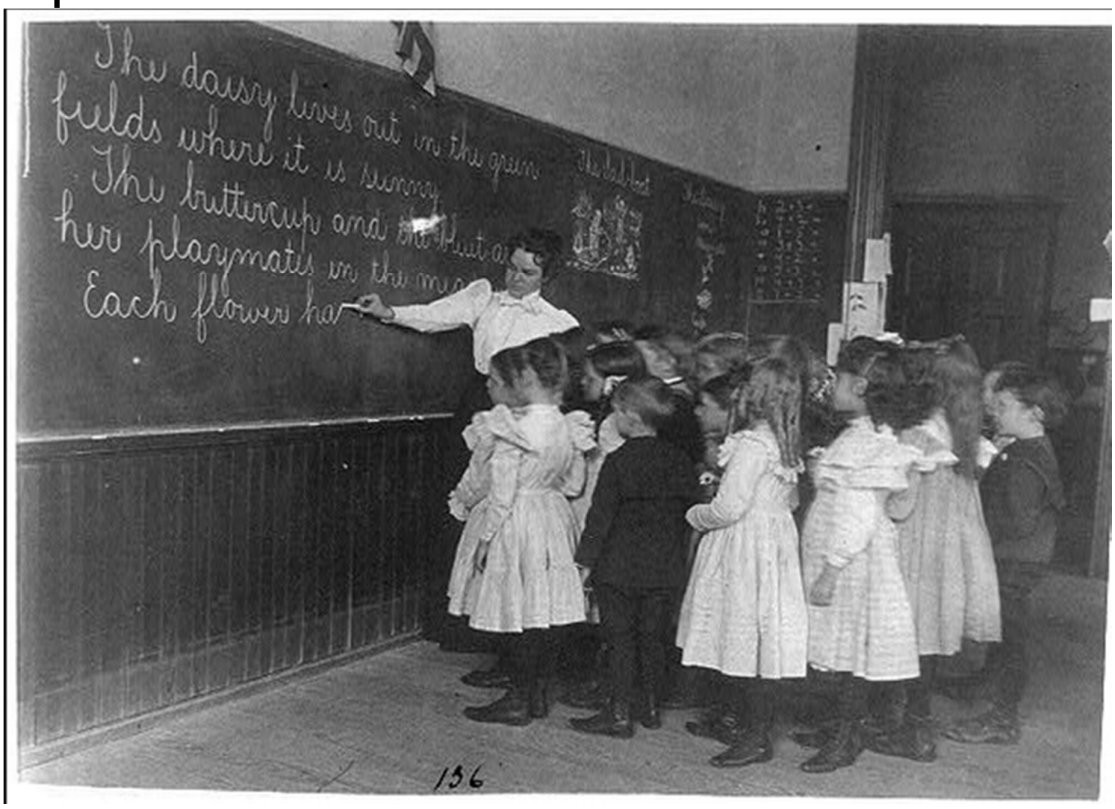
Better Coding - Performance Optimization



Step up the stairs or stare at the steps. - ***Ralph Nichols***

In this chapter we will discuss VBS code optimization and UFT performance tuning to automate test scripts efficiently.Primary objective of automation is to perform repetitive,unattended and bulk-processing tasks. Most of the users complain about the slow script execution,unresponsive system and runtime errors during automation execution.

Performance optimization is open-ended and context based quality attribute. It depends on many factors including hardware configuration,underlying technologies,optimum software configuration and usage of each component in the Infrastructure. So, Performance is related to address the bottleneck in overall system seeping out from code, infrastructure, network or component integration and configuration.

1.0 Lets Destroy the system - Importance of Performance

Lets write a VBS program to open a file,write the multiples of 5 and close the file.

```
	startTimer = Timer  ' Timer Returns the number of seconds elapsed since 12:00
AM
	Const ForWriting = 2
	Dim fso, MyFile
		For i = 1 To 10000 Step 1
			Set fso = CreateObject("Scripting.FileSystemObject")
			Set MyFile =fso.OpenTextFile("C:\Users\myFile.txt", ForWriting,
		True)
			MyFile.WriteLine i *5
			MyFile.Close
		Next
	endTimer = Timer
	MsgBox "Total Execution time" & ( endTimer  - startTimer) 'Time spent in
execution
```

Program: Prog 1.0 - Writing multiplier of 5 in a text file

You need not to worry about nitty-gritty of this code. We will cover Filesystem Object and other bits of this program later in this book.

Now, Lets see how this VBS program breaks the system. Fig 1.0 depicts the CPU spike and very high disk usage. System eventually becomes unresponsive during run time and lead to system failure.

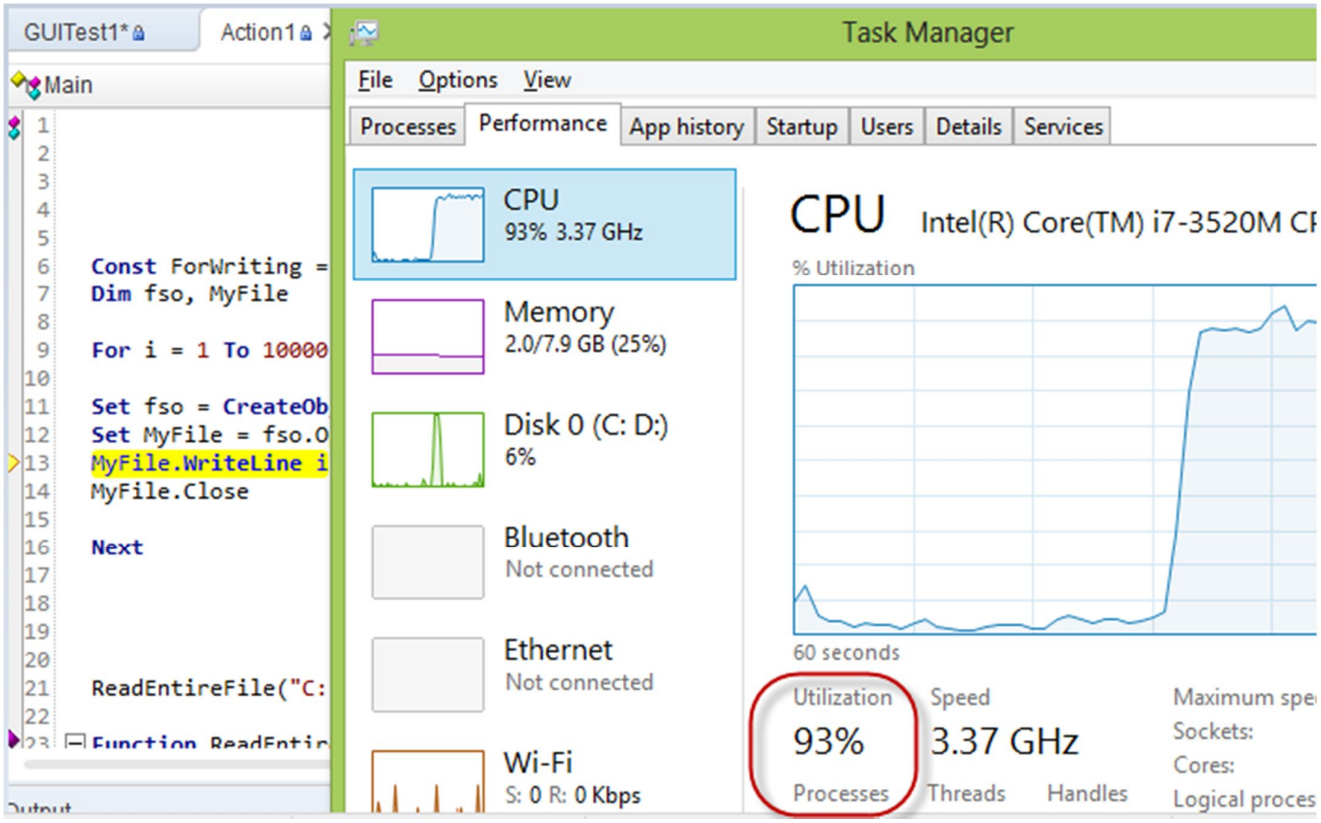


Figure 1-1 Script Execution and System Performance

Performance bottlenecks and failures will shoot-up maintenance cost and make automation,error prone and susceptible to deliver good return on investment.So, effectively, performance issues must address to realize the efficient test automation.

2.0 Performance Key Components

These are the four key component for finding any performance bottleneck in the system.

1. CPU
2. Memory
3. Network

4. Disk

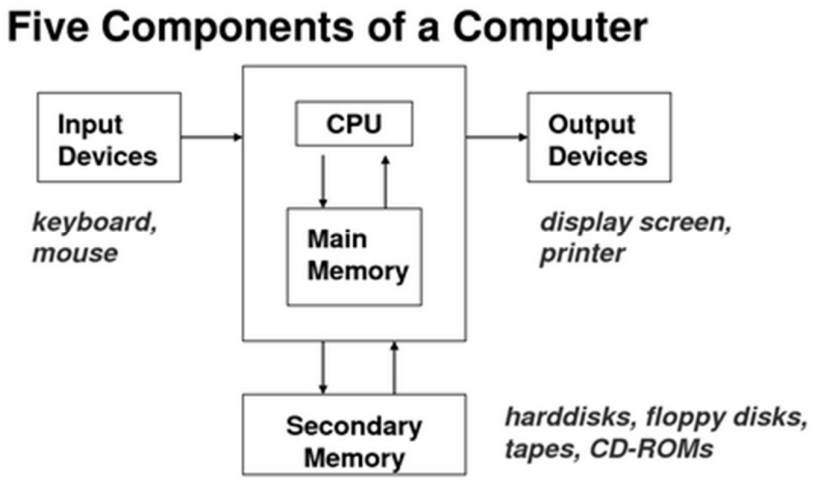


Figure 1-2 Performance Key Components

2.0.1 Program Execution Flow

VBScript program run by VBS engine/interpreter.Interpreter execute line by line statement.

