the DateCreated property to return the date when the specified folder was created:

```
<%
Dim fs,fo
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set fo=fs.CreateFolder("c:\test")
Response.Write("Folder created: " & fo.DateCreated)
set fo=nothing
set fs=nothing
%>
```

Output:

Folder created: 10/22/2008 10:01:19 AM

The Folder object's collections, properties, and methods are described below:

Collections

Collection	Description
Files	Returns a collection of all the files in a specified folder
SubFolders	Returns a collection of all subfolders in a specified folder

Properties

Property	Description
Attributes	Sets or returns the attributes of a specified folder
DateCreated	Returns the date and time when a specified folder was created
DateLastAccessed	Returns the date and time when a specified folder was last accessed
DateLastModified	Returns the date and time when a specified folder was last modified
Drive	Returns the drive letter of the drive where the specified folder resides
IsRootFolder	Returns true if a folder is the root folder and false if not
Name	Sets or returns the name of a specified folder
ParentFolder	Returns the parent folder of a specified folder
Path	Returns the path for a specified folder
ShortName	Returns the short name of a specified folder (the 8.3 naming convention)
ShortPath	Returns the short path of a specified folder (the 8.3 naming convention)
Size	Returns the size of a specified folder
Type	Returns the type of a specified folder

Methods

Method	Description
Copy	Copies a specified folder from one location to another
Delete	Deletes a specified folder
Move	Moves a specified folder from one location to another

CreateTextFile	Creates a new text file in the specified folder and returns a TextStream object to access the file

Debugging ASP and error-handling.

Types of Error

- **Compile-time errors:** These errors are usually in the syntax of the code and stop the ASP from compiling. You may have experienced this if you left the closing "Next" statement off of a "For" loop.
- **Runtime errors:** These happen when you try to execute the ASP page. For example, if you try setting a variable outside its allowed range.
- **Logic errors:** Logic errors are harder to detect. The problem lies within the structure of the code, and the computer cannot detect an error. These types require thorough testing before rolling out the application.

As compile-time errors are always trapped and logic errors are only found through thorough testing. This leaves us to worry only about runtime errors. These can stop the execution of your page and leave the user with a lot of non-user-friendly text on the screen.

Error Handling

Error handling allows you to display friendly error messages for the end users and at the same time it helps you debug your ASP application.

Well-written applications include error-handling code that allows them to recover gracefully from unexpected errors. When an error occurs, the application may need to request user intervention, or it may be able to recover on its own. In extreme cases, the application may log the user off or shut down the system.

The error handling functions enable you to receive and display error information for your application. For more information, see the following topics:

- Error Mode: The error mode indicates to the system how the application is going to respond to serious errors. Serious errors include disk failure, drive-not-ready errors, data misalignment, and unhandled exceptions.
- Last Error Code: When an error occurs, most system functions return an error code, usually 0, **NULL**, or –1.
- Notifying the User: To notify the user that some kind of error has occurred, many applications simply produce a sound by using the **MessageBeep**function or flash the window by using either the **FlashWindow** or **FlashWindowEx** function. An application can also use these functions to call attention to an error and then display a message box or an error message containing details about the error.
- Message Tables: Message tables are special string resources used when displaying error messages. They are declared in a resource file using the MESSAGETABLE resource-definition statement. To access the message strings, use the **FormatMessage** function.
- Fatal Application Exit: The **FatalAppExit** function displays a message box and terminates the application when the user closes the message box. This function should only be used as a last resort, because it may not free the memory or files owned by the application.

Debugging

A *debugger* is an application that enables a developer to observe and correct programming errors.

Identifying a problem, isolating the source of the problem, and then either correcting the problem or determining a way to work around it. The final step of debugging is to test the correction or workaround and make sure it works.

1) Add On Error Resume Next at top of the page
2) Add following code at bottom of the page

```
If Err.Number <> 0 Then

 Response.Write (Err.Description)

 Response.End
```

```
End If

On Error GoTo 0
```

Handling the Error

In ASP, the best way to handle errors is to place code at the bottom of each page that can display an appropriate message to the user. I also recommend using the buffer on every page. If an error occurs, the contents of the page can be cleared before displaying error details. This should be less confusing for the user and you. Here is some sample code:

```
<%

    Err.Clear
    On Error Resume Next //  This line simply tells the ASP interpreter, to continue with the executing of the ASP script if there is an error, instead of throwing exception and stopping the script execution.

    dim div
    div= 12/0

    If Err.Number <> 0 Then //If the Err.Number value is not 0 then we know that there has been an error in our application and we call a custom subroutine called HandleError passing it the error message as a parameter.

      Response.Write ("Error : " & Err.Description& "<br><br>")

      Response.End

    End If
    On Error GoTo 0  disable any error handling.
    %>
Output:
      Error : Division by zero
```

As you can see from above, I first set the On Error Resume Next so that errors don't stop page execution. Once the execution point falls to the Error Handler I clear the page from memory and return a complete error page to the user. This can say anything. It doesn't have to offer a printout of the error object or ask the user to contact the support desk. You could of course add some code to log the error in a file or a database.

Error Handling and Databases

Adding a database to the error-handling equation can complicate things. Assume that we have an ASP page where a couple of calls are made to a database to display some data, but then an insert/update query is executed at the bottom of the page. Now, because we have the On Error Resume Next switched on, if an error occurs in the select queries, the insert/update will still fire. This can cause data integrity problems within the database or fail to give the desired functionality. To prevent this, a check for an error must be made before any insert/update/delete queries are fired. To achieve this, wrap the database call up. It would look something like this:

```
If Err.Number = 0 And objConnection.Errors.Count = 0 Then

      ' Fire the database query, because there are no errors
      Set rstResults = dbData.Execute(txtSql)

End If
```

Redirects with the Error Handler

One more thing to watch out for is redirecting from the page before the execution point reaches the error handler. If a redirect happens, then the Error Handler is rendered useless. So you need to wrap any code that redirects, just like you did for the database calls. Here is an example:

```
If Err.Number = 0 And objConnection.Errors.Count = 0 Then

        ' OK to redirect
        Response.Clear
        Response.Redirect "<URL Here>"

End If
```