UNIT 3

What is VBScript?

As the name itself explains, **VBScript is a ‘Scripting Language’**. It is a lightweight case insensitive programming language developed by Microsoft. It is a subset of ‘Visual Basic’ or we may also say it as a lighter version of Microsoft’s programming language Visual Basic.

Most of us would have used Visual Basic during our course curriculum in our school or college. Visual Basic is an event-driven programming language and an Integrated Development Environment from Microsoft.

VBScript language is used in QTP for coding and running Automated Test Scripts. This is not a very difficult language to learn and with a little knowledge of basic programming skills and passion for writing code, anyone can learn this easily. For those who know Visual Basic, it is an added advantage.

Automation Testers, who want to create, maintain and execute the tests in QTP need to have basic programming skills using VBScript.

Basic of VB Scripting Concepts

***Now let’s move on to some basics topics that are revolving around VBScript to enable clear understanding and knowledge about VBScript.***

Data types

1) There is only one data type: **Variant**. It can store different kinds of information based on the context in which it is used.
2) If used in a numeric context it is a number or a string if used in a string expression.
3) If a number has to behave as a string we could enclose it within “ “.
4) There are various subtypes to a variant. You can explicitly specify these subtypes to achieve a clear definition for your data. The below is a screenshot from the VB User guide that shows all the subtypes of data that can be used:
*(click on image to enlarge)*

5) Conversion functions can be used to convert one subtype of data into another.
6) Since it is the only data type available, all the return values from a function are variants.

Here are different VBScripting [examples](http://www.w3schools.com/vbscript/vbscript_examples.asp) you can try on your own.

**Variables**

**1)** A variable is nothing but a space in the computer’s memory that can store certain information. This information is bound to change from time to time. Where the information goes physically is immaterial but when needed, it can be accessed or changed by addressing the name of the variable.

E.g: If there is a statement that you want to run several times, you could use a variable to contain that count. Say X. X is a variable that can be used to store, change and use the space in the memory where we want to keep the count.

**2)** All variables are of the datatype Variant.

**3)** Declaring a variable before its use is optional, although it’s a good practice to do so.

**4)** To make the declaration mandatory there is an “**Option Explicit”**Statement available. To declare variables:

Dim x – This declares  x
Dim x, y, z – This declares multiple variables
X=10 – This is how a value is assigned. As a general rule, the variable is the left-hand side component and the right is its value.
X=”Swati” – this is the way a string value is assigned.

To make declarations mandatory this is how the code has to be written:
*Option Explicit*
*Dim x, stri*

If Option explicit statement was not used, we could have directly written:
*x=100*
*stri=”Swati”*
and it would not have thrown an error.

**5)** **Naming convention**: Names must start with an alphabetic character, must be unique, cannot contain an embedded period and cannot exceed 255 chars.

**6)** A variable containing a single value is a scalar variable and the one that has more than one is an array.

**7)** A one dimensional Array can be declared as Dim A(10). All the arrays in VB Script are zero-based that means the array index starts from 0 through the number declared. That means, our array A has 11 elements. Starting from 0 to 10.

**8)** To declare a 2-dimensional array simply separate the row count and column count by a comma. Eg: Dim A(5, 3). This means it has 6 rows and 4 columns. The first number is always row and the second a comma.

**9)** There is also a dynamic array whose size can change during runtime.  These arrays can be declared using dim or redim statements.

If an array is declared as Dim A(10) and during runtime, if we need more space we can do the same by using the statement: redim A(10). There is a “Preserve” statement that can be used in conjunction with the redim statement.

Dim A(10,10)
……
….
Redim preserve A(10,20)

This piece of code shows how we do it. Initially, A is a 11 by 11 array. Then we are resizing it to be an 11 by 21 array and the preserve statement will make sure that the data that is previously contained in the array is not lost.

**Constants**

1. As the name implies a constant is nothing but an unchanging value in a program that is assigned a name.
2. They can be declared by prefixing “Const” to a name.
3. Eg: Const a=”10” or Const Astr=”Swati”.
4. This value cannot be changed accidentally while the script is running.

**Operators**

Some of the important operators that are most commonly used are:

1. String concatenation: & (Eg: Dim x=”good”&”day”, so x contains “goodday”
2. Addition (+)
3. Subtraction (-)
4. Multiplication (\*)
5. Division(/)
6. Logical negation (Not)
7. Logical conjunction (And)
8. Logical disjunction ( Or)
9. Equality(=)
10. Inequality (<>)
11. Less than (<)
12. Greater than(>)
13. Less than or equal to(<=)
14. Greater than or equal to (>=)
15. Object equivalence(Is)

It is important to note that the list is not complete but merely a subset containing the most commonly used operators.

**The operator precedence rules are:**

1. Multiplication or Division take precedence over addition or subtraction
2. If multiplication and division exist in the same expression, then left to right order is considered
3. If Addition and subtraction occur in the same expression, then too, left and right order is taken into consideration.
4. The order can be overridden by using parenthesis. In this case, the expression within the parenthesis is executed first.
5. & operator takes precedence after all arithmetic operators and before all logical operators.

Environments Supporting VBScript

Primarily, there are 3 Environments where VBScript can be run.

**They include:**

**#1) IIS (Internet Information Server):** **I**nternet **I**nformation **S**erver is Microsoft's Web Server.

**#2) WSH (Windows Script Host): W**indows **S**cript **H**ost is the hosting environment of the Windows Operating System.

**#3) IE (Internet Explorer): I**nternet **E**xplorer is a simple hosting environment that is most frequently used to run scripts.

Data Types in VBScript

Unlike other languages, VBScript has only 1 data type called **Variant**.

As this is the only data type that is used in VBScript, it’s the only data type that is returned by all the functions in the VBScript.

A variant data type can contain different kinds of information, depending on how it is used. ***For Example,*** If we use this data type in String context then this will behave like a String and if we use this in the Numeric context then this will behave like a Number. This is the specialty of a Variant data type.

A Variant data type can contain several subtypes. Now, let's take a look at what all values/data will be returned if a particular subtype is used.

**Subtypes include:**

**#1) Empty:** This subtype indicates that the value will be 0 in case of Numeric Variables and “” for String Variables.

**#2) Null:**This subtype indicates that there is no valid data.

**#3) Boolean:**This subtype indicates that the resultant value will be either true or false.

**#4) Byte:**This subtype exhibits that the resultant value will lie in the range between 0 to 255 i.e. the result will be from any value ranging from 0 to 255.

