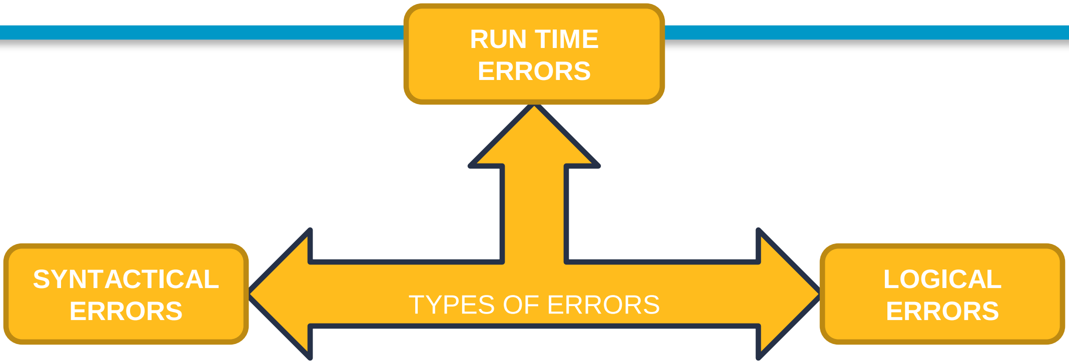
VB Script - Advanced

People matter, results count.

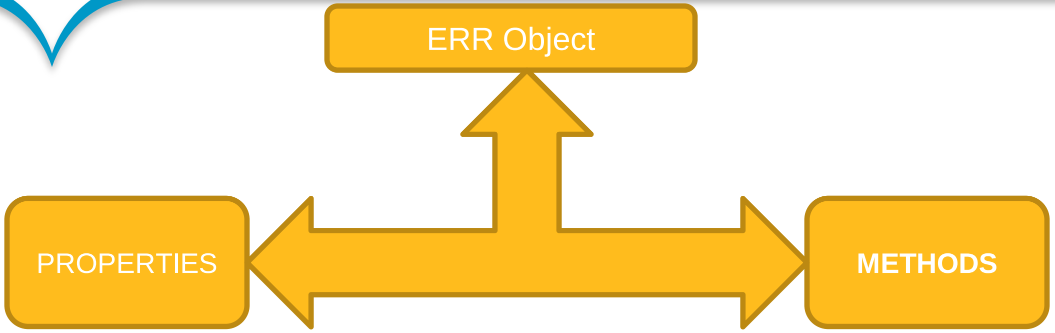
VB Script Advanced – Exception Handling



EXCEPTION HANDLING & USAGE

Exception handling is a programming language construct or computer hardware mechanism designed to handle the occurrence of **exceptions**, special conditions that change the normal flow of program execution.

VB Script Advanced – Exception Handling



- Description Property
- HelpFile Property
- Source Property
- HelpContext Property
- Number Property

- Clear Method
- Raise Method

PROPERTIES AND METHODS EXPLAINED...

PROPERTIES

❖ Description **Property**
This property returns or sets a string containing a brief textual description of an error.
Syntax: **object.Description [= string]**

❖ HelpContext **Property**
This property is used to set or return a context ID for a Help topic specified with the **HelpFile** property.
Syntax: **object.HelpContext [= contextID]**

❖ HelpFile **Property**
This property is used to get or define the path to a Help file.
Syntax: **object.HelpFile [= contextID]**

❖ Number **Property**
This property is used to retrieve or set the value that relates to a specific runtime error.
Syntax: **object.Number [= errnumber]**

❖ Source **Property**
This property lets us determine the object or application that caused an error, and returns a string that contains its **Class** name or programmatic ID.
Syntax: **object.Source [= string]**

METHODS

❖ Clear **Method**
This method is used to clear the settings of the Error object.
Syntax: **object.Clear**

❖ Raise **Method**
This method is used to generate a VBScript runtime error.
Syntax: **object. Raise (number[, source, description, helpfile, helpcontext])**