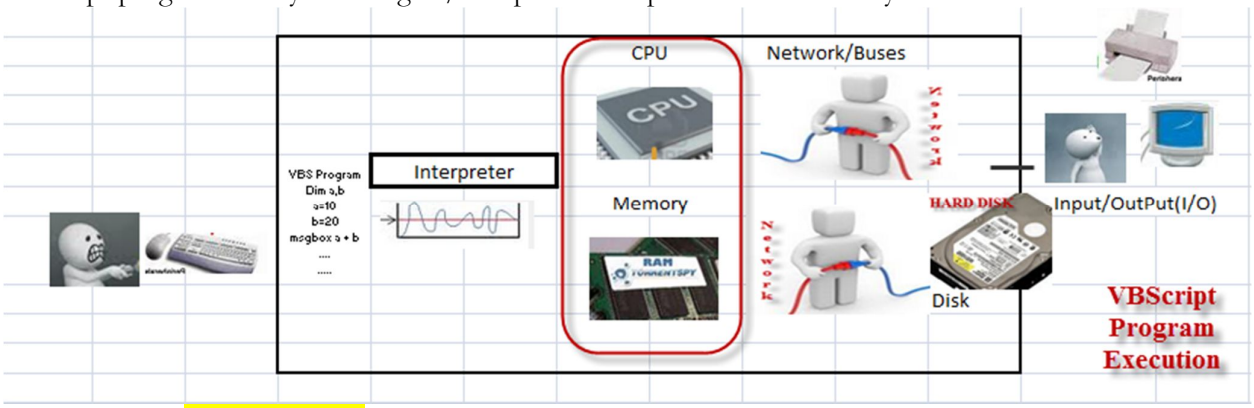


Figure 1-2-1 Program Execution Flow

Following are the major generic steps for executing any programs.

1.User runs the Program. VBScript engine (Interpreter*) starts
2. Program instructions and data placed into memory (RAM**).
3. CPU fetch each instruction from Memory provided by interpreter.
4. CPU Computes instruction (fetch is not same as compute). CPU interacts with other system components like Hard disk(via I/O Channel),memory and other device like Printer,I/O or display screen as instructed in code.

**ConfuSense: Interpreter picks line by line code and change it into machine language for processing.CPU does all the computation. Opposite to Interpreter, Compiler compiles whole program in one go and change it to machine(or Intermediate) language for processing.*

***ConfuSense: RAM(Primary/Main memory or 'Memory') contains program at runtime and Hard Disk (secondary memory or 'Disk') is a storage to keep our files and*

data.CPU runs the program from Main Memory only. ***Golden Rule****:*
High system performance translated by Fast CPU, Large RAM,optimize program,minimal I/O operations and their communication.
Resource Monitor provides the system state in relation of system components and running processes in the system

HandyBit*: Navigate to Start | Run | Taskmgr.exe | Resource Manager*

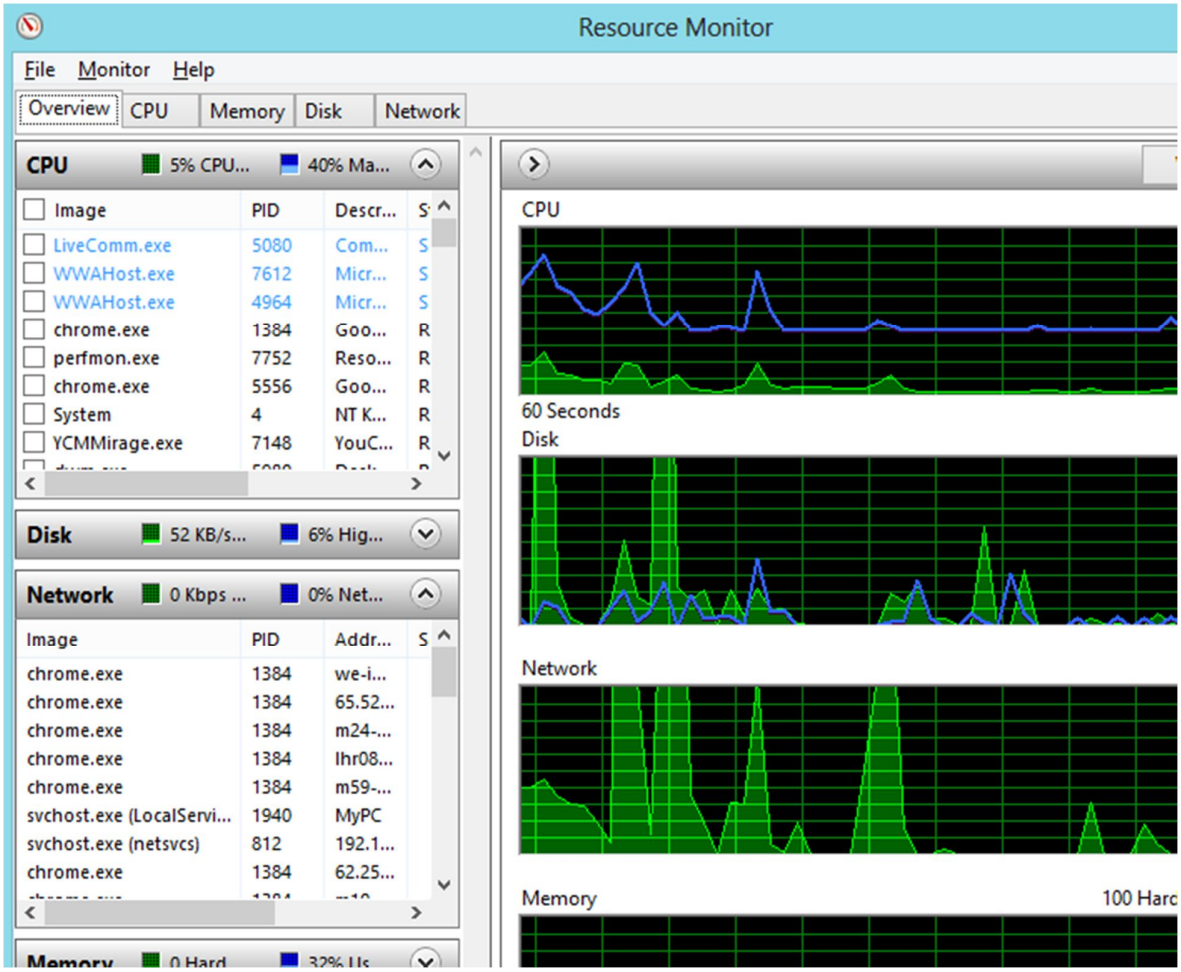


Figure 2-1 Resource Manager show system resources

3.0 Secret Ingredient of efficient Automation

Programmer should develop the program to use computing resources optimally.We will explore the areas where we can reduce the cost of computing. Our aim is to write the code to minimize CPU computing, lower the RAM usage and manipulation,CPU communication, paging,cache and other operating system activities. Disk and I/O operations and communication also impact performance of the system and should carefully use.

Better performance can be managed by considering better coding practices and best possible system and application (example UFT) configuration.Our focus is to learn VBScript program optimization and UFT configuration so our programs don't put unnecessary burden on computing resources during run time.

- VBScript Code Optimization
- UFT Configuration and Tuning

We have listed down the most vital 10/10 points matrix. This list can be very exhaustive* So we opt the most important areas to get rid of any bottleneck. Area covered below are sufficient to deliver the efficient automation.

***ConfuSense**: We are not covering many key performance areas e.g. parallel computing,cache settings,LANs and networks,Buses architecture, CPU & clock speeds, communication protocols,I/O dataTransfer (DMA),VBscript internal architecture,Optimal configuration of system

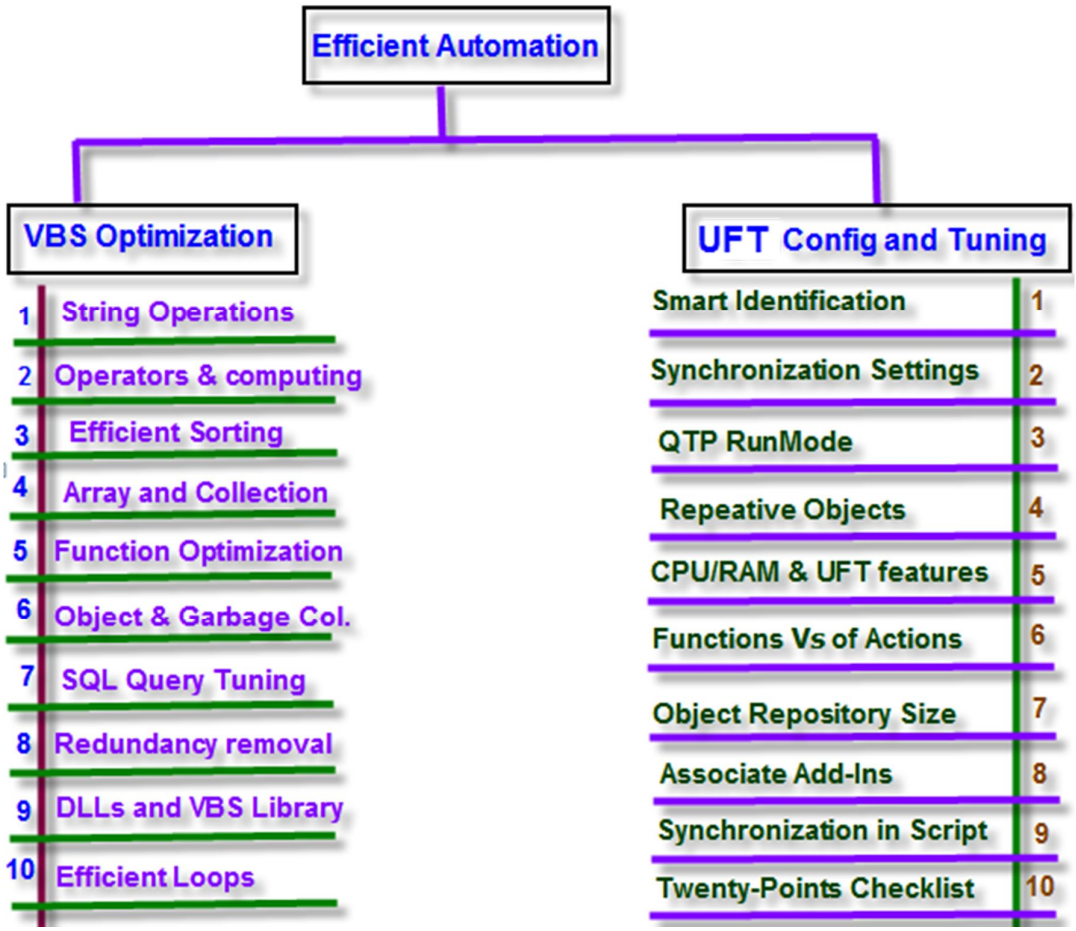


Figure 3-1 Key Performance areas in VBScript and UFT

4.0 VBScript Code Optimization

VBScript is the soul of UFT automation. VBScript was born to automate system administration tasks and designing web applications. Newer technologies replaced VBScript for web-designing but it is still a powerful tool in the hand of Windows system administrator to do system configuration and automation tasks. Internet Explorer was/is (disputable) leader in web browsers. Windows machines that used majorly in regression and system testing in 2000s era.QTP, UFT's predecessor, proved very efficient with VBScript to manage Windows as well as Web applications.