**#5) Integer:**This subtype shows that the resultant value will lie in the range between -32768 to 32767 i.e. the result will be from any value ranging from -32768 to 32767

**#6) Currency:**This subtype indicates that the resultant value will lie in the range between -922,337,203,685,477.5808 to 922,337,203,685,477.5807 i.e. the result will be from any value ranging from -327-922,337,203,685,477.5808 to 922,337,203,685,477.5807.

**#7) Long:**This subtype shows that the resultant value will lie in the range from -2,147,483,648 to 2,147,483,647 i.e. result will be from any value in between -2,147,483,648 to 2,147,483,647.

**#8) Single:**This subtype exhibits that the resultant value will be from any value in between -3.402823E38 to -1.401298E-45 in case of negative values.

And for positive values, the result will be from any value in between 1.401298E-45 to 3.402823E38.

**#9) Double:**This subtype indicates that the resultant value will be from any value in between -1.79769313486232E308 to 4.94065645841247E-324 in case of negative values.

And for positive values, the result will be from any value in between 4.94065645841247E-324 to 1.79769313486232E308.

**#10) Date (Time):**This subtype will return a number which will represent a date value in between January 1, 100 to December 31, 9999

**#11) String:**This subtype will return a variable-length string value which can approximately be up to 2 billion characters in length.

**#12) Object:**This subtype will return an object.

**#13) Error:**This subtype will return an error number.

How to Create a Simple VBScript?

To create a VBScript, there are only 2 things required.

**They are:**

* **Text Editors** like Notepad++ or even Notepad to write the VBScript Code.
* **IE** (good to have IE6 or above) to run the VBScript Code.

Now, let’s see a few VBScript Code for clarity purpose but before that, it is important to know where can the Scripts be inserted in an HTML Page.

Where to Insert Scripts in an HTML Page?

VBScript provides you the liberty to place a code in any of the following sections:

* **Within the Header Tags** i.e. in between <head> and </head>.
* **Within the Document’s Body** i.e. between <body> and </body> tags.

**First VBScript Code in HTML:**

***Now, let’s take a simple example to understand how VBScript code can be written inside HTML tags.***

<html>

<head>

<title> Testing VBScript Skills </title>

</head>

<body>

<script type=”text/vbscript”>

variable1 = 1

variable2 = 2

output = (variable1 + variable2) / 1

document.write (“resultant from the above equation is ” & output)

</script>

</body>

</html>

**Note**: Whatever is put inside the brackets of ‘document.write’, will be displayed as an output on the display page.

**The Output** of this program is: **resultant from the above equation is 3**

After completing the code, you can save this in a file and give a file name as anyfilename.html.

**To run**, just open this file in IE.

**Important to Know:**

We have just seen the implementation of VBScript code in the HTML file. **However, VBScript in QTP is not placed inside the HTML tags. It is saved with an extension ‘.vbs’ and is executed by the QTP Execution Engine.**

To understand the practical implementation of VBScript in terms of QTP, you must know variables, constants, etc. and I will cover that in my upcoming tutorials and for time being, I just want to show you the VBScript code with the concept of an external file.

**VBScript in External File:**

<html>

<head>

<script type=”text/vbscript” src=”nameofthefile.vbs”> </script>

</head>

<body>

variable1 = 22

variable2 = 21

subtraction = variable1 - variable2

document.write (“subtraction of 2 numbers is” & subtraction)

</body>

</html>

**To access this code from an external source, save this code in a text file with an extension “.vbs”.**

How Comments are Handled in VBScript

It is considered to be a good programming practice to include comments in the Scripts for better readability and understanding purposes.

**There are 2 ways in which Comments can be handled in a VBScript:**

**#1) Any Statement that starts with a Single Quote (‘) is treated as a comment:**

<html>

<head>

<script type=”text/vbscript”</script>

</head>

<body>

‘let’s do subtraction of 2 numbers

variable1 = 11

variable2 = 10

subtraction = variable1 - variable2

document.write (“subtraction of 2 numbers is” & subtraction)

</body>

</html

**#2)** **Any Statements that start with the keyword REM are treated as Comments.**

<html>

<head>

<script type=”text/vbscript”></script>

</head>

<body>

**REM let’s do subtraction of 2 numbers**

variable1 = 11

variable2 = 10

subtraction = variable1 - variable2

document.write (“subtraction of 2 numbers is” & subtraction)

</body>

</html>

**Formatting Tips:**

**#1)** No Semicolon is required to end the particular statement in VBScript.

**#2)** If 2 or more lines are written in the same line in VBScript then**Colons (:) act as a line separator**.

**Let’s understand this with the help of an Example:**

<html>

<head>

<script type=”text/vbscript”></script>

</head>

<body>

**variable1 = 11:variable2 = 21:variable3=34**

</body>

</html>

 **#3)** If a statement is lengthy and required to break into multiple statements then you can use **underscore** “\_”.

***Let’s see its Example:***

<html>

<head>

<script type=”text/vbscript”></script>

</head>

<body>

variable1 = 11

variable2 = 10

output = (variable1 - variable2) \* 10

document.write (“output generated from the calculation”& \_

“of using variable1 and variable2 with the multiplication of resultant”&\_

from 10 is” & output)

</body>

</html>

Reserved Keywords

In any language, there are a set of words which work as Reserved Words and they cannot be used as Variable names, Constant names or any other Identifier names.

**Following is the list of Reserved Keywords in VBScript:**

| **Loop** | **New** | **Null** | **ParamArray** |
| --- | --- | --- | --- |
| LSet | Next | On | Preserve |
| Me | Nothing | Option | Private |
| Mod | Not | Optional | Public |
| RaiseEvent | ReDim | Dem | Rem |
| Resume | RSet | Select | Set |
| Shared | Single | Static | Stop |
| Sub | Then | To | True |
| Type | And | As | Boolean |
| Case | Class | Const | Currency |
| Debug | Dim | Do | Double |
| Each | Else | Elself | Empty |
| Event | Exit | False | For |
| Function | GoTo | If | Imp |
| Implements | In | Integer | Is |
| Until | Variant | Wend | While |
| With | Xor | Eval | Execute |
| MsgBox | Erase | ExecuteGlobal | Option |
| OptionExplicit | Randomize | SendKeys | Let |
| Let | Like | Long | Type |
| End | EndIf | Enum | Eqv |

Conditional Statements

**#1) If Statement**

*If (condition)..Then*
*{Statement or a block of statement}*
*Else*
*{Statement or a block of statement}*
*End if*



This is the typical syntax to write this statement.