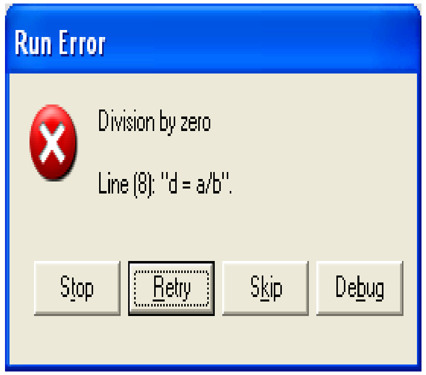
VB Script Fundamentals – Exception Handling: Code Snippets

```
' DATABASE HANDLING – USAGE OF ERROR.NUMBER & ERROR. DESCRIPTION
' USAGE OF On Error Resume Next
Const adOpenDynamic = 2
Const adLockOptimistic = 3
Const adCmdTableDirect = 512
Set cn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.Recordset")
Set errorObject = CreateObject("ADODB.Error")
On Error Resume Next
cn.Provider = "cap.localprovider.1"
cn.Properties("Data Source") = "c:\testdata"
cn.Open rs.Open "lostDataset", cn, adOpenDynamic, adLockOptimistic,
adCmdTableDirect DisplayErrorInfo
rs.Close
cn.Close

Sub DisplayErrorInfo()
For Each errorObject In rs.ActiveConnection.Errors
     Msgbox "Description: " & errorObject.Description & Chr(10) & Chr(13) & _
                    "Number: " & Hex(errorObject.Number)
      Next
End Sub
```

```
'USAGE OF ON ERROR  STATEMENT….
On Error GOTO 0
'Late Binding - Purposely added for invoking error

Dim a
Dim b
a = 90
b =0
d = a/b
If err.number <> 0 then
 Msgbox "Error Number is :- " & Cstr(Err.number) &
vbCrLf & "Error Description is :- " & Err.Description
end If
```

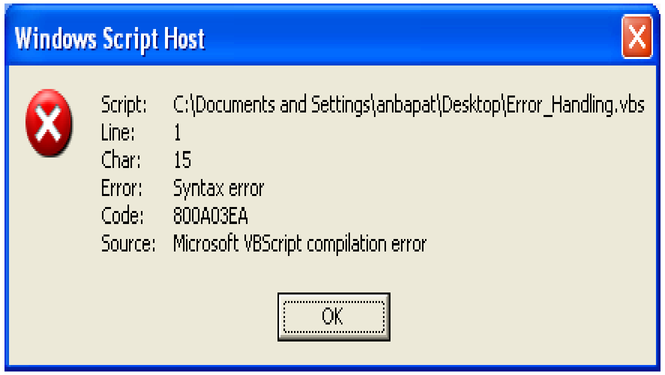
```
'USAGE OF HelpFile  & HelpContext Properties….

On Error Resume Next
Dim Msg Err.Clear
Err.Raise 6 ' Generate "Overflow" error.
Err.Helpfile = "yourHelp.hlp"
Err.HelpContext = yourContextID

If Err.Number <> 0 Then
      Msg = "Press F1 or Help to see " & Err.Helpfile & "
topic for" & _ " the following HelpContext: " &
Err.HelpContext
Msgbox Msg, , "error: " & Err.Description, Err.Helpfile,
      Err.HelpContext
End If
```

Error Message Snapshots

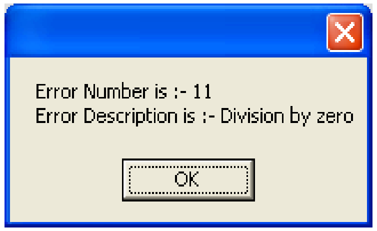

Run Error

Division by zero

Line (8): "d = a/b".


Windows Script Host

```
Script:   C:\Documents and Settings\anbapat\Desktop\Error_Handling.vbs
Line:     1
Char:     15
Error:    Syntax error
Code:     800A03EA
Source:   Microsoft VBScript compilation error
```


Error Number is :- 11
Error Description is :- Division by zero

VB Script Advanced – Regular Expression

What is Regular Expression?
It is a way of representing data using symbols. They are often used within matching, searching or replacing algorithms.
Regular Expressions in QTP - Regular expressions can be used in QTP for identifying objects and text strings with varying values.

Where we use:
• Defining the property values of an object in Descriptive programming for handling dynamic objects
• For parameterizing a step
• creating checkpoints with varying values

Using Regular Expressions in QTP:
We can define a regular expression for a constant value, a Data Table parameter value, an Environment parameter value, or a property value in Descriptive programming.
We can define a regular expression in standard checkpoint to verify the property values of an object; we can set the expected value of an object's property as a regular expression so that an object with a varying value can be verified.
We can define the text string as a regular expression, when creating a text checkpoint to check that a varying text string is displayed on our application,
For XML checkpoints we can set attribute or element values as regular expressions.