Now QTP, reskinned as UFT, provides features to access many technologies and browsers. But this is also a main reason for QTP/UFT not to have a cordial relation with UNIX and Linux systems till now. UFT with help of VBscript can do many automation tasks including

1. Managing and scheduling System tasks
2. Window scripting host programming (WSH)
3. Registry Manipulation
4. managing system configuration,Time and settings
5. Interacting with Browser DOM
6. Interacting with other COM applications APIs,Databases, Filesystem and custom applications
7. Programming of WMI, ADSI, Active Directory, Microsoft Office

A single book will be insufficient to list down on VBScript capabilities. Our goal is to focus on the most efficient VBScript programming ways for automation. Automation accomplish a repetitive task. Saving few seconds in one run can translate into many minutes (and some time many hours) time saving for whole execution.

4.1 String Computing

VBScript has only one datatype "Variant". Variant, the chameleon, behaves according to the usage

```
2 + "A"   '2 is String in this case
2 + 2     ' 2 is Integer in this case.
```

Variant mostly behaves as string. String operations and manipulations are the most used statement in script. UFT use string datatype in nearly everything e.g. validations,handling object properties, decision making,exception handling and reporting. It is correct to say that String manipulations are the building blocks of automation.

4.1.1 How String Stores in RAM

VBScript string is a BSTR* datatype. String is a sequence of one or more characters and copied in RAM in contiguous memory blocks during run time. Storage of one character in string takes 2 byte (they are Unicode instead of ASCII) so total number of storage in RAM can be calculated as

Byte Needed for String = 4 + 2 * (Number of character in string) + 2 Byte

Sequency Of Storage	Header Length Prefix	Each Char in string take 2 Byte	String Terminate Signal
Bytes Used			
Number of Bytes	Four Byte	2 Byte per char	2 Byte
Data		Unicode Chr	NULL NULL

STRING (BSTR) strorage in RAM = 4 +2(each chr) + 2 = 8 Byte for one char in strin

Figure 4-1 String storage in RAM

String can store up to 2 Billion (Yes it can) characters but after 64KiloByte size (default cache size) performance degrade substantially.

***ConfuSense**: A BSTR(binary string) is a pointer. The pointer points to the first character of the data string, not to the length prefix.* . - MSDN Library

4.1.2 String concatenation mechanism

String concatenation is very intensive process and consume lot of RAM and CPU cost.
Following are the steps for **String1 = String1 & String2** statement.

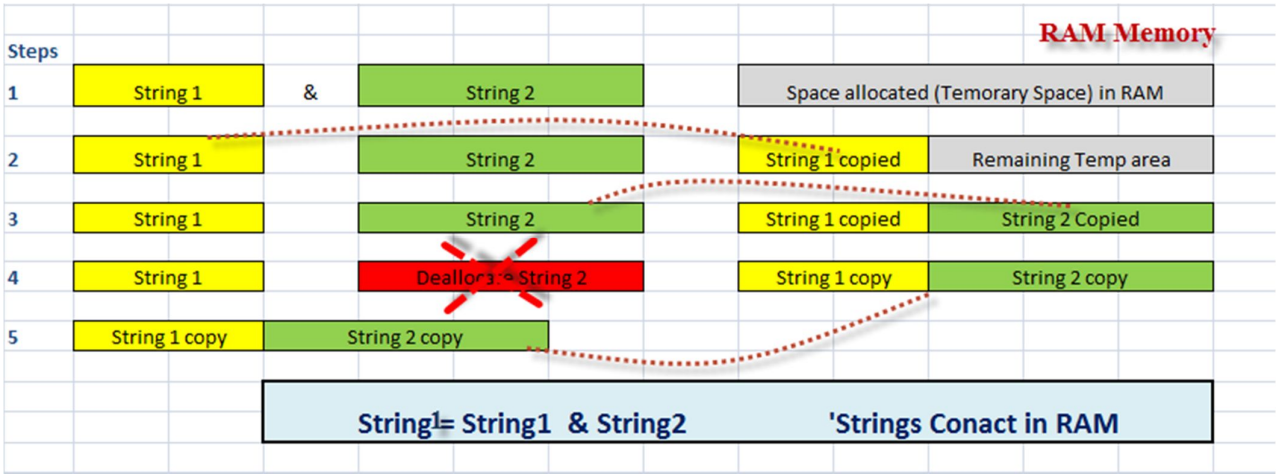


Figure 4-2 String concatenation in RAM (String1 = String1 & String2)

Steps in RAM:

1. Allocates temporary memory large enough to hold the result.
2. Copies String1 to the start of the temporary area.
3. Copies String2 to the end of the temporary area.
4. De-allocates the old copy of String1.
5. Allocates memory for String1 large enough to hold the result.

4.1.3 Performance Optimization of Strings

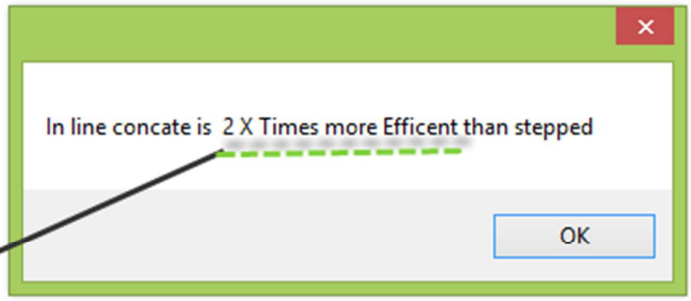
String manipulations can have severe impact on system performance and programmer should use them judiciously. We will explore the string concatenation mechanism and performance rewarding approaches in this section.

4.1.3.1 Minimum use of intermediate strings - Concatination Optimization

Minimum use of intermediate strings will save lots of computing. Following program concat string in two ways
1. Multi-step concatenation
2. Single step concatenation

Following figure depicts that single-step concatenate is twice efficient than the other. If strings are big (e.g. reading file from filesystem,database,webpages) than the difference explode exponentially and at the end performance become very sluggish by option one. String manipulations in loops are also severely degrade the performance .

```
beforStep = Timer
For i = 1 To 1000
    strMySting= "Hi"
    strMySting = strMySting & "hello"
    strMySting= strMySting & "Bye"
Next
afterStep = Timer

dblconcatInSteps = afterStep - beforStep

beforStep1 = Timer
For i = 1 To 1000
    strMySting= "Hi" & "hello" & "bye"
Next
afterStep1 = Timer

dblconcatInline = afterStep1 - beforStep1
msgbox "In line concate is  " & Cint(dblconcatInSteps / dblconcatInline) & " X Times more Efficent
```



Program: Prog 3.0 - Advantage of String Concatenation in single step over multiple steps

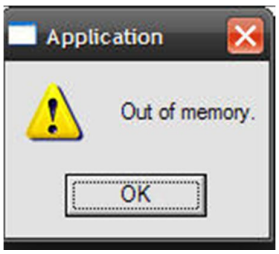
4.1.3.2 Huge Strings - Concatination Optimization

If we are reading a file or manipulating huge size strings then we may get "Out of Memory" errors. If we run out of 64K string cache space than performance degrade substantially and eventually can run out of RAM space as well.

To overcome "Out of Memory" errors or paging difficulties we can save the strings directly in file or temporary storage and escape manipulation in RAM. The performance penalty to interact with file should be compensate by more dangerous "Out of Memory" errors or paging cost.

```
Function ReadEntireFile(filespec)
   Const ForReading = 1
   Dim fso, theFile, retstring
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set theFile = fso.OpenTextFile(filespec, ForReading, False)

Do While theFile.AtEndOfStream <> True
   If File Size is long then it is death call
       retstring = retstring & theFile.ReadLine
   Loop
  theFile.Close
   ReadEntireFile = retstring
End Function
```

Program: Prog 4.0 - Creating Huge string in LOOP to kill memory



4.1.4 VBScript String Function Optimization

VBScript have many in-built functions for arithmetic operations, string operations, datatype conversion operations and validation functions. In this section we will discuss the relative cost of few functions and their alternate solutions.

4.1.4.1 Use InStrRev Judiciously

InStrRev is much slower than InStr. Best way is to avoid it. InStrRev has much better performance with vbBinaryCompare than with vbTextCompare.Searching for the middle

character in 20 character input string, InStrRev spent 5 times the amount InStr spent.

Figure 4-3 InStrRev specification and drawback with vbTetxCompare

4.1.4.2 Replace Function Optimization

Replace works slowly for vbTextCompare compare to vbBinaryCompare .We can club Instr with Replace for case sensitive vbTextCompare option when application of replace depends on the outcome of decision control. Suppose If a replace is unlikely to occur, we should first call Instr to check whether a replace is required or not.

Example

If you need to replace "ThisString" with "NewString" then first test with InStr if "ThisString" exist and then call Replace function.

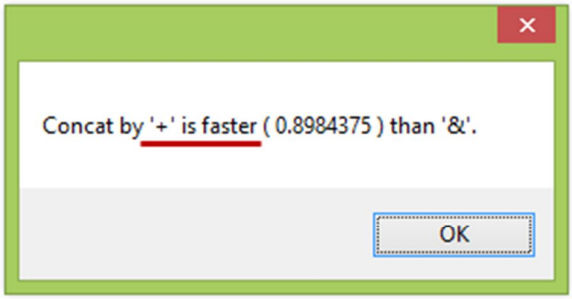
4.1.4.3 CDbl and CCur On Requirment

CDbl and CCur are worst performer in comparison of other conversion functions. They perform much worse inside the big time consuming loops. Programmer should use them judiciously.

4.1.5 Trade-Off between '+' and '&' Operator

In VBScript, there are two ways to concatenate two string variables: the & operator and the + operator. The '+' operator is normally used to add two numeric values and it works fast.But we should avoid it to eliminates any ambiguity in our code. Due to dynamic nature of VBScript datatype '+' may add the two strings as integer and can lead code readability and unexpected outcome issues.Performance penalty is not significant in comparison of better coding practice in this case.

```
beforStep = Timer
For i = 1 To 1000

   strMySting= "Hi" + "hello" + "Bye"

Next
afterStep = Timer

dblconcatByPlus = afterStep - beforStep

beforStep1 = Timer
For i = 1 To 1000
   strMySting= "Hi" & "hello" & "bye"
Next
afterStep1 = Timer

dblconcatByAnd = afterStep1 - beforStep1

print dblconcatInSteps
print dblconcatInline

msgbox "Concat by '+' is faster ( " & ( dblconcatByAnd - dblconcatByPlus ) & " ) than '&'. "
```



Program: Prog 5.0 - '+' operator is slightly better but always use '&' for string concatenation

4.1.6 Use Null String instead of Empty string

There are two ways to represent a zero-length string:

The null string vbNullString
The empty string ""

The only difference is that vbNullString is faster and it saves bytes in memory.