* The if..then..else statements can be nested to any number of levels.
* Alternately, the else block can be used or not.
* Elseif is another variation that can be used when choosing one of the multiple options.

*if x=0 then*
*Msgbox “value=0”*
*Elseif x=1 then msgbox “value=1”*
*Elseif x=2 then msgbox “value=2”*
*Else msgbox “value not found”*
*End if*

**#2) Select Statement**

This is to choose one out of the many options depending on the condition that comes satisfied. The condition is evaluated once and based on the value it attains one of the following blocks of code gets chosen to be run.

*Select Case (expression)*
*Case “case1”*
*{Block 1}*
*Case “case 2”*
*{Block 2}*
*…..*
*Case Else*
*{Else block}*
*End Select*

Looping Statements

*There are 4 kinds of loop statements:*

**#1) Do…loop**
This is used when a statement or a block of statements need to be executed while or until a said condition is true. Let us first  look at the syntax of this:

***Syntax 1:***
*Do (While | Until) condition*
*{Statement or statements}*
*[Exit Do]*
*{Statement or statements}*
*Loop*

***Syntax 2:***
*Do*
*{statement or statements]*
*[Exit Do]*
*{statement or statements]*
*Loop [{While | Until} condition]*

**Condition**: could be a numeric or string component that either attains a true or false value. If the condition is null it is treated as False.
Observe the ‘Exit do’ in the above.

There is also a slight difference between syntax 1 and syntax 2.

In the case of syntax 1, the statements within the do loop do not get executed unless the condition becomes true or holds true.

For syntax 2, the statements within the loop get executed at least once and then the check is performed on the condition.

**Exit Do:**In order to avoid infinite loops we will have to force the loop to exit. Exit Do is a statement that is used in such circumstances.

**#2) While…wend**
***Syntax:***

*While*
*{Statement or block of statement}*
*Wend*

It is self-explanatory from the syntax that the statements nestled under the while block get executed as long as the condition holds true.

Although this statement is available, it is not very flexible so it is recommended to use the Do…Loop statement.

**#3) For…Next**
This is the statement that you would use when you want a statement/set of statements to run a certain number of times while a certain counter gets incremented or decremented.

*For counter = start To end [Step step]*
*[statements]*
*[Exit For]*
*[statements]*
*Next*

* As you can see from the syntax above, there is a ‘Step’ clause to this statement. This clause is optional.
* The step clause if not specified, the for loop steps one counter forward by default
* The step can be used to increment or decrement the counter value
* Exit For is similar to “Exit Do”, it can be used to come out of the For block and execute the statement that follows.
* Any number of Exit For statements can be used within one block of For statement
* It is usually used with an if..then statement to make sure some condition that would lead to infinitely looping is true and in case if it does, the For statement has a way to exit.
* Any number of For statements can be nested within a For.

***Example:***

*For i=1 to 10*
*……*
*If x=true then*
*……*
*Exit for*
*End if*
*Next*

***Example of a positive step:***
*For i = 2 To 12 Step 2*
*total = total + k*
*Next*

***Example of a negative step:***
*For i = 12 To 2 Step -2*
*total = total + k*
*Next*

**#4) For each…next**

This is similar to ‘For …next’. This is used for collection objects or arrays. This statement runs a statement or a set of statements for every object or item in an array instead of the number of times specified. As the ‘For…next’ statement Exit is used to exit before the looping is through, like in case of an error. Also, any number of For Each statement can be nested within each other.
***Syntax:***

*For Each element In group*
*[statements]*
*[Exit For]*
*[statements]*
*Next [element]*

* The element is the variable that is used to iterate through the elements in the array or collection object
* the group stands for the name of the collection object or array

**Note:** We have not discussed collection objects so far in our series, but a collection object is nothing but an object which is a set of related items (objects, these might be of the same type or could be of different types)

Best Practices For Code Writing In VB Script

* At the beginning of every program, write down a brief description of what the program does
* Provide comments for every variable used during its declaration to briefly describe what this variable is going to do
* Keep the code modular, as much as possible
* Have the main program readable and have all the logic segregated in terms of function, so it makes them easy to read and maintain.
* The segregation of code into functions will also increase reusability
* An agreed naming convention has to be used to maintain consistency
* Comments – Provide comments to make the code more understandable
* Indentation – Make sure that you indent the lines of code to clearly understand the sequence of execution
* Option Explicit is to be declared so that you don’t run into an issue when you have a spelling mistake in a variable name
* Watch out for infinite loops

### What is an Array?

**An Array** is a variable having named memory location which serves as a **Container** and can hold multiple values in a single location.

In short, Arrays group different type of elements together in one place.

Let’s take a real-life **example** to understand this better. If you want to store the names of different students at a single place then Array of string type can be used starting at index 0. If you want to fetch the name of the first student then you can pick the value present at index 0 and so on.

Now, let’s move ahead to the next topics to learn how Arrays are actually declared and used in the script.

### Declaration of Arrays in VBScript

Declaration of an Array can be done in the same manner in which Variables are declared but with the difference that array variable is declared by using **parenthesis** ‘()’.

**The Dim** keyword is used to declare an Array.

**Ways to declare an Array:**

There are 3 ways in which an Array can be declared.

**They are as follows:**

#### #1) Way 1: Dim array1()

Here, array1 is the name of an array and as parenthesis is empty it means that the size of an array is not defined here.

If you want to declare an array by mentioning its size then it can be done in the following way.

#### #2) Way 2: Dim array1(5)

In this, array1 is declared with the size as 5 which states it holds 6 values considering that the index of an array always starts from 0. These 5 values can be of integer type, string or character types.

#### #3) Way 3 : array1 = Array(1,2,3,4,5,6)

Here, Array Function is used to declare an array with a list of arguments inside the parenthesis and all integer values are passed directly inside the parenthesis without any need of mentioning the size of an array.

**Note**: Index value of an Array can never be a negative value.

Next, let’s discuss how to assign values to an array.