Ways of Regular Expressions:

a) Backslash Character:

- A backslash (\) can serve two purposes. It can be used in conjunction with a special character to indicate that the next character be treated as a literal character.
- Alternatively, if the backslash (\) is used in conjunction with some characters that would otherwise be treated as literal characters, such as the letters n, t, w, or d, the combination indicates a special character.

b) Matching Any Single Character:
A period (.) instructs QTP to search for any single character (except for \n).
Example : -
welcome. - Matches welcomes, welcomed, or welcome followed by a space or any other single character.

c) Matching Any Single Character in a List:
Square brackets instruct QTP to search for any single character within a list of characters.
Example : -
To search for the date 1867, 1868, or 1869, enter: **186[789]**

VB Script Advanced – Regular Expressions

❖**Matching Any Single Character within a Range:** To match a single character within a range, we can use square brackets ([ ]) with the hyphen (-) character.

For Example: For matching any year in the 2010s, enter: 201[0-9]

❖**Matching Zero or More Specific Characters:** An asterisk (*) instructs QTP to match zero or more occurrences of the preceding character.

For example: ca*r - Matches car, caaaaaar, and cr

❖**Matching One or More Specific Characters:** A plus sign (+) instructs QTP to match one or more occurrences of the preceding character.

For example: ca+r Matches car and caaaaaar, but not cr

❖**Matching Zero or One Specific Character:** A question mark (?) instructs QTP to match zero or one occurrences of the preceding character.

For example: ca?r Matches car and cr, but nothing else

VB Script Advanced – Regular Expressions

- **Grouping Regular Expressions:** Parentheses (()) instruct QTP to treat the contained sequence as a unit, just as in mathematics and programming languages. Using groups is especially useful for delimiting the argument(s) to an alternation operator (|) or a repetition operator (* , + , ? , { })

- **Matching One of Several Regular Expressions:** A vertical line (|) instructs QTP to match one of a choice of expressions.

- **Matching the Beginning of a Line:** A caret (^) instructs QTP to match the expression only at the start of a line, or after a newline character.

- **Matching the End of a Line:** A dollar sign ($) instructs QTP to match the expression only at the end of a line, or before a newline character.

- **Matching Any AlphaNumeric Character Including the Underscore:** \w instructs QTP to match any alphanumeric character and the underscore (A-Z, a-z, 0-9, _).

- **Matching Any Non-AlphaNumeric Character**: \W instructs QTP to match any character other than alphanumeric characters and underscores.

- **Combining Regular Expression Operators:** We can combine regular expression operators in a single expression to achieve the exact search criteria we need.
  For example, start.* - Matches start, started, starting, starter, and so forth.
  We can use a combination of brackets and an asterisk to limit the search to a combination of non-numeric characters.
  For example: [a-zA-Z]*

- To match any number between 0 and 1200, we need to match numbers with 1 digit, 2 digits, 3 digits, or 4 digits between 1000-1200.
  The regular expression below matches any number between 0 and 1200:
  - For Example - ([0-9]?[0-9]?[0-9]|1[01][0-9][0-9]|1200)

VB Script Advanced – Regular Expressions

RegExp object - VB Script is providing RegExp object for defining Regular expressions, It provides simple support for defining regular expressions.
Regular Expression Object Properties and Methods:

Properties:
a) Global Property
b) IgnoreCase Property
c) Pattern Property

Methods:
a) Execute Method
b) Replace Method
c) Test Method

Regular Expressions Examples:
1) Match File Names in a Directory against Regular Expression

```
Set objFS = CreateObject("Scripting.FileSystemObject")
Set objShell = CreateObject("WScript.Shell")
strCurrentDirectory = objShell.CurrentDirectory
Set objFolder = objFS.GetFolder(strCurrentDirectory)
Set colFiles = objFolder.Files

Set objRE = New RegExp
objRE.Global     = True
objRE.IgnoreCase = False
objRE.Pattern    = WScript.Arguments(0)
For Each objFile In colFiles
  bMatch = objRE.Test(objFile.Name)
  If bMatch Then
    WScript.Echo objFile.Name
  End If
Next
```