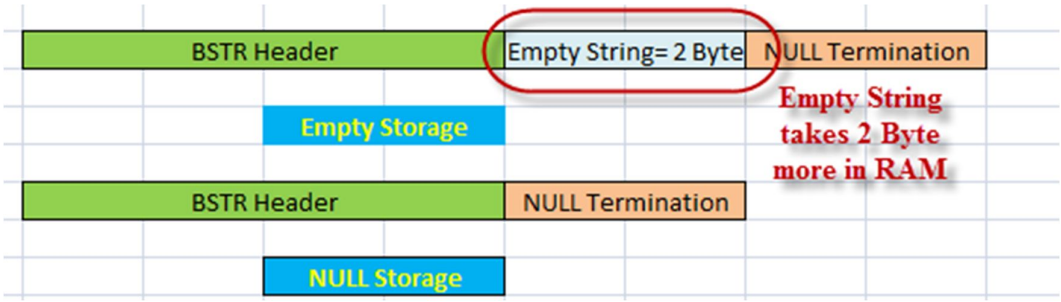


Figure 4-4 Difference of NULL and EMPTY string in RAM

4.1.7 Checklist of String Optimization

4.1.7.1 StrComp is a performance intensive operation. If we need case-sensitive search for multiple iterations then UCase and LCase should be used.
4.1.7.2 Many VBScript functions provide Binary and Text comparision options.Preference should be given to binary comparision.

4.2 Operator Computing

Following table gives the comparative idea about the relative performance of mathematical and decision computing

Operator	Slower	Faster	Comment
Evalponential	val ^ 2	val * val	The evalponential operator and functions(sqr) is often slow.
	val ^ 3	val * val * val	
>, <, <>, >=, <=		Greater than (>) and (>=)	

Figure 4-5 Performance Comparison of Operator computing

4.3 Efficient Sorting

A bubble sort is simple and good for small arrays, especially ones that are nearly sorted already. However, as the set of data to be sorted starts to grow, a bubble sort becomes quite inefficient. The sort that is regarded as the best sort for large arrays that are NOT nearly sorted is Quicksort.Figure depicts the relative cost of most popular sorting algorithm. Sorting algorithms are out of scope of this chapter.

Name	Best	Average	Worst	Method
Bubble sort	n	n^2	n^2	Exchanging
Insertion sort	n	n^2	n^2	Insertion
Merge sort	$n \log n$	$n \log n$	$n \log n$	Merging
Quicksort	$n \log n$	$n \log n$	n^2	Partitioning
Selection sort	n^2	n^2	n^2	Selection

MOST POPULAR SORTING COST COMPARISION

Figure 4-6 Cost of most popular sorting algorithms

4.4 Array and Collection Computing

There are static and dynamic arrays in VBScript. Array use contiguous memory space in RAM. Static arrays are very efficient and each element is OLE VARIANT type and take 16 byte memory. Discussing about OLE VARIANT is out of scope of this book but in nutshell, it is a complex data structure contains memory pointer to represent run time data location.

Dynamic arrays and Dictionaries (also term as Collection) are two ways to add value at runtime in VBScript. Array use Redim statement and dictionary use Add/Remove{key,value} methods to manipulate the data at run time.

Lets understand how Dynamic array work at run time. Dynamic arrays reserve new space and copy elements of old array to new location with every redim (resize) statement. Space reservation and element cloning is a performance overhead. The performance become big issue when arrays grow very long.So, Effectively resizing array from 800 to 801 elements takes approximately 100 times more effort compare to resize it from 5 to 6 elements.

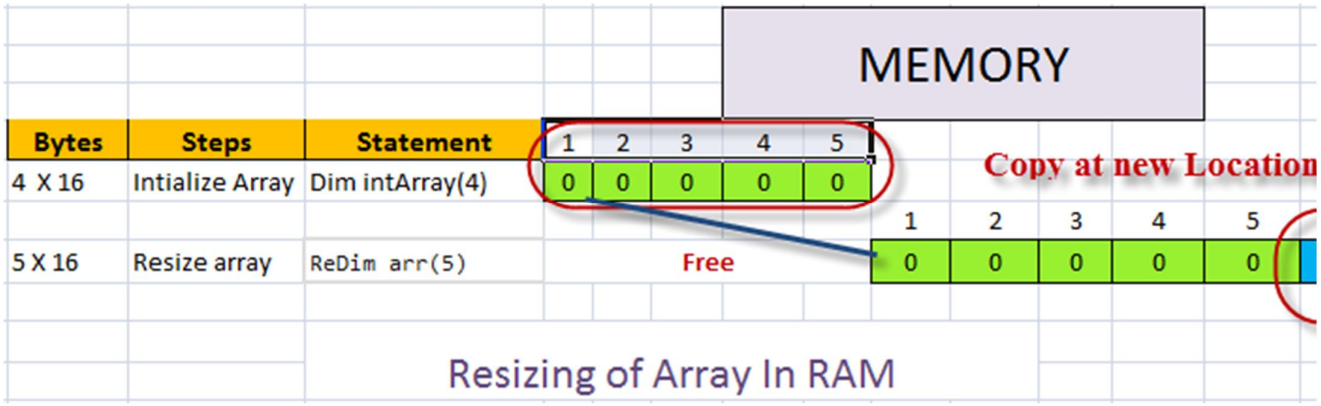


Figure 4-7 Dynamic array (ReDim) and RAM operations

Creating a dictionary takes much longer than creating a dynamic array but as array grows the benefit of array disappear and dictionary's initial creation performance penalty overcome by overall performance benefit. Dictionaries also win single-handedly in functional features*

***ConfuSense**: *Every value in a* Dictionary *object is a **Variant**, which means you can create a **Dictionary** object that consists of almost any kind of value (including other **Dictionary** objects), and that you can store any combination of **Variant** types in the same **Dictionary** object.*

- MSDN Library

4.5 Function Optimization

Function and Sub Procedure modularize the program/logic. They can take arguments and perform user business logic. Function can return value while Sub Procedure cannot return anything.

```
Function Example(ByVal intVal1, ByRef intVal2)
  'intVal1 is a copy and intVal2 is orginal variable
  Example = intVal1 + intVal2
  intVal1 = 32
  intVal2 = 21
End Function
```

Program: Prog 6.0 - Sample Function with argument passed ByVal & ByRef

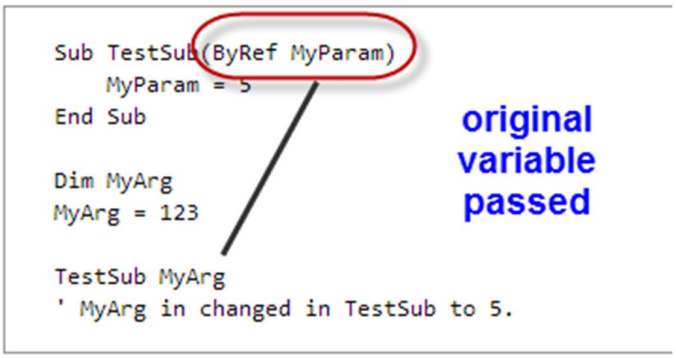
4.5.1 Passing argument using ByVal and ByRef

Function can take argument in two ways.

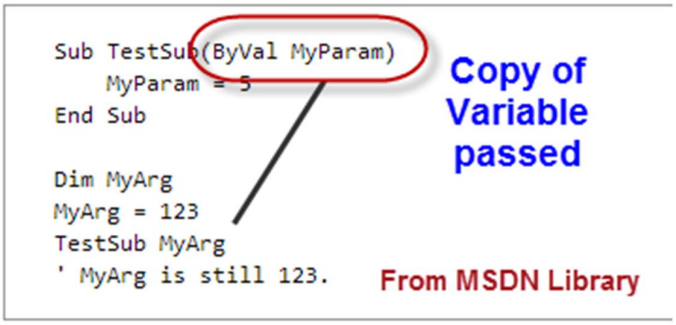
1. ByRef*
2. ByVal

ByRef is by default.

ByVal copy the variable from original variable in the function argument while ByRef pass the memory location. So, operation on ByVal happens on the new copied variable and operation on



In the following example, the **ByVal** keyword is used. Therefo
the value of MyArg remains unchanged.



ByRef happens on original variable.

Figure 4-5-1 Dynamic array (ReDim) and RAM operations

If variable size is huge and original value destine to change after function call than ByRef is most suitable option.But if we need to preserve the original variable than ByVal should be used in the function argument.

4.5.2 Local Variable Vs. Global Variable inside Function

Local variables are those declared within subroutines and functions while Global variable is accessible for all functions. Within a function or subroutine, local variable access is faster than global variable access. Also local variables usage make code more cleaner.

4.5.3 Exit Function and Exit Sub

Sometime if user want to exit the block of code after specific event, error or matching any condition. Programmer must keep checking the execution state and forbid interpreter to execute unnecessary statements.These are the following braking function that can save enormous amount of computation time.

VBScript EXIT CODE	UFT EXIT CODE
Exit Do	ExitActionIteration
Exit For	ExitAction
Exit Function	ExitTestIteration
Exit Property	ExitTest
Exit Sub	~~ExitGlobalIteration~~ (Obsolete)
	~~ExitRun~~ (Obsolete)

Figure 4-5-3 Exit in UFT and VBScript

4.6 Object Utilization and Garbage Collection

Optimal Memory usage and program computing is vital element of system performance. When we create variables they store in memory. If we keep creating variable without cleaning old unused and unwanted variables then memory will be out of order after sometime. To overcome this problem VBScript provides implicit Garbage Collector. In this section,We will discuss the scenarios when we should proactively release memory.

4.6.1 Garbage Collection

VBScript is smart enough to deallocate memory automatically. VBScript Garbage collection is Stack-Based* algorithm. Variable inserted into stack when they come in scope and become live during program run.But If variable goes out of scope e.g. function terminates and local variable goes out of scope or script terminate than memory reclaim back by garbage collector and used for other variables.

Figure 4-8 Automatic Stack-Based VBScript Garbage Collector

The story differs in the case of Objects. Object can have multiple references pointed by many variables. So,setting object to nothing explicitly will decrement the count of reference by 1.

Set MyObj = Nothing

If count reaches to zero than object 'can' claim by garbage collector and memory become available for other task otherwise memory will block for any purpose i.e. 'Memory Leak'.

*__ConfuSense__ : *Stack is Last-In-First-Out (LIFO) mechanism i.e. Last inserted value will come out first. It is opposite to First-In-First-Out (FIFO) mechanism e.g. ticket counter line.*

HandyBit: Verify Object existence by validate with IsNull function.

4.6.2 Object Deallocation and Reuse Object in Multiple Functions

Programmer must deallocate the objects after their use in programs . Various applications expose API to programmatically handle the feature e.g. Quality center, excel,database connection,Internet Explorer, other COM API etc. Garbage Collector does not interfere in memory deallocation for the Object communicating with these applications.