### Assignment of Values Inside an Array

Once an Array is declared, values are assigned to an Array Variable. To assign values, each specific index location is accessed as values are assigned specifically to the index values in an Array.

Taking the reference to the second way of declaring an array (as discussed above), let’s see how to assign values to such an array.

**Dim array1(5)**

Here, the size of an array is 6, which means you have to assign 6 values to an array starting at index 0 and ending at 5.

**So, following is the way to do so:**

array1(0) = “hello”
array1(1) = 12
array1(2) = 13
array1(3) = 14
array1(4) = 15
array1(5) = 16

Each index has one specific value.

**Following is an example to show the usage of an Array:**

<html>

<head>

<title>Let’s see implementation of an Array</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Dim array1(5)

array1(0) = “hello”

array1(1) = 12

array1(2) = 13

array1(3) = “how are you”

array1(4) = 15

array1(5) = 16

For i = 0 to ubound(array1)

Msgbox “Value present at index ” & i & ” is “ & array1(i) & “<br />”

Next

</script>

</body>

</html>

**Output is:**

Value present at index 0 is hello
Value present at index 1 is 12
Value present at index 2 is 13
Value present at index 3 is how are you
Value present at index 4 is 15
Value present at index 5 is 16

In the above example, an array of size ‘5’ is declared and the values are assigned to each index which is the combination of integer and string values. Next, using ‘For loop’, the value present at each index is displayed with the help of a message box. Loop will start from 0 and will go till the unbound i.e. upper bound which is the maximum subscript(5 in this case) of an array.

I will discuss unbound later in this tutorial.



### Types of Arrays

There are basically 2 types of Arrays that are used in the VBScript.

**They are:**

#### #1) Single Dimensional Array:

This is a simple type of array which is used more often in the scripts, the one which is discussed above

#### #2) Multi-Dimensional Array:

When an array has more than 1 dimension then it is known as a multi-dimensional array. Normally, a **2-Dimensional Array** is the one which is used most of the times i.e. there will be rows and columns in an array. The maximum dimension of an array can reach up to 60.

***Let’s understand the working of a 2 Dimensional Array with the help of a simple example.***

***Example:***

<html>

<head>

<title>Let’s see implementation of a 2 Dimensional Array</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Dim array1(1,1)

array1(0,0) = “hello”

array1(0,1) = 12

array1(1,0) = “how are you”

array1(1,1) = 14

Msgbox “Value present at index 0,0” & ” is “ & array1(0,0) & “<br />”

Msgbox “Value present at index 0,1” & ” is “ & array1(0,1) & “<br />”

Msgbox “Value present at index 1,0” & ” is “ & array1(1,0) & “<br />”

Msgbox “Value present at index 1,1” & ” is “ & array1(1,1)

</script>

</body>

</html>

**Output is:**

Value present at index 0,0 is hello
Value present at index 0,1 is 12
Value present at index 1,0 is how are you
Value present at index 1,1 is 14

In the above example, an array having 2 rows and 2 columns are declared with the size as (1,1) representing values present at the indexes 0 and 1 for both row and the column.



Next, let’s understand some of the frequently used concepts of an Array.

### Usage of REDIM Statement and PRESERVE Keyword in an Array

**Redim**Statement is used to re-define the size of an Array. When the array is declared without any size, then it can be declared again using Redim with the feasibility of specifying the size of an array.

**Preserve** keyword is used to preserve the contents of a current array when the size of an array gets changed.

***Let’s understand the usage of these keywords with the help of a simple example.***

***Example:***

<html>

<head>

<title>Let’s see implementation of Redim and Preserve</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Dim array1()

REDIM array1(3)

array1(0) = “hello”

array1(1) = 12

array1(2) = 13

array1(3) = “how are you”

REDIM PRESERVE array1(5)

array1(4) = 15

array1(5) = 16

For i = 0 to ubound(array1)

Msgbox “Value present at index ” & i & ” is “ & array1(i) & “<br />”

Next

</script>

</body>

</html>

### In-Built Array Functions

There are various inbuilt functions related to an Arras that is supported by the VBScript.

**Following is the list:**

**#1) lbound**:

This is the opposite of ubound (used above). This returns the smallest integer index value of an array i.e. the smallest subscript of an array.

**Example:** In the above example, the size of an array is 5. Hence, lbound will be 0 as this is the smallest subscript of an array.

**#2) ubound**:

This is already used above. This returns the largest subscript of a defined array.

**Example:**In the above example, the size of an array is 5. Hence, in this case, unbound is 5

**#3) Split**:

This returns an array consisting of a number of sub-strings and can be split using some delimiter. The syntax of this is: Split(expression,[delimiter])

Using a delimiter is an optional condition.

**#4) Join**:

This is the opposite of the Split function. Here, String is returned which includes various substrings in an array and thus joins all the sub-strings into one string.

The **syntax** of this is: Join(array,[delimiter]. Using a delimiter is an optional condition.

**#5) IsArray**:

This returns True/False on the basis of a specified variable. If the variable is passed is an Array then True is returned else False.

The **syntax** is: IsArray( array variable)

**#6) Filter**:

This returns a subset of an array based on the filter condition i.e. data is filtered on the basis of some condition.

The**syntax** is: Filter(array, filter condition)

***Let’s see the implementation of these functions with the help of a simple Example.***

***Example:***

<html>

<head>

<title>Let’s see implementation of In-Built Array Functions</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Dim array1 = Array(“January”, ”February”, ”March”, ”April”)

Dim a , b , c , d , e , f

a = lbound(array1)

b = ubound(array1)

c = Split(array1,”,”)

d = Join(array1,” $ “)

e = IsArray(array1)

f = Filter(array1,”J”)

Msgbox(a) & “<br />”

Msgbox(b) & “<br />”

Msgbox(c) & “<br />”

Msgbox(d) & “<br />”

Msgbox(e) & “<br />”

Msgbox(f)

</script>

</body>

</html>

**Output** **is**:

### 03January February March AprilJanuary $ February $ March $ AprilTrueOverview of Functions And Procedures

When a need arises to accomplish a particular task then a piece of code can be written including several statements under a named section inside a program which is known as **Function/Procedure** in terms of programming language.

Functions and Procedures are mainly used to provide assistance for arranging the code in a program in an organizable way. Using Function/Procedure, the same code can be used multiple times by calling them and this reduces the pain of writing the same code again and again at the Script level.