2) **Match Content in a File against a Regular Expression**

```
strFileName = "E:\testing.txt"
Set objFS = CreateObject("Scripting.FileSystemObject")
Set objTS = objFS.OpenTextFile(strFileName)

strFileContents = objTS.ReadAll

WScript.Echo "Searching Within: "
WScript.Echo strFileContents

objTS.Close

Set objRE = New RegExp
objRE.Global     = True
objRE.IgnoreCase = False
objRE.Pattern    = WScript.Arguments(0)

Set colMatches = objRE.Execute(strFileContents)
WScript.Echo vbNewLine & "Resulting Matches:"

For Each objMatch In colMatches
  WScript.Echo "At position " & objMatch.FirstIndex & " matched
" & objMatch.Value
Next
```

VB Script Advanced – Dictionary Object

Dictionary Object stores data in " (KEY–ITEM) " Pair format

A Dictionary object is the equivalent of a PERL "Associative array/Hash Variable". 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 an integer or a string, but can be anything except an array.

Creating a Dictionary Object:
Set objDictionary = CreateObject("Scripting.Dictionary")

Dictionary Objects Methods:

- **Add Method** - Adds a key and item pair to a Dictionary object
- **Exists Method -** Returns true if a specified key exists in the Dictionary object, false if it does not.
- **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.
- **Remove Method -** Removes a key, item pair from a Dictionary object.
- **RemoveAll Method -** The RemoveAll method removes all key, item pairs from a Dictionary object.

Example:
```
Dim cities
Set cities = CreateObject("Scripting.Dictionary")
cities.Add “1", “Pune"
cities.Add “2", “Rajkot"
cities.Add “3", “Noida“
```

Dictionary Objects Properties:
- **Count Property** - Returns the number of items in a collection or Dictionary object. Read-only.
- **CompareMode Property** - Sets and returns the comparison mode for comparing string keys in a Dictionary object.
- **Key Property** - Sets a key in a Dictionary object.
- **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.

VB Script Advanced – Dictionary Object

Examples:

1) Add Elements to a Dictionary

```
Set objDictionary = CreateObject("Scripting.Dictionary")
objDictionary.Add "Printer 1", "Printing"
objDictionary.Add "Printer 2", "Offline"
objDictionary.Add "Printer 3", "Printing"
```

2) **Delete All Elements from a Dictionary**

```
Set objDictionary = CreateObject("Scripting.Dictionary")
objDictionary.Add "Printer 1", "Printing"
objDictionary.Add "Printer 2", "Offline"
objDictionary.Add "Printer 3", "Printing"
colKeys = objDictionary.Keys          ‘ It forms collection of all the available keys within mentioned Dictionary

Wscript.Echo "First run: "
For Each strKey in colKeys
    Wscript.Echo strKey
Next
objDictionary.RemoveAll
colKeys = objDictionary.Keys
Wscript.Echo VbCrLf & "Second run: "
For Each strKey in colKeys
    Wscript.Echo strKey
Next
```

3) **Delete One Element from a Dictionary**

```
Set objDictionary = CreateObject("Scripting.Dictionary")
objDictionary.Add "Printer 1", “Testing Printer - 1"
objDictionary.Add "Printer 2", “Testing Printer - 2"
objDictionary.Add "Printer 3", “Testing Printer - 3“
```

VB Script Advanced – Dictionary Object

3) Delete One Element from a Dictionary

```
Set objDictionary = CreateObject("Scripting.Dictionary")
objDictionary.Add "Printer 1", "Printing"
objDictionary.Add "Printer 2", "Offline"
objDictionary.Add "Printer 3", "Printing“
colKeys = objDictionary.Keys
Wscript.Echo "First run: "
For Each strKey in colKeys
   Wscript.Echo strKey
Next
objDictionary.Remove("Printer 2")
colKeys = objDictionary.Keys
Wscript.Echo VbCrLf & "Second run: "
For Each strKey in colKeys
   Wscript.Echo strKey
Next
```

4) List the Number of Items in a Dictionary

```
Set  objDictionary  =  CreateObject("Scripting.Dictionary")
objDictionary.Add  "Printer  1",  "Printing"
objDictionary.Add  "Printer  2",  "Offline"
objDictionary.Add  "Printer  3",  "Printing"
Wscript.Echo  objDictionary.Count
```