Also, If programmer need to call the object in many actions/functions than he/she must keep alive the object in global context and use in multiple functions. Otherwise creating and destroying the object multiple time severely impact the performance of system.

Example:
If Excel read operation is called by many functions in program.Only one time creation and release should happen for excel object for all the functions. Performance will severely impacted if programmer create and release object for each function call.

4.7 SQL Query Optimization

This topic can be explore in great detail and out of scope of our book. VBScript can make connection of database and run DDL* and DML* SQL queries. Programmer should use queries that takes minimum time.

***ConfuSense**: *DDL - Create, Alter, Drop queries and cannot be rollback i.e. auto-committed. DML - Select, Insert, Update,Delete queries and can be rollback.*

4.8 Redundancy Removal

VBScript is very flexible Object Oriented Base language. Programmer should avoid duplicity of the statement to remove the re-executing same statement.

Code Review process can reduce redundancy and identify any logical script errors in script. So keep eye on the others comments.

```
oExcelSheet.cells(1,1).interior.colorindex=36
oExcelSheet.cells(1,2).interior.colorindex=36
‘  Again write same code or repeat code unnecessarily
oExcelSheet.cells(1,1).interior.colorindex=45
oExcelSheet.cells(1,2).interior.colorindex=45
```

Program: Prog 7.0 - Repetitive redundant steps

4.9 DLLs and Library

There are two types of library
1. Static Library
2. Dynamic Link Library (DLLs)

Static and Dynamic library provide the modularity by writing pre-compile programs in various languages (C++, VB.Net, C#.Net and so on). Static libraries are fast comparative to DLLs but there are other advantages of using DLLs in automation design. To chose the effective method depends on the automation architecture. Test automation architecture is out of scope from this chapter and will be discussed this area in another chapter.

4.10 Efficient Loops

IT will be appropriate to say that automation is all about Loops, decisions,Object and strings manipulations. These are the pillars of automation design. Loop computing create heavy performance penalty for unwise use. We will discuss few approach that will lower the computation.If you are using UFT to run VBScript code then you should use ‘Fast Mode’ otherwise you will not able to judge the difference.

4.10.1 Avoid assignment and Calculations in Loops

In this example we assigned a=i inside the loop and time exponentially increase to 190 seconds. The performance will more degrade if these assignment comprises huge strings or objects.

Example 1:

```
startTime_SC= timer
For i = 1 To 10000
a=i
If a=10 Then

    If b=12 Then

    print "a and b are even number"
    End If
End If
Next

endTime_SC=timer
print  (endTime_SC- startTime_SC)
```

2 X Times Higher
Assigning a variable in
Loop increase time from
93.89844
to
190.4023

Program: Prog 8.0 - Assignment inside the Loop decrease performance

Example 2:

```
Dim angle : angle=45
startTime = Timer
 For i=0 To 100000000
   'COMPUTING INSIDE THE LOOP
   y = y + Sin(angle)*100
Next
 endTime = Timer
print  (endTime- startTime)

startTime_SC1 = Timer
 'COMPUTING OUTSIDE THE LOOP
 math_sind = Sin(angle)*100
 For i=0 To 100000000
  y = y + math_sind
 Next
endTime_SC1 = Timer
print  (endTime_SC1- startTime_SC1)
```

UFT Fast Mode

33 Sec

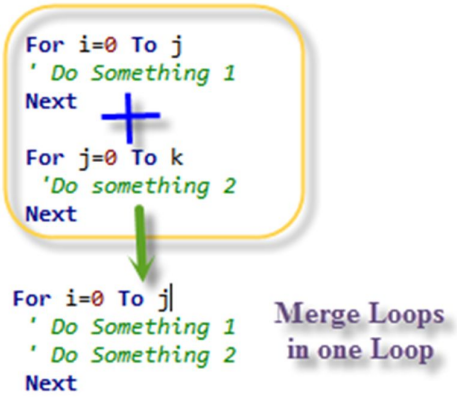
19 Sec

UFT Normal Mode gives no difference

Program: Prog 9.0 - Calculation inside the Loop decrease performance

4.10.2 Merge Loops

If two loops are independent and run for same number of time then merging them will greatly enhance performance of test execution. Beware to remove redundancy and test comments so



refactoring*

Program: Prog 10.0 - Merge two independent loops in one

***ConfuSense** Refactoring is a term to modify and optimize computer programs. Many tool assist automatically in this task.e.g renaming one function can automatically change the name of that function in every used script. Unfortunately UFT still lack most of the refactoring capabilities.*

4.10.3 Exit to Force Loop Break

'Exit For' and 'Exit Do clubbed with decision controls (If, Select-Case, If-Elseif) can break the loop prematurely. It is one of the most beneficial performance saving way to prevent loop processing.

5.0 UFT Tuning and Optimum Configuration Settings

First let's understand how UFT works and the area where we should look to enhance automation performance.UFT mimic the user instruction on the GUI level. So effectively, to perform any operation, UFT depends on system configuration(OS, IE settings,Registry settings), GUI technology,interference of other processes on system(Firewall,proxies,anti-virus algos,pop-ups etc) and the communication intricacy between application and UFT. Any delay or latency in these area leads the performance bottleneck during the execution. We will discuss few common area where performance can be optimize.

5.1 Smart Identification

If QuickTest is unable to find any object that matches the recorded object description, or if it finds more than one object that fits the description, then QuickTest ignores the recorded description, and uses the Smart Identification mechanism to try to identify the object.

These are the disadvantages of Smart Identification

1. Its time consuming process and decrease performance
2. OR size gets increased so RAM memory utilization increase
3. Some time Smart Identification induce error by finding the wrong Object.
4. Smart Identification does not work well with Set/GetRo and GetTo

Golden Rule If you have robust error handling and well maintained test suite than Disabling the Smart Identification is the best action.

5.1.1 Disabling Smart Identification

User can disable smart identification from UFT 11.5 GUI or by UFT Object Model

***HandyBit**: Navigate to UFT11.5 | File| Setting | Run*

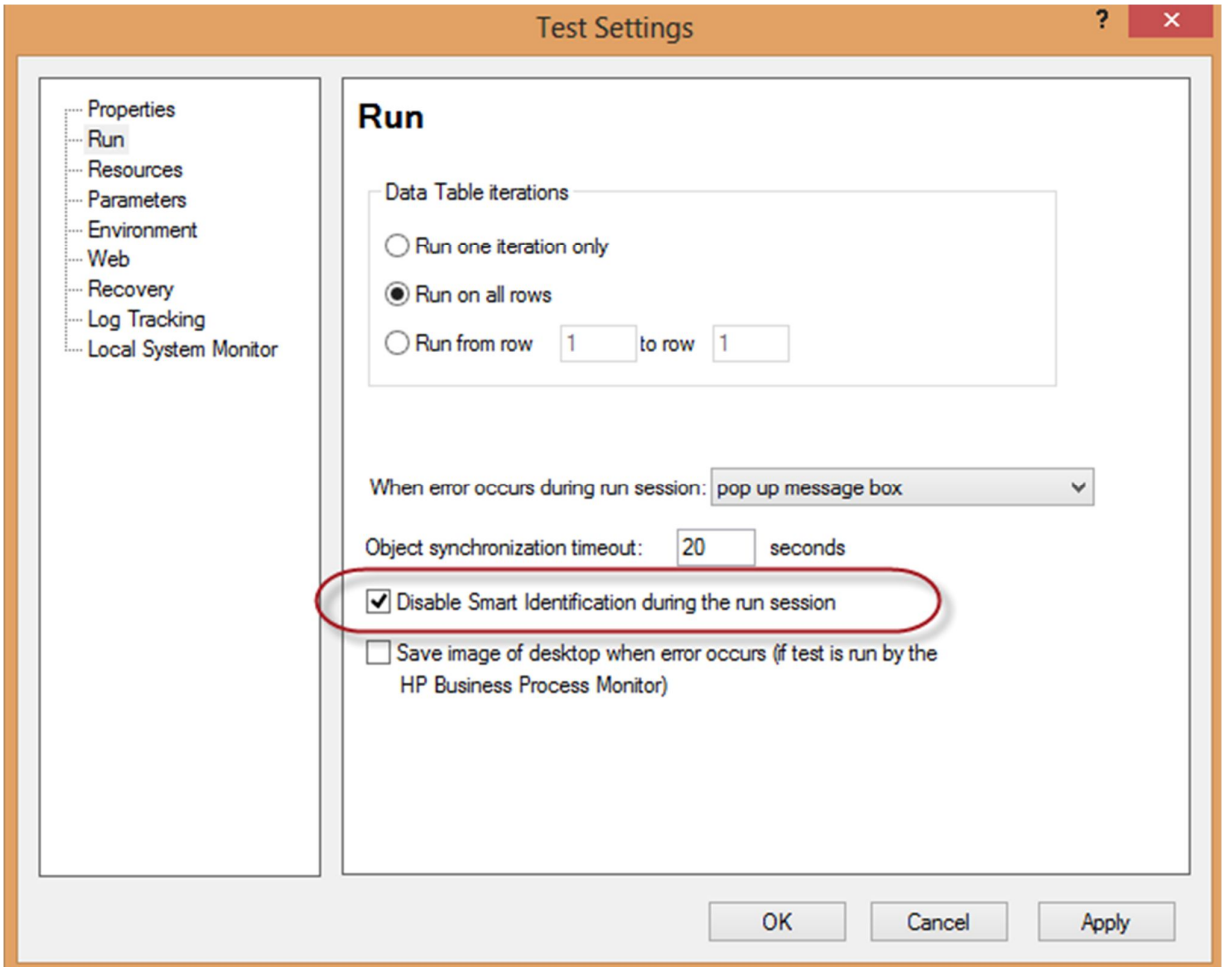


Figure: Fig 5.1 Disable Smart Identification by UFT GUI

```
'In UFT Object Model
Set qtApp = CreateObject("QuickTest.Application")
qtApp.Launch
qtApp.Visible        =        True
qtApp.Test.Settings.Run.DisableSmartIdentification = True
```

Program: Prog 11.0 - Disable Smart Identification by UFT Object Model

5.2 Synchronization Optimization

Most of the time speed of executing instruction by UFT (fast) mismatch with Application(AUT) navigation speed(slow). To overcome this difficulty, UFT provides various ways to synchronize UFT and AUT.UFT also provide wait, Exist and WaitProperty methods to handle the synchronization at run time that we will discuss in section 5.9.1.

5.2.1 Object Synchronization TimeOut

Object Synchronization Timeout (OST) is the time UFT wait for object appearance during run time.