This is a good programming practice to follow some guidelines while creating the scripts and by working with the functions and procedures you can ensure this to happen. Instead of writing the code in a linear manner in the scripts, it is advisable to divide the scripts into Procedures to provide better readability and understanding of the Scripts.

Dividing the scripts is an important part while dealing with the creation of frameworks in QTP/UFT. As they help in making the code manageable, it is easy to debug and less complex to use.

This is just an overview of Functions and Procedures.

Let’s move on to the next topics to gain more knowledge about Functions and Procedures.

Types of Procedures in the VBScript

Basically, there are two different types of Procedures in the VBScript.

* **Sub Procedures**
* **Function Procedures**

Both are same in usage as both work for providing reusability of code but with few differences, let’s discuss each of them in detail along with some examples.

#1) Sub Procedures

This is a type of procedure which includes a set of statements inside the block of code and after execution, it **does not return** any value.

This can be defined by making use of keywords like ‘Sub’ and ‘End Sub’ respectively. It may or may not take an input.

This can be parameterized i.e. can take arguments inside the brackets if required.

***Below is an Example to show the usage of Sub Procedure:***

<html>

<head>

<title>Let’s see implementation of Sub Procedure</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Sub Returnvalue()

 Msgbox “Let’s get back to work!!”

End Sub

Call Returnvalue()

</script>

</body>

</html>

**The output** **is**: Let’s get back to work!!

In the above example, Sub Procedure is created without arguments and is called in a program using **‘Call’** keyword following the name of the Sub procedure. If you do not want to use ‘Call’ keyword while calling Sub procedure then you can simply call it by just writing the name of the Sub Procedure without making use of the Call keyword.



Next, let’s see about Function Procedures.

#2) Function Procedures

This is a type of procedure which includes a set of statements inside the block of the code and after execution **may return** value also. This can take an input if required, depending upon the situations.

This can be defined by making use of keywords like ‘Function’ and ‘End Function’ respectively.

Function Procedures when used with arguments i.e. by taking parameters inside the brackets, will **return a value** in such a case.

***Below is an Example to show the usage of Function Procedure:***

<html>

<head>

<title>Let’s see implementation of Function Procedure</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Function addition(a,b)

 Dim result

 result = a + b

 addition = result ‘storing value of result in the name of the function

End Function

Dim a, b, output

a= 10

b=20

output = addition(10,20)

Msgbox(output)

</script>

</body>

</html>

**The Output** **is**: 30

In the above example, Function Procedure is created using arguments and is called in a program by using the name of the Function procedure. This is the case when the value is returned. Value of a resultant from the function is stored in another variable ‘output’ and its value is displayed in the message box.

Function Procedure without argument works in the same way as shown in the Example of Sub Procedure.



Let’s summarize few basic differences between these 2 types of procedures.

Difference Between Sub Procedure and Function Procedure

**The Differences include:**

* Sub Procedure never takes an input while Function Procedure may take an input if required.
* Sub Procedure starts and ends with using Sub and End Sub respectively while Function Procedure starts and ends with Function and End Function respectively.
* The most **important** difference is Sub Procedure never returns a value while the Function Procedure may return a value (as shown above).

Ways to Pass a Value in the Function

There are 2 ways to pass a value in the function.

**They are:**

#1) Pass by Value:

While passing an argument, the Changes that take place in the called procedure and if the value of a variable does not remain to persist, then it means it is passing by value.

Keyword used in this case is **ByVal**.

***Let’s understand this with the help of a simple Example:***

<html>

<head>

<title>Let’s see implementation of Pass by Value </title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Function output1(ByVal a)

 Dim result

 result = a / 5

 output1 = result ‘storing value of result in the name of the function

End Function

Dim a, val

a= 10

val = output1(10)

Msgbox(val)

</script>

</body>

</html>

**Output** **is:**10



#2) Pass by Reference:

While passing an argument, the Changes that take place in the called procedure and if the value of a variable remains to persist then it means it is passing by reference.

Keyword used in this case is **ByRef.**

***Let’s understand this with the help of a simple Example:***

<html>

<head>

<title>Let’s see implementation of Pass by Reference</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Function output1(ByRef a)

 Dim result

 result = a / 5

 output1 = result ‘storing value of result in the name of the function

End Function

Dim a, val

a= 10

val = output1(10)

Msgbox(val)

</script>

</body>

</html>

**Output is:** 2



There are several types of In-Built Functions in the VBScript.

Let’s have a look at them in detail.

Inbuilt Functions in the VBScript

Some important inbuilt functions that can be used directly in the script are explained below in detail.

#1) Format Functions

There are various inbuilt functions available for formatting the expression and they can be used directly in the script without any need for writing the code explicitly before using them in the script.

* **FormatCurrency:**This is used to convert a specified expression in the form of a Currency value.
* **FormatDateTime:**This is used to convert a specified expression in the form of a Date and Time value.
* **FormatNumber:**This is the most **important** and widely used Format Function and is used to convert the specified expression in the form of a Number.
* **FormatPercent:**This is used to convert the specified expression in the form of a Percentage value.

#2) Math and Conversion Functions

There are various inbuilt functions for performing mathematical operations and conversion purposes and can be used directly in the script without any need of writing the code explicitly before using them in the script.

* **Abs**: This is the **frequently** used Math function for the purpose of returning the Absolute value for a related number which is specified as a parameter.
* **Int:** This is Math Function and is used in those cases where it is required to fetch an integer part from a given expression/ number.
* **cDate**: This is one of the **frequently** used Conversion Function for converting the expression which includes the Date or Time parameters into Date subtype.
* **cStr:** This is the Conversion function which is used for converting the expression into String subtype.

***Let’s see a simple example using these inbuilt functions***

<html>

<head>

<title>Let’s see implementation of Inbuilt Functions</title>

</head>

<body>

<script language=”vbscript” type=”text/vbscript”>

Dim a , b

a = Abs (-10)

b = CStr(#10-10-17#)

Msgbox(a)

Msgbox(b)

</script>

</body>

</html>

**Output** **is**:

10

UNIT 4

What are Web Services?

Different books and different organizations provide different definitions to Web Services. Some of them are listed here.