5) Verify the Existence of a Dictionary Key

```
Set objDictionary = CreateObject("Scripting.Dictionary")
objDictionary.Add "Printer 1", "Printing"
objDictionary.Add "Printer 2", "Offline"
objDictionary.Add "Printer 3", "Printing“

If objDictionary.Exists("Printer 4") Then
   Wscript.Echo "Printer 4 is in the Dictionary."
Else
   Wscript.Echo "Printer 4 is not in the Dictionary."
End If
```

VB Script Advanced – Database Connectivity

Basic usage - Microsoft's ActiveX Data Objects (ADO) is a set of Component Object Model (COM) objects for accessing data sources. A part of MDAC, it provides a middleware layer between programming languages and OLE DB (a means of accessing data stores, whether they be databases or otherwise, in a uniform manner). ADO allows a developer to write programs that access data without knowing how the database is implemented. You must be aware of your database for connection only. No knowledge of SQL is required to access a database when using ADO, although one can use ADO to execute SQL commands. The disadvantage of this (i.e. using SQL directly) is that it introduces a dependency upon the type of database used.

Some basic steps are required in order to be able to access and manipulate data using ADO :
❖Create a connection object to connect to the database.
❖Create a recordset object in order to receive data in.
❖Open the connection
❖Populate the recordset by opening it and passing the desired table name or SQL statement as a parameter to *open* function.
❖Do all the desired searching/processing on the fetched data.
❖Commit the changes you made to the data (if any) by using *Update* or *UpdateBatch* methods.
❖Close the recordset
❖Close the connection

ADO collections

Fields - This collection contains a set of Field objects. The Collection can be used in either a Recordset object or in a Record object. In a Recordset object, each of the Field objects that make up the Fields collection corresponds to a column in that Record set object. In a Record object, a Field can be an absolute or relative URL that points into a tree-structured namespace (used for semi-structured data providers like the Microsoft OLE DB Provider for Internet Publishing) or as a reference to the default Stream object associated with that Record object.

Properties - An object can have more than one Property object, which are contained in the object's Properties collection.

Parameters - A Command object can have several Parameter commands to change its predefined behaviour, and each of the Parameter objects are contained in the Command object's Parameters collection

Errors - All provider created errors are passed to a collection of Error objects, while the Errors collection itself is contained in a Connection object. When an ADO operation creates an error, the collection is cleared and a new group of Error objects are created in the collection.

VB Script Advanced – Database Connectivity

ADO objects

Connection - The connection object is ADO's connection to a data store via OLE DB. The connection object stores information about the session and provides methods of connecting to the data store. As some data stores have different methods of establishing a connection, some methods may not be supported in the connection object for particular OLE DB providers. A connection object connects to the data store using its 'Open' method with a connection string which specifies the connection as a list of key value pairs (for example: "Provider='SQLOLEDB';Data Source='TheSqlServer'; Initial Catalog='Northwind';Integrated Security='SSPI';"). The start of which must identify the type of data store connection that the connection object requires: an OLE DB provider (for example SQLOLEDB), using the syntax "provider=";
a file name, using the syntax "file name=";
a remote provider and server (see RDS), using the syntax "Remote provider=" and "Remote server="; or
an absolute URL, using the syntax "URL="

Command - After the connection object establishes a session to the data source, instructions are sent to the data provider via the command object. The command object can send SQL queries directly to the provider through the use of the CommandText property, send a parameterised query or stored procedure through the use of a Parameter object or Parameters collection or run a query and return the results to a dataset object via the Execute method. There are several other methods that can be used in the Command object relating to other objects, such as the Stream, RecordSet or Connection objects.

Recordset - A recordset is a group of records, and can either come from a base table or as the result of a query to the table. The RecordSet object contains a Fields collection and a Properties collection. The Fields collection is a set of Field objects, which are the corresponding columns in the table. The Properties collection is a set of Property objects, which defines a particular functionality of an OLE DB provider. The RecordSet has numerous methods and properties for examining the data that exists within it. Records can be updated in the recordset by changing the values in the record and then calling on the Update or UpdateBatch method. Adding new records is performed through the AddNew function and then by calling on the Update or UpdateBatch method. Records are also deleted in the recordset with the Delete method and then by calling on the Update method. However, if for some reason the deletion cannot occur, such as because of violations in referential integrity, then the recordset will remain in edit mode after the call to the Update method. The programmer must explicitly call on the CancelUpdate function to cancel the update. Additionally, ADO can rollback transactions (if this is supported) and cancel batch updates. Recordsets can also be updated in one of three ways: via an immediate update, via a batch update, or through the use of transactions:

Immediate - The recordset is locked using the adLockOptimistic or adLockPessimistic lock. The data are updated at the data source after the record is changed and the Update method is called.