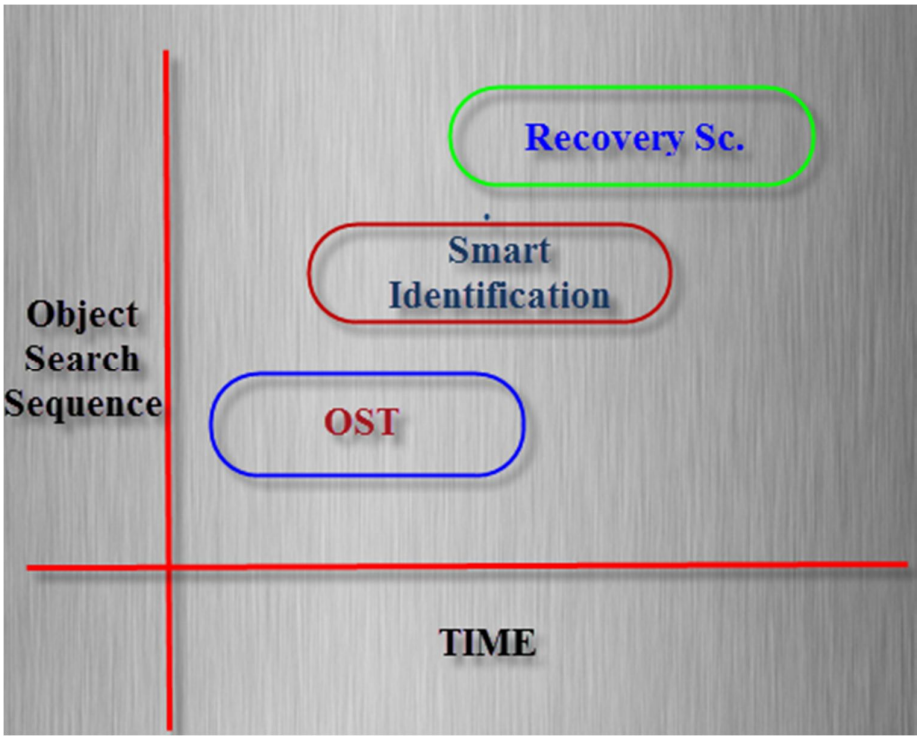


Figure 5-2 OST , Smart Identification and Recovery invocation Timeline

UFT should set the optimal OST time. By default it is 20 second.Normally It should set to much lower value.

***HandyBit**: Navigate to UFT11.5 | Files | Setting | Run | Object Synchronization*

TimeOut

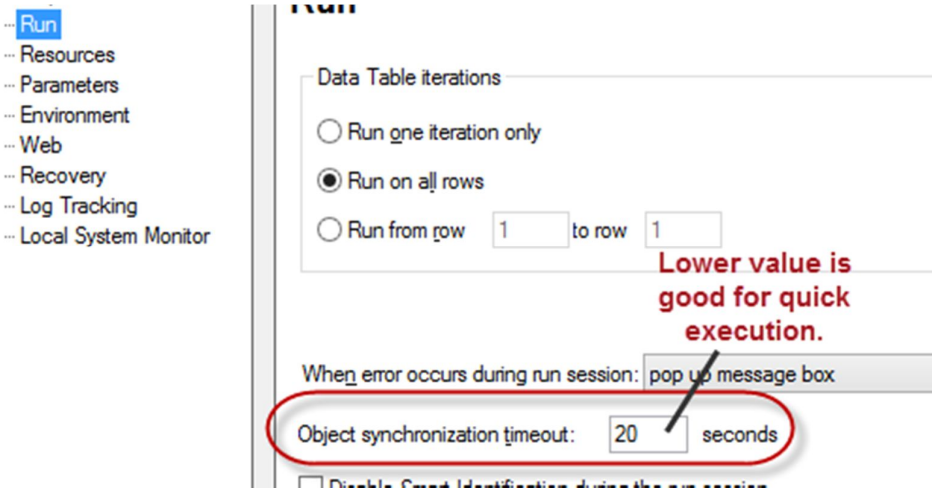


Figure 5-2 Object Synchronization TimeOut setting in UFT

5.2.2 Browser Navigation Timeout

UFT set maximum time to wait for a web page to load before running step into the test. UFT waits up to the amount of time set for the Browser Navigation Timeout option, plus the time set for the Object Synchronization Timeout.

***HandyBit**: Navigate to UFT11.5 | Files | Setting | Run | Browser Navigation TimeOut*

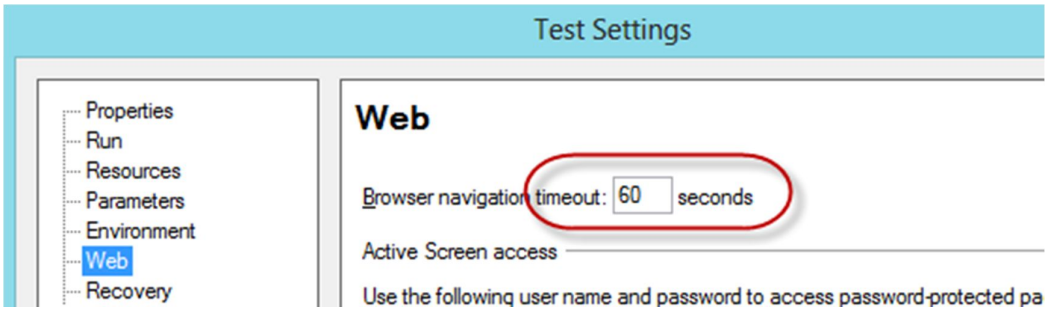


Figure 5-2-1 Browser Navigation TimeOut setting in UFT

5.3 Fast Mode Run

UFT runs tests in 'Normal Mode' from File System and 'Fast Mode' from Quality Center(QC). Fast mode should be set for quick execution. It is one of the most critical performance enhancement in the automation execution.

You can change in mic.ini file setting to run UFT in fast mode from QC. Default location of mic.ini can be found at C:\Program Files (x86)\HP\Unified Functional Testing\bin\mic.ini Search for [RemoteAgent] section in the file and provide RunInFastMode=1 (you may need to change the permission of read-only mic.ini file before amendment).

***HandyBit**: Navigate to UFT11.5 | Tools | Options...| GUI Testing | Test Runs*

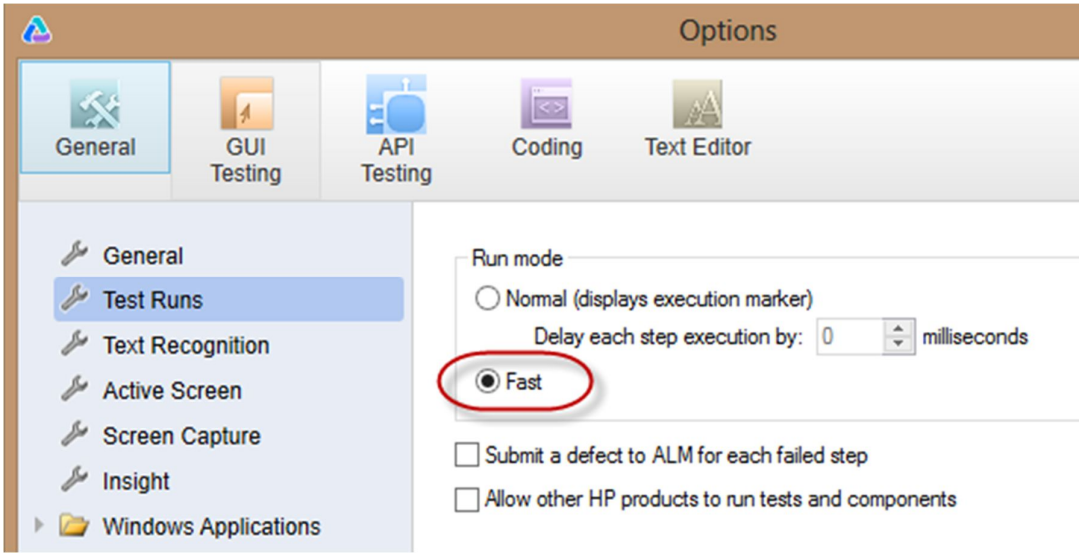


Figure 5-3 Setting Fast Run Mode in UFT

5.4 Reuse Object Reference of repeated Object Hierarchy

UFT identifies the object by computing the full hierarchy of the Object. Suppose we want to set the value in WebEdit box

We can do it in **three** ways

1. Set value by calling complete hierarchy
Browser("X").Page("Y").WebEdit("Z").Set myVal

2. Set value by reuse the wrapped object reference
Set objWebEdit = Browser("X").Page("Y").WebEdit("Z")
objWebEdit .set myVal

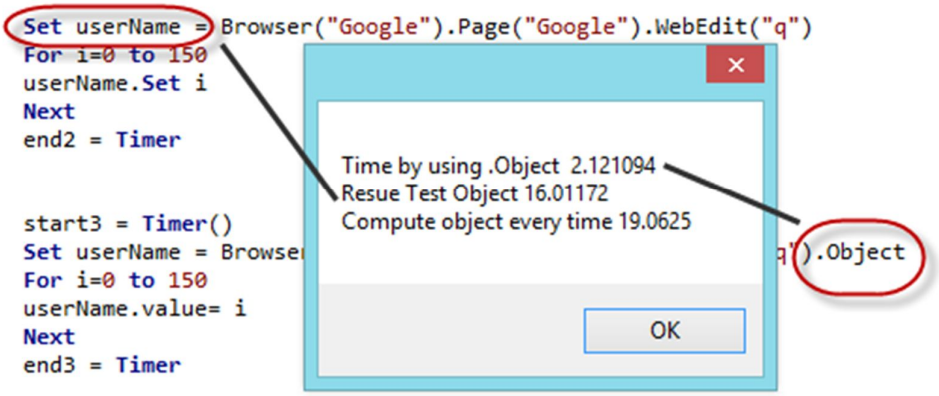
3. Set value by manipulate Object at runtime
Set objROWebEdit = Browser("X").Page("Y").WebEdit("Z").**Object**
objROWebEdit.value = myVal

If we need to run the same operation repeatedly then the performance of option-3 is best in comparison to option-2 and option-1(less than Option-2). Lets run one example

```
start1 = Timer
For myVal=0 to 150
```

```
Browser("Google").Page("Google").WebEdit("q").Set myVal
Next
end1 = Timer


start2 = Timer
Set userName = Browser("Google").Page("Google").WebEdit("q")
For myVal=0 to 150
userName.Set myVal
Next
end2 = Timer


start3 = Timer
Set userName = Browser("Google").Page("Google").WebEdit("q").Object
For i=0 to 150
userName.value= myVal
Next
end3 = Timer
Msgbox "Time by using .Object  " & (end3 - start3) & vbNewLine & "Resue Test Object " &
(end2 - start2) & vbNewLine & "Compute object every time " & (end1 - start1)
```

Program: Prog 12.0 - Writing multiplier of 5 in a text file



Program: Prog 13.0 - Reuse Object computing in UFT

5.4 Disable CPU and Memory Intensive UFT Features

Capturing and storing images and video at run and design time consume lot of memory and resources and should be avoided.