* A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.
* Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
* Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
* A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

To summarize, a complete web service is, therefore, any service that −

* Is available over the Internet or private (intranet) networks
* Uses a standardized XML messaging system
* Is not tied to any one operating system or programming language
* Is self-describing via a common XML grammar
* Is discoverable via a simple find mechanism

Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components −

* SOAP (Simple Object Access Protocol)
* UDDI (Universal Description, Discovery and Integration)
* WSDL (Web Services Description Language)

All these components have been discussed in the [Web Services Architecture](https://www.tutorialspoint.com/webservices/web_services_architecture.htm) chapter.

How Does a Web Service Work?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −

* XML to tag the data
* SOAP to transfer a message
* WSDL to describe the availability of service.

You can build a Java-based web service on Solaris that is accessible from your Visual Basic program that runs on Windows.

You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

Example

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

The steps to perform this operation are as follows −

* The client program bundles the account registration information into a SOAP message.
* This SOAP message is sent to the web service as the body of an HTTP POST request.
* The web service unpacks the SOAP request and converts it into a command that the application can understand.
* The application processes the information as required and responds with a new unique account number for that customer.
* Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
* The client program unpacks the SOAP message to obtain the results of the account registration process

Purpose

Different books and different organizations provide different definitions to Web Services. Some of them are listed here.

* A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.
* Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
* Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
* A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

To summarize, a complete web service is, therefore, any service that −

* Is available over the Internet or private (intranet) networks
* Uses a standardized XML messaging system
* Is not tied to any one operating system or programming language
* Is self-describing via a common XML grammar
* Is discoverable via a simple find mechanism

Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components −

* SOAP (Simple Object Access Protocol)
* UDDI (Universal Description, Discovery and Integration)
* WSDL (Web Services Description Language)

All these components have been discussed in the [Web Services Architecture](https://www.tutorialspoint.com/webservices/web_services_architecture.htm) chapter.

How Does a Web Service Work?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −

* XML to tag the data
* SOAP to transfer a message
* WSDL to describe the availability of service.

You can build a Java-based web service on Solaris that is accessible from your Visual Basic program that runs on Windows.

You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

Example

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

The steps to perform this operation are as follows −

* The client program bundles the account registration information into a SOAP message.
* This SOAP message is sent to the web service as the body of an HTTP POST request.
* The web service unpacks the SOAP request and converts it into a command that the application can understand.
* The application processes the information as required and responds with a new unique account number for that customer.
* Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
* The client program unpacks the SOAP message to obtain the results of the account registration process.

Web Services - Components

Over the past few years, three primary technologies have emerged as worldwide standards that make up the core of today's web services technology. These technologies are discussed below.

SOAP

SOAP is an XML-based protocol for exchanging information between computers.

* SOAP is a communication protocol.
* SOAP is for communication between applications.
* SOAP is a format for sending messages.
* SOAP is designed to communicate via Internet.
* SOAP is platform independent.
* SOAP is language independent.
* SOAP is simple and extensible.
* SOAP allows you to get around firewalls.
* SOAP will be developed as a W3C standard.

To learn more about SOAP, visit our [SOAP Tutorial](https://www.tutorialspoint.com/soap/index.htm).

WSDL

WSDL is an XML-based language for describing web services and how to access them.

* WSDL stands for Web Services Description Language.
* WSDL was developed jointly by Microsoft and IBM.
* WSDL is an XML based protocol for information exchange in decentralized and distributed environments.
* WSDL is the standard format for describing a web service.
* WSDL definition describes how to access a web service and what operations it will perform.
* WSDL is a language for describing how to interface with XML-based services.
* WSDL is an integral part of UDDI, an XML-based worldwide business registry.
* WSDL is the language that UDDI uses.
* WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'.

**What is Restful Web Service?**

REST is used to build Web services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a RESTful service. The underlying protocol for REST is HTTP, which is the basic web protocol. REST stands for REpresentational State Transfer

**RESTful Key Elements**

Web services have really come a long way since its inception. In 2002, the Web consortium had released the definition of WSDL and SOAP web services. This formed the standard of how web services are implemented.

In 2004, the web consortium also released the definition of an additional standard called RESTful. Over the past couple of years, this standard has become quite popular. And is being used by many of the popular websites around the world which include Facebook and Twitter.

REST is a way to access resources which lie in a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of resources. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST defines a way on how these resources can be accessed.

The key elements of a RESTful implementation are as follows:

1. **Resources** – The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is **http://demo.guru99.com**. Now in order to access an employee record resource via REST, one can issue the command **http://demo.guru99.com/employee/1** - This command tells the web server to please provide the details of the employee whose employee number is 1.
2. **Request Verbs** - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example **http://demo.guru99.com/employee/1** , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.
3. **Request Headers** – These are additional instructions sent with the request. These might define the type of response required or the authorization details.
4. **Request Body** - Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web service. In a POST call, the client actually tells the web service that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.
5. **Response Body** – This is the main body of the response. So in our example, if we were to query the web server via the request **http://demo.guru99.com/employee/1** , the web server might return an XML document with all the details of the employee in the Response Body.
6. **Response Status codes** – These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

**Restful Methods**

The below diagram shows mostly all the verbs (POST, GET, PUT, and DELETE) and an example of what they would mean.

Let's assume that we have a RESTful web service is defined at the location. **http://demo.guru99.com/employee** . When the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent by the client.

1. **POST** – This would be used to create a new employee using the RESTful web service
2. **GET** - This would be used to get a list of all employee using the RESTful web service
3. **PUT** - This would be used to update all employee using the RESTful web service
4. **DELETE** - This would be used to delete all employee using the RESTful web service

Let's take a look from a perspective of just a single record. Let's say there was an employee record with the employee number of 1.

The following actions would have their respective meanings.

1. **POST** – This would not be applicable since we are fetching data of employee 1 which is already created.
2. **GET** - This would be used to get the details of the employee with Employee no as 1 using the RESTful web service
3. **PUT** - This would be used to update the details of the employee with Employee no as 1 using the RESTful web service
4. **DELETE** - This is used to delete the details of the employee with Employee no as 1



**Why Restful**

Restful mostly came into popularity due to the following reasons:

1. Heterogeneous languages and environments – This is one of the fundamental reasons which is the same as we have seen for [SOAP](https://www.guru99.com/soap-simple-object-access-protocol.html) as well.
* It enables web applications that are built on various programming languages to communicate with each other
* With the help of Restful services, these web applications can reside on different environments, some could be on Windows, and others could be on Linux.