VB Script Advanced – Database Connectivity

Batch - The recordset is locked using adLockBatchOptimistic and each time Update is called the data are updated in a temporary buffer. Finally, when UpdateBatch is called the data are completely updated back at the data source. This has the advantage of it all being done in memory, and if a problem occurs then UpdateCancel is called and the updates are not sent to the data source

Transaction - If the OLE DB provider allows it, transactions can be used. To start the transaction, the programmer invokes the BeginTrans method and does the required updates. When they are all done, the programmer invokes the CommitTrans method. RollbackTrans can be invoked to cancel any changes made inside the transaction and rollback the database to the state before the transaction began

Record This object represents one record in the database and contains a fields collection. A RecordSet consists of a collection of Record objects.

Stream - A stream, mainly used in a RecordSet object, is a means of reading and writing a stream of bytes. It is mostly used to save a recordset in an XML format, to send commands to an OLE DB provider as an alternative to the CommandText object and to contain the contents of a binary or text file.

Parameter - A parameter is a means of altering the behaviour of a common piece of functionality, for instance a stored procedure might have different parameters passed to it depending on what needs to be done; these are called parameterised commands.

Field - Each Record object contains many fields, and a RecordSet object has a corresponding Field object also. The RecordSet object's Field object corresponds to a column in the database table that it references.

Property -This object is specific to the OLE DB provider and defines an ability that the provider has implemented. A property object can be either a built-in property — it is a well-defined property implemented by ADO already and thus cannot be altered — or can be a dynamic property — defined by the underlying data provider and can be changed

Error - When an OLE DB provider error occurs during the use of ADO, an Error object will be created in the Errors collection. Other errors do not go into an Error object, however. For instance, any errors that occur when manipulating data in a RecordSet or Field object are stored in a Status property.

VB Script Advanced – Database Connectivity

```
Dim cn, rs ,fout, i, line
Const Server = "yourServer"
Const DB = "yourDatabase"
Const Uid = "yourUsername"
Const Pwd = "yourPassword"
Const Proc = "yourProcedure"
Const outFile = "out.csv“

'Open output file
With CreateObject("Scripting.FileSystemObject")
    Set fout = .OpenTextFile(outFile, 2, True)
End With


'Connect to database
Set cn = CreateObject("ADODB.Connection")
With cn
    .Provider = "SQLNCLI"
    .ConnectionString = "Server=" & Server & ";Database=" & DB
&  ";Uid=" & Uid & ";Pwd=" & Pwd & ";“
    .CursorLocation = 3
    .Open
End With
```

```
'Run procedure
    Set rs = CreateObject("ADODB.Recordset")
    rs.Open Proc, cn, 0, 1, 4
    'Dump results to file
    Do Until rs.EOF
        line = ""
        For i = 0 To rs.Fields.Count - 1
            If Not IsNull(rs.Fields(i).Value) Then
                line = line & ","""" & rs.Fields(i).Value & """"
             Else
                line = line & ","
             End If
        Next 'i
        fout.WriteLine Mid(line, 2)
        rs.MoveNext
    Loop

    'Clean up
    out "#[EoS]#"
    rs.Close
    cn.Close
    fout.Close

Sub out(s)
    On Error Resume Next
    WScript.StdOut.WriteLine s
End Sub
```

VB Script Fundamentals - Arrays

An array is a special type of variable which stores a set of related values.
An array can be static or dynamic, one dimension wide or multidimensional.

Arrays are a very useful and easy way of storing variables – and especially easy to use in VBScript due to several factors --

- VBScript particularly liberal with any variable definition means no strict defining of variables to a particular data is type
- the data type is assigned automatically when the variable is loaded with a value
- it's even possible to mix data types within the same array

It is also possible to define the arrays in different ways, for example --

  - create the array element by element
  - use the VBScript array method
  - use the VBScript split method

- Creating a Simple VBScript Array

  In this case an array with three elements (0, 1 and 2) has been defined; and then it's just a matter of assigning values to the elements --

```
Dim Country(2)
Country(0)="India"
Country(1)="US"
Country(2)="UK”
```

Note that base index value for VBScript array is zero(0).