5.4.1 Disable Active Screen

UFT saves the application Objects and screen during recording.It increase size of the test in RAM during execution. User should disable the Active screen options.

***HandyBit**: Navigate to UFT11.5 | Tools | Options...| GUI Testing | Test Runs*

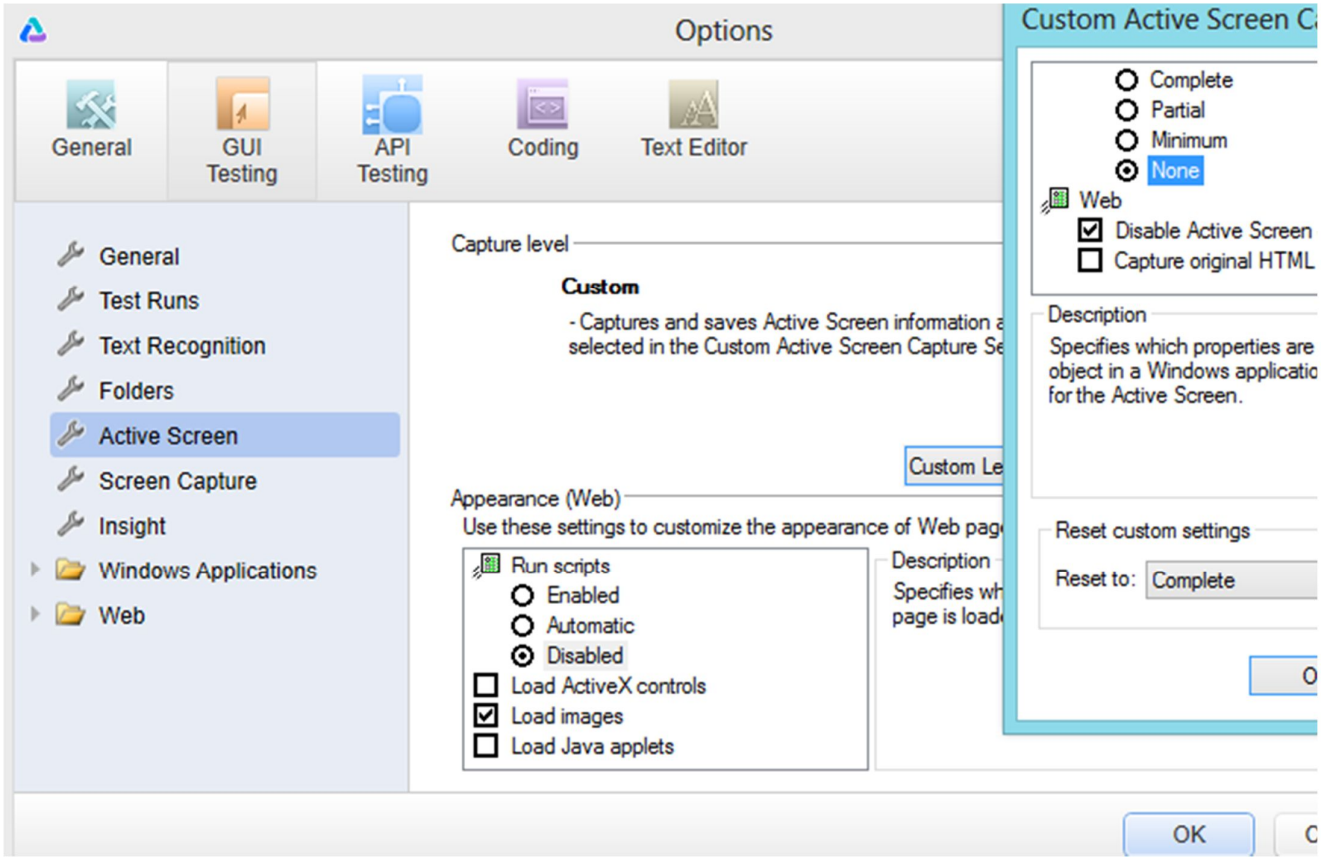


Figure 5-4-1 Active Screen Settings in UFT

5.4.2 Disable Screen/Movie Capture

UFT can save images and create movies during execution time . This option slows the performance of the test and It should use carefully.

***HandyBit**: Navigate to UFT11.5 | Tools | Options...| GUI Testing | Screen Capture*

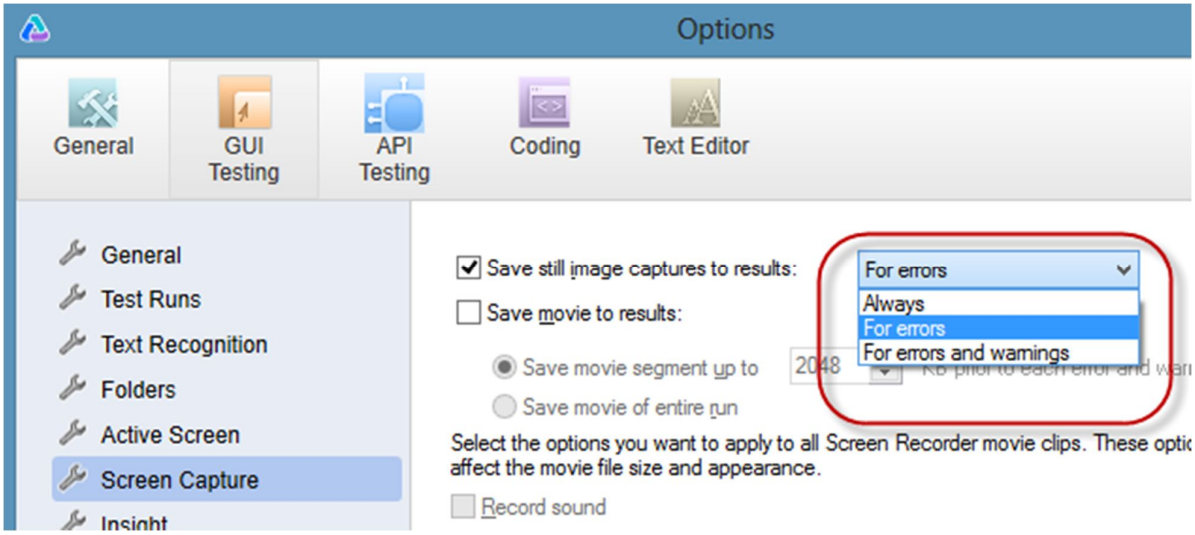


Figure 5-4-2 Screen Recorder in UFT

5.5 Prefer Functions instead of Actions

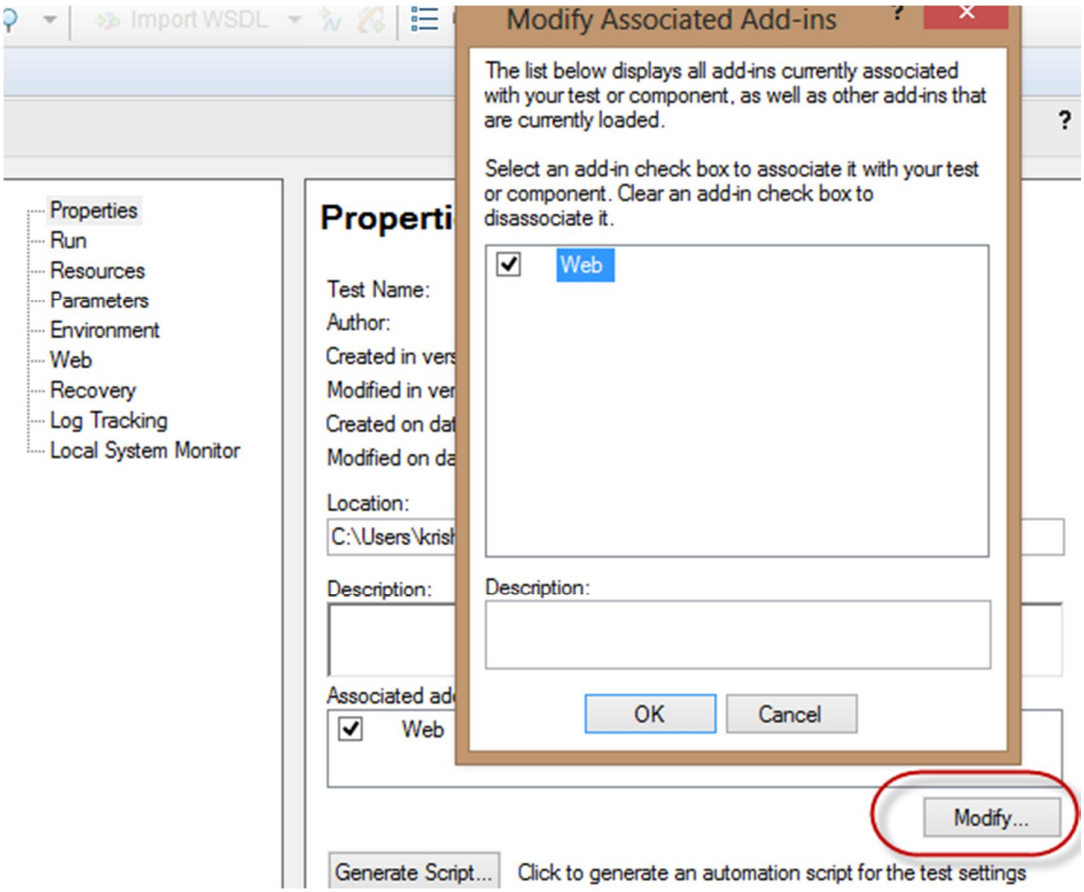
Action is a UFT concept. Action comprises repository, datatable and It executes slower than VBScript functions. Functions can be written in .qfl and .vbs files which we will discuss in next chapter.

5.6 Object Repository Size

Object repository(OR) loads in RAM during execution. If OR size is big then computation will severely impacted and can slow down the performance. UFT provide repositorycollection object to associate and remove the OR at runtime. Managing the good modular OR will give the better performance.

5.7 Associate Only Required Add-In

UFT provide Add-In for various AUT technologies. Associating Add-In will impact on the memory.



5.8 Descriptive Programming Vs Object Repository

This topic is out of scope of this chapter. We will discuss about it in great detail.

5.9 Prefer Other Sync method instead of Wait

UFT provide following ways for synchronization from code
1. Wait
2. .Exist
3. WaitProperty

5.9.1 Customize Synchronization with .Exist and Loop

As we discussed in Section 5.2 Synchronization Optimization, UFT provide configurable timeout settings. Object Synchronization timeout should set for minimum and programmer either use waitProperty or .Exist in code. Wait is a fix-time pause and prevent script to proceed even UFT finds object in less time. WaitProperty force UFT to wait till UFT match the specific Object Property. Details of these method covered in other chapter.