But in the end, no matter what the environment is, the end result should always be the same that they should be able to talk to each other. Restful web services offer this flexibility to applications built on various programming languages and platforms to talk to each other.

The below picture gives an example of a web application which has a requirement to talk to other applications such Facebook, Twitter, and Google.

Now if a client application had to work with sites such as Facebook, Twitter, etc. they would probably have to know what is the language Facebook, Google and Twitter are built on, and also on what platform they are built on. Based on this, we can write the interfacing code for our web application, but this could prove to be a nightmare.

Facebook, Twitter, and Google expose their functionality in the form of Restful web services. This allows any client application to call these web services via REST.



1. The event of Devices – Nowadays, everything needs to work on[Mobile](https://www.guru99.com/mobile-testing.html)devices, whether it be the mobile device, the notebooks, or even car systems.

Can you imagine the amount of effort to try and code applications on these devices to talk with normal web applications? Again Restful API's can make this job simpler because as mentioned in point no 1, you really don't need to know what is the underlying layer for the device.

1. Finally is the event of the Cloud – Everything is moving to the cloud. Applications are slowly moving to cloud-based systems such as in Azure or Amazon. Azure and Amazon provide a lot of API's based on the Restful architecture. Hence, applications now need to be developed in such a way that they are made compatible with the Cloud. So since all Cloud-based architectures work on the REST principle, it makes more sense for web services to be programmed on the REST based architecture to make the best use of Cloud-based services.

**Restful Architecture**

An application or architecture considered RESTful or REST-style has the following characteristics

1. State and functionality are divided into distributed resources – This means that every resource should be accessible via the normal HTTP commands of GET, POST, PUT, or DELETE. So if someone wanted to get a file from a server, they should be able to issue the GET request and get the file. If they want to put a file on the server, they should be able to either issue the POST or PUT request. And finally, if they wanted to delete a file from the server, they an issue the DELETE request.
2. The architecture is client/server, stateless, layered, and supports caching –
* Client-server is the typical architecture where the server can be the web server hosting the application, and the client can be as simple as the web browser.
* Stateless means that the state of the application is not maintained in REST.

For example, if you delete a resource from a server using the DELETE command, you cannot expect that delete information to be passed to the next request.

In order to ensure that the resource is deleted, you would need to issue the GET request. The GET request would be used to first get all the resources on the server. After which one would need to see if the resource was actually deleted.

**RESTFul Principles and Constraints**

The REST architecture is based on a few characteristics which are elaborated below. Any RESTful web service has to comply with the below characteristics in order for it to be called RESTful. These characteristics are also known as design principles which need to be followed when working with RESTful based services.

1. **RESTFul Client-Server**



This is the most fundamental requirement of a REST based architecture. It means that the server will have a RESTful web service which would provide the required functionality to the client. The client send's a request to the web service on the server. The server would either reject the request or comply and provide an adequate response to the client.

1. **Stateless**

The concept of stateless means that it's up to the client to ensure that all the required information is provided to the server. This is required so that server can process the response appropriately. The server should not maintain any sort of information between requests from the client. It's a very simple independent question-answer sequence. The client asks a question, the server answers it appropriately. The client will ask another question. The server will not remember the previous question-answer scenario and will need to answer the new question independently.

1. **Cache**



The Cache concept is to help with the problem of stateless which was described in the last point. Since each server client request is independent in nature, sometimes the client might ask the server for the same request again. This is even though it had already asked for it in the past. This request will go to the server, and the server will give a response. This increases the traffic across the network. The cache is a concept implemented on the client to store requests which have already been sent to the server. So if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information. This saves the amount of to and fro network traffic from the client to the server.

1. **Layered System**

The concept of a layered system is that any additional layer such as a middleware layer can be inserted between the client and the actual server hosting the RESTFul web service (The middleware layer is where all the business logic is created. This can be an extra service created with which the client could interact with before it makes a call to the web service.). But the introduction of this layer needs to be transparent so that it does not disturb the interaction between the client and the server.

1. **Interface/Uniform Contract**

This is the underlying technique of how RESTful web services should work. RESTful basically works on the HTTP web layer and uses the below key verbs to work with resources on the server

* POST - To create a resource on the server
* GET - To retrieve a resource from the server
* PUT - To change the state of a resource or to update it
* DELETE - To remove or delete a resource from the server

A SOAP message is an ordinary XML document containing the following elements −

* **Envelope** − Defines the start and the end of the message. It is a mandatory element.
* **Header** − Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point. It is an optional element.
* **Body** − Contains the XML data comprising the message being sent. It is a mandatory element.
* **Fault** − An optional Fault element that provides information about errors that occur while processing the message.

All these elements are declared in the default namespace for the SOAP envelope − <http://www.w3.org/2001/12/soap-envelope> and the default namespace for SOAP encoding and data types is − <http://www.w3.org/2001/12/soap-encoding>

**NOTE** − All these specifications are subject to change. So keep updating yourself with the latest specifications available on the W3 website.

## **SOAP Message Structure**

The following block depicts the general structure of a SOAP message −

<?xml version = "1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"

 SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

 <SOAP-ENV:Header>

 ...

 ...

 </SOAP-ENV:Header>

 <SOAP-ENV:Body>

 ...

 ...

 <SOAP-ENV:Fault>

 ...

 ...

 </SOAP-ENV:Fault>

 ...

 </SOAP-ENV:Body>

</SOAP\_ENV:Envelope>

# SOAP Vs. REST: Difference between Web API Services

The earlier tutorials have given a lot of details on 2 key types of web service designs. One is **SOAP** protocol (Simple Object Access Protocol) and the other being **REST** for Representational State Transfer.

Each technique has its own advantages and disadvantages. Hence, it's always good to understand in which situations each design should be used. This tutorial will go into some of the key differences between these techniques as well as what challenges you might encounter while using them.

## SOAP vs. REST

Let's have a short overview of soap vs rest before we do a deep dive into the key differences between them.

**SOAP** - SOAP is a protocol which was designed before REST and came into the picture. The main idea behind designing SOAP was to ensure that programs built on different platforms and programming languages could exchange data in an easy manner.