The VBScript Array Method of Creating Arrays

VBScript has its own built in method for creating arrays in bulk rather than having to do it element by element --

```
dim country
country = array(“India", “US", “UK")
```

The VBScript **Split** Method of Creating Arrays

VBScript can also create arrays from information such as csv (comma separated variable) data --

```
dim country, details
details = “India", “US", “UK "
country = split (details, ",")
```

VB Script Advanced- Multidimensional Arrays

***Arrays aren't limited to a single dimension. You can have as many as 60 dimensions*.**

As the amount of information increases (for example storing an age as well as a name) then you can use a multidimensional array where each dimension represents a different aspect of the data.

```
Dim person(2,1)
```

Would create a 2 column, 3 row 2D array

.

Name	Age
0,0 => Tom	1,0 =>20
0,1 => Dick	1,1 =>25
0,2 => Harry	1,2 =>30

This is about Static array i.e. fixed size array.

You can also declare an array whose size changes during run time. It is called a dynamic array.

Dynamic array is initially declared using either the Dim or ReDim statement and subsequently redefined using ReDim to determine the number of dimensions and the size of each dimension.

```
Dim MyArray(25) . . .
ReDim Preserve MyArray(30)
```

To preserve the contents of the array as the resizing takes place, keyword 'Preserve' is used.

There is no limit to the number of times you can resize a dynamic array, although if you make it smaller, you will lose the data in the eliminated elements.

- **Array Functions and statement**

UBound(<array>,<dimension>) – gives upper bound of an array

LBound(<array>,<dimension>) -- returns lower bound of array

IsArray(<array>) -- Returns a Boolean value indicating whether a variable is an array.

Erase Statement - Erase <Array> - Reinitializes the elements of fixed-size arrays & deallocates dynamic-array storage space.

VB Script Advanced – File System Object [FSO]

.Many a times during testing you may need to interact with Drives, folder and text files using QTP. Interaction can be(but not limited to) in the form of reading input from a file, writing output to a file. The FileSystemObject (FSO) provides an API to access the Windows filesystem, providing access to files, drives, text streams

As shown in figure below ,FSO object model contains the following objects and collections.

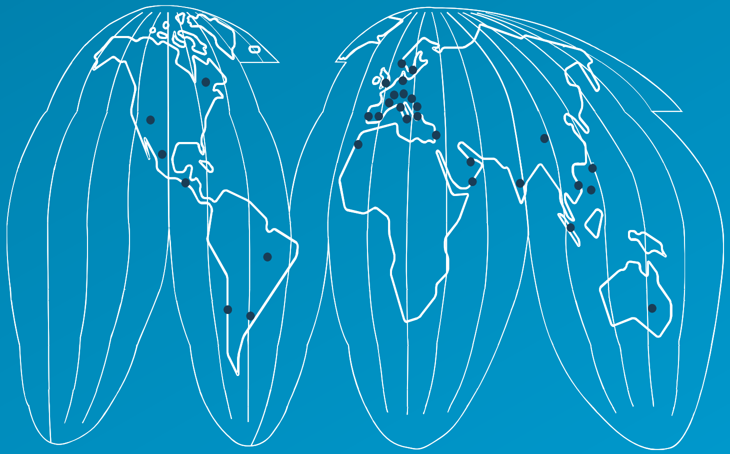
FileSystemObject	- Properties - Methods	Part of VBScript Run-Time Library. Allows creating, deleting, manipulating and getting status information on files, folders and drives,
Drives Collection Object	- Properties	Retrieves information about a drive collection
Drive Object	- Properties	Retrieves information about a drive
Folders Collection Object	- Properties - Methods	A list of all subfolders in a given folder, or folders on a drive.
Folder Object	- Properties - Methods	Allows creating, deleting, and moving of folders. Can also be used to get information about a folder
Files Collection Object	- Properties	A list of all files in a given folder
File Object	- Properties - Methods	Allows creating, deleting, and moving of files. Can also be used to retrieve file properties.
TextStream Object	- Properties - Methods	Allows reading from and writing to text files.

People matter, results count.

About Capgemini

With more than 120,000 people in 40 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2011 global revenues of EUR 9.7 billion.
Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini



www.capgemini.com