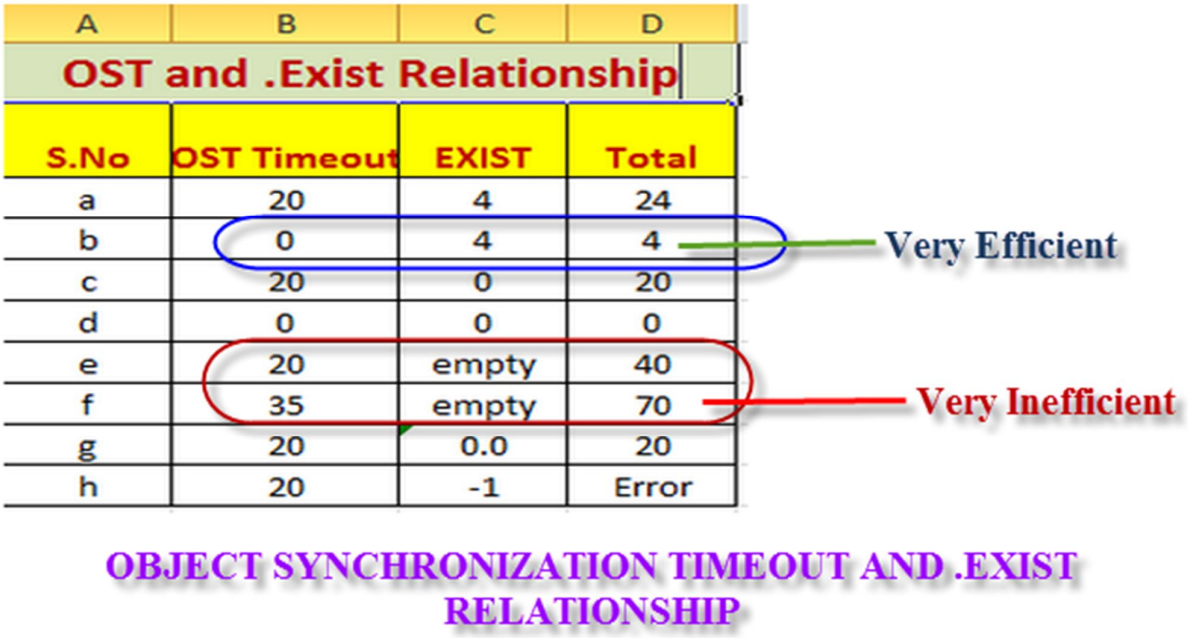


Figure 5.9 - Object synchronization and Exist relationship

We can set OST = 0 , .Exist(0) and can iterate loop with user define interval.

Example: If programmer wants to wait for 15 second for than we can use loop with .exist to wait only for 15 seconds.

```
Dim intCounter
intCounter = 0
        Do while  Browser("title:=My application").Page("Y").WebButton("Z")Exist(0) = False
                        Wait 1
                        intCounter = intCounter + 1


                        if intCounter = 15 Then
                              Exit For
                        end if
         Loop
```

Program: Prog 14.0 - Customize Synchronization with help of .exist,OST and loop

5.9.2 Set Global Wait for Script

Sometime we need to run same script in various environment (T1,T2,T3...) So we are force to use Wait(x) in the script. If this is the situation than we should provide the Wait with variable i.e.

Wait(intWaitingTime) at every place. So, effectively we can manage optimal wait for specific environment

```
Const dblWaitTestEnv1 = 10
Const dblWaitTestEnv2 = 15
Const dblWaitTestEnv3 = 20


Dim intTestEnv,intWaitingTime


intTestEnv = InputBox("Please provide the number of TestEnv")


Select Case intTestEnv
Case 1
    intWaitingTime = dblWaitTestEnv1
Case 2
    intWaitingTime = dblWaitTestEnv2
Case 3
    intWaitingTime = dblWaitTestEnv3
End Select
Wait(intWaitingTime) 'Location1
Wait(intWaitingTime) 'Location2
Wait(intWaitingTime) 'Location3
```

Figure 5.9 - Global Wait(intWaitingTime) can be manage with single variable

5.10 Twenty-Points Checklist

These generic settings and design consideration should be use as a checklist and most of them should apply on most of the automation framework.

5.10.1 Prefer Object Model instead of GUI test suite execution

5.10.2 Use Object Identification property best suited for Application

5.10.3 Prefer Custom Error handling instead of recovery scenario

5.10.4 Use ADODB connection instead of Excel API. Also prefer it instead of DB Checkpoint

5.10.5 Close all unnecessary process during execution

5.10.6 Use custom reporting instead of Reporter module of UFT

5.10.7 Clear temporary space of machine

5.10.8 Disable unnecessary Browser Add-ons before run. BHOManager Class Add-on is UFT add-on and It should never disable.

***HandyBit**: Navigate to IE | Tools | Manage Add-ons*

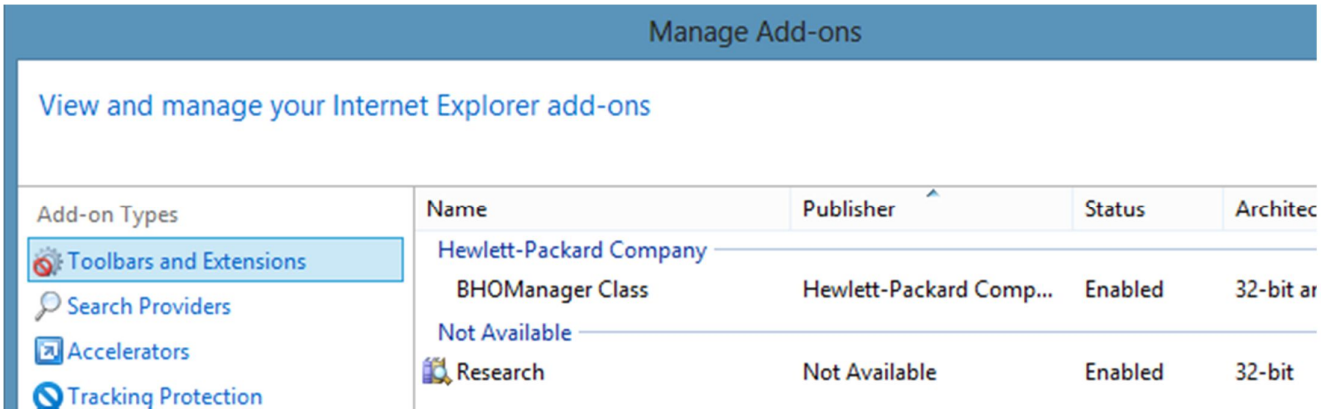


Figure 5-10.8 Add-On in Browsers

5.10.9 Declare (Dim) your variable. An undeclared variable references by name every time it is used. We should achieve this by forcing declaration by using 'Option Explicit'.

5.10.10 Prefer Local variable usage than Global variable

5.10.11 Clear Res-n result files that saved for each UFT results run. You can get the location in run dialogue before execution.

5.10.12 Delete all unnecessary objects from Object repository

5.10.13 If you can turn-off firewall and escape proxy then it will save computation

5.10.14 Interaction with QC takes time so if you do not need UFT and QC then disconnect QC. Prefer UFT-Filesystem instead of QC-UFT for debugging.

5.10.15 If UFT run on virtual machine(VM) than VM must have proper RAM allocation

5.10.16 External library should comprises the functions and function should not be the part of reusable action. Otherwise function will keep reside in RAM during complete execution.

5.10.17 Use "With" statement. It saves roundtrip to OR for adding more properties to object

***HandyBit**: Navigate to UFT11.5 | Tools | Options... | General*

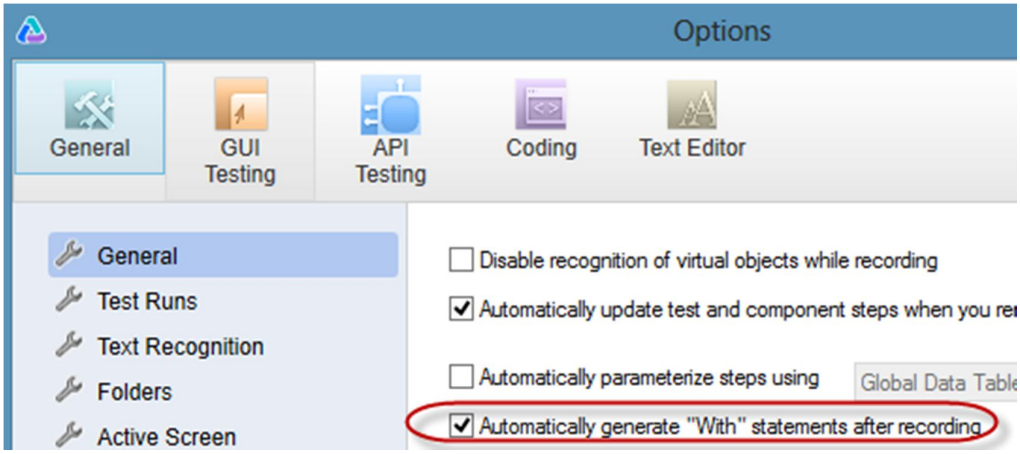


Figure 5-3 With Statement for UFT recording

5.10.18 Keep track of HP patches for UFT.

5.10.19 Restart machine everyday to flush out memory leak in RAM (volatile memory).

5.10.20 Prefer custom code instead of OptionalStep. OptionalStep slows performance for each run.

Summary

Good performance is vital quality character for successfully automation. Slow execution or run-time errors are the most critical automation problems during execution time. Performance considerations reap during planning and script design phase. We discussed various ways to enhance the performance of automation by VBScript code optimization and UFT tuning. Performance, a non-function quality attribute, is the context-driven criteria. Specific recommended best settings can lead worst performance in other scenario. So , Performance analysis and correct approach should be device for the overall system instead of case-by-case and script-by-script basis.

UFT and VBScript provide multiple ways to perform same task so we should also evaluate best possible method in particular situation. UFT and the system processes, system configuration,patches,firewall,proxies, anti-virus software, browser versions and settings,integration with other systems e.g QC,Database,APIs, OS and hardware play the pivotal role to impact the overall automation performance.

Where to Go from Here

Understand the deficiency in VBScript features and finding the alternate solutions by using different technologies and tools and integrate them in automation framework.

Good understanding of UFT Settings,files and UFT Object Model

Research on relative performance tradeoff metrics between UFT and Excel,database,datatable,network resource utilization in different scenarios.

Reference:

http://msdn.microsoft.com/en-us/library/ms123401.aspx