**REST** - This was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components.

Below are the main differences between SOAP and REST

|  |  |
| --- | --- |
| **SOAP** | **REST** |
| * SOAP stands for Simple Object Access Protocol
 | * REST stands for Representational State Transfer
 |
| * SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service.
 | * REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the constraints of being
	1. Client Server
	2. Stateless
	3. Cacheable
	4. Layered System
	5. Uniform Interface
 |
| * SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern.
 | * REST can make use of SOAP as the underlying protocol for web services, because in the end it is just an architectural pattern.
 |
| * SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer.
 | * REST use Uniform Service locators to access to the components on the hardware device. For example, if there is an object which represents the data of an employee hosted on a URL as http://demo.guru99 , the below are some of URI that can exist to access them

http://demo.guru99.com/Employeehttp://demo.guru99.com/Employee/1 |
| * SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.

<?xml version="1.0"?><SOAP-ENV:Envelope xmlns:SOAP-ENVs="http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodingStyle=" http://www.w3.org/2001/12/soap-encoding"><soap:Body> <Demo.guru99WebService xmlns="http://tempuri.org/"> <EmployeeID>int</EmployeeID> </Demo.guru99WebService> </soap:Body></SOAP-ENV:Envelope> | * REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.

{"city":"Mumbai","state":"Maharastra"} |
| * SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.
 | * REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.
 |

## When to use REST and when to use SOAP

One of the most highly debatable topics is when REST should be used or when to use SOAP while designing web services.

Below are some of the key factors that determine when each technology should be used for web services **REST services should be used in the following instances**

* **Limited resources and bandwidth** – Since SOAP messages are heavier in content and consume a far greater bandwidth, REST should be used in instances where network bandwidth is a constraint.
* **Statelessness** – If there is no need to maintain a state of information from one request to another then REST should be used. If you need a proper information flow wherein some information from one request needs to flow into another then SOAP is more suited for that purpose. We can take the example of any online purchasing site. These sites normally need the user first to add items which need to be purchased to a cart. All of the cart items are then transferred to the payment page in order to complete the purchase. This is an example of an application which needs the state feature. The state of the cart items needs to be transferred to the payment page for further processing.
* **Caching**– If there is a need to cache a lot of requests then REST is the perfect solution. At times, clients could request for the same resource multiple times. This can increase the number of requests which are sent to the server. By implementing a cache, the most frequent queries results can be stored in an intermediate location. So whenever the client requests for a resource, it will first check the cache. If the resources exist then, it will not proceed to the server. So caching can help in minimizing the amount of trips which are made to the web server.
* **Ease of coding**– Coding REST Services and subsequent implementation is far easier than SOAP. So if a quick win solution is required for web services, then REST is the way to go.

SOAP should be used in the following instances

1. **Asynchronous processing and subsequent invocation** – if there is a requirement that the client needs a guaranteed level of reliability and security then the new SOAP standard of SOAP 1.2 provides a lot of additional features, especially when it comes to security.
2. **A Formal means of communication** – if both the client and server have an agreement on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction. An example is an online purchasing site in which users add items to a cart before the payment is made. Let's assume we have a web service that does the final payment. There can be a firm agreement that the web service will only accept the cart item name, unit price, and quantity. If such a scenario exists then, it's always better to use the SOAP protocol.
3. **Stateful operations –**ifthe application has a requirement that state needs to be maintained from one request to another, then the SOAP 1.2 standard provides the WS\* structure to support such requirements.

## SOAP vs. REST API challenges

API is known as the **Application Programming Interface** and is offered by both the client and the server. In the client world, this is offered by the browser whereas in the server world it's what is provided by the web service which can either be SOAP or REST.

**Challenges with the SOAP API**

1. WSDL file - One of the key challenges of the SOAP API is the WSDL document itself. The WSDL document is what tells the client of all the operations that can be performed by the web service. The WSDL document will contain all information such as the data types being used in the SOAP messages and what all operations are available via the web service. The below code snippet is just part of a sample WSDL file.

<?xml version="1.0"?>

<definitions name="Tutorial"

 targetNamespace=http://demo.guru99.com/Tutorial.wsdl

 xmlns:tns=http://demo.guru99.com/Tutorial.wsdl

 xmlns:xsd1=http://demo.guru99.com/Tutorial.xsd

 xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/

 xmlns="http://schemas.xmlsoap.org/wsdl/">

 <types>

 <schema targetNamespace=http://Demo.guru99.com/Tutorial.xsd

 xmlns="http://www.w3.org/2000/10/XMLSchema">

 <element name="TutorialNameRequest">

 <complexType>

 <all>

 <element name="TutorialName" type="string"/>

 </all>

 </complexType>

 </element>

 <element name="TutorialIDRequest">

 <complexType>

 <all>

 <element name="TutorialID" type="number"/>

 </all>

 </complexType>

 </element>

 </schema>

</types>

As per the above WSDL file, we have an element called "TutorialName" which is of the type String which is part of the element TutorialNameRequest.

Now, suppose if the WSDL file were to change as per the business requirements and the TutorialName has to become TutorialDescription. This would mean that all the clients who are currently connecting to this web service would then need to make this corresponding change in their code to accommodate the change in the WSDL file.

This shows the biggest challenge of the WSDL file which is the tight contract between the client and the server and that one change could cause a large impact, on the whole, client applications.

1. Document size – The other key challenge is the size of the SOAP messages which get transferred from the client to the server. Because of the large messages, using SOAP in places where bandwidth is a constraint can be a big issue.

**Challenges with the REST API**

1. **Lack of Security**– REST does not impose any sort of security like SOAP. This is why REST is very appropriate for public available URL's, but when it comes down to confidential data being passed between the client and the server, REST is the worst mechanism to be used for web services.
2. **Lack of state**– Most web applications require a stateful mechanism. For example, if you had a purchasing site which had the mechanism of having a shopping cart, it is required to know the number of items in the shopping cart before the actual purchase is made. Unfortunately, the burden of maintaining this state lies with the client, which just makes the client application heavier and difficult to maintain.

## KEY DIFFERENCE

* SOAP stands for Simple Object Access Protocol whereas REST stands for Representational State Transfer.
* SOAP is a protocol whereas REST is an architectural pattern.
* SOAP uses service interfaces to expose its functionality to client applications while REST uses Uniform Service locators to access to the components on the hardware device.
* SOAP needs more bandwidth for its usage whereas REST doesn’t need much bandwidth.
* SOAP only works with XML formats whereas REST work with plain text, XML, HTML and JSON.
* SOAP cannot make use of REST whereas REST can make use of SOAP.