

UFT MIGRATION TO ASCENTIALTEST

BY Matryxsoft Tech LLP

Date: Sept 2022

Background and Introduction:

One of the customers having PowerBuilder application used UFT tool to automate 300 Testcases, over a period of time they had maintenance and some controls identifying issue. They were looking for an alternative testing tool and approached Zeenyx who owns AscentialTest™ (AT) Product, known as the Next Generation Management tool which stands No.1 for PowerBuilder applications.

Zeenyx knew about Matryxsoft Tech's experience of migrating Selenium code to AT code and also about the migration techniques of UFT to AT. Hence, they recommended the customer to approach us for migration of their entire UFT Project.

Matryxsoft Tech, an independent software testing company well known for its specialization in Migration and localization techniques had analysed the requirement from the customer and worked towards the same with its partner Zeenyx and migrated the UFT code to AT code which gave more robust and reliable tests.

The purpose of this paper is to provide a detailed understanding on how the migration was successfully performed from UFT to AT w.r.t to Object Repository, Steps/Functions, Tests.

Setup:

AscentialTest Tool and Project built on UFT are the basic requirement for migration.

Customer provided one of the complex testcase which consist of more than 100 function calls which extracted the input data from excel as part of the POC.

We segregated all the files w.r.t to extensions according to the conversion required. One can go right away for the conversion with the utilities that are prepared for migration but for a better conversion rate it is good to understand the UFT architecture and the framework built around for the application under test.

Application Knowledge and the manual flow testcase document will help in fine tuning the conversion at a faster rate.

AppObjects to AppObjects:

UFT Objects are recognized by the path statements like “window id”, “nativeclass”, “regexpwndtitle” and so on. At first all the files related to the app objects are copied and are converted to the AscentialTest AppObjects through the conversion utilities created. The conversion utilities handle all the necessary indentation and the syntax changes which brings exactly the way AscentialTest requires. The challenges are seen on the indentation of objects with multiple properties and some of the attributes which are not exactly the same with AT. By our utilities, challenges were addressed and brought them to the AT AppObjects Format. Please find the example of AppObjects conversion as shown in the Screenshot.

UFT	AT
"window id"	"@WinId"
"nativeclass"	"@WinClass"
"regexpwndtitle"	"@Title"

Challenges:

- Working on indentation on two - three levels.
- Parent and child object identification in UFT AppObjects is quite a challenge as parent object opening tags and child object closing tags are quite similar.
- It is difficult to transform UFT properties into AT Properties as some UFT properties differs from AT properties.

UFT AppObjects:

```

▼<qtpRep:ObjectRepository xmlns:qtpRep="http://www.mercury.com/qtp/ObjectRepository">
  ▼<qtpRep:Objects>
    ▶<qtpRep:Object Class="Window" Name="Report_InsigthObjects2">
      ...
    </qtpRep:Object>
    ▶<qtpRep:Object Class="Window" Name="PDF_Form_InsigthObjects">
      ...
    </qtpRep:Object>
    ▼<qtpRep:Object Class="Window" Name="BranchPLUSLogon"> Rectangular Snip
      ▼<qtpRep:Properties>
        ▶<qtpRep:Property Name="regexpwdntitle" Hidden="0" ReadOnly="0" Type="STRING">
          ...
        </qtpRep:Property>
        ▶<qtpRep:Property Name="regexpwdnclass" Hidden="0" ReadOnly="0" Type="STRING">
          ...
        </qtpRep:Property>
        ▼<qtpRep:ChildObjects>
          ▼<qtpRep:Object Class="WinEdit" Name="txtUserName">
            ▼<qtpRep:Properties>
              ▼<qtpRep:Property Name="window id" Hidden="0" ReadOnly="0" Type="NUMBER">
                ▶<qtpRep:Value RegularExpression="0">
                  ...
                </qtpRep:Value>
              </qtpRep:Property>
              ▼<qtpRep:Property Name="nativeclass" Hidden="0" ReadOnly="0" Type="STRING">
                ▶<qtpRep:Value RegularExpression="0">
                  ...
                </qtpRep:Value>
              </qtpRep:Property>
            </qtpRep:Properties>
          </qtpRep:Object>
        </qtpRep:ChildObjects>
      </qtpRep:Properties>
    </qtpRep:Object>
  </qtpRep:Objects>
</qtpRep:ObjectRepository>
  
```

AT AppObjects:

```

[-] app DialogBox BranchPLUSLogon
  ♦ path [@Caption=="Branch PLUS Logon",contains PbLabel [@Text=="User ID:"]]
  [-] TextField txtUserName
    ♦ path [right of PbLabel[@Text=="User ID:"]]
  [-] TextField txtPassword
    ♦ path [right of PbLabel[@Text=="Password:"]]
  [-] PushButton btnOK
    ♦ path [@Text=="OK"]
  [-] PushButton Cancel
    ♦ path [@Text=="Cancel"]
  
```

Functions to Steps/Functions:

UFT functions are written on multiple Action files that are created w.r.t features or modules. Each Action can be a function and there could be 'n' number of functions with 'n' number of parameters which may or may not have return data type. In AT, for all the functions which returns more than one datatype are kept as a function and the one with no return type or only one return datatype are converted to steps in AT. Numerous challenges were faced while migrating the functions on the datatypes, parameters, function calls, Subfunction calls, indentation, return datatypes and reading input data. All these challenges were successfully handled using the steps and function utilities created by Matryxsoft. There are multiple utilities that were written to handle the conversion techniques and with the help of this, we converted Actions of UFT to AT Steps/Functions. A Sample of the Action written in UFT is migrated to AT as shown in the below pictures

UFT Function:

```
'@Method clickButton_Level1PBwindow_DP(windowTitle,buttonName)
'@Author Chandra
'@Date 2 May 2018
'@Description This Function will click on the buttonName inside the Window Specified which will be identified by Window Title

'Ex: Call clickButton_Level1PBwindow_DP("Branch Setup", "Cancel")
'*****

Function clickButton_Level1PBwindow_DP(windowTitle,buttonName)
  On Error Resume Next
  Flage = False
  If Pbwindow("pbname:=" & "w_bp_frame").Pbwindow("regexpwndtitle:=" & windowTitle).PbButton("regexpwndtitle:=" & buttonName).exist(10) Then
    Pbwindow("pbname:=" & "w_bp_frame").Pbwindow("regexpwndtitle:=" & windowTitle).PbButton("regexpwndtitle:=" & buttonName).Click
    Flage = True
  ElseIf Pbwindow("pbname:=" & "w_bp_frame").Pbwindow("text:=" & windowTitle).PbButton("regexpwndtitle:=" & buttonName).exist(2) Then
    Pbwindow("pbname:=" & "w_bp_frame").Pbwindow("text:=" & windowTitle).PbButton("regexpwndtitle:=" & buttonName).click
    Flage = True
  ElseIf Pbwindow("pbname:=" & "w_bp_frame").Pbwindow("regexpwndtitle:=" & windowTitle).PbButton("regexpwndtitle:=" & buttonName).exist(2) Then
    Pbwindow("pbname:=" & "w_bp_frame").Pbwindow("regexpwndtitle:=" & windowTitle).PbButton("regexpwndtitle:=" & buttonName).click
    Flage = True
  End If
  clickButton_Level1PBwindow_DP = Flage
'wait 1
'Error Handling
If Err.number <> 0 Then
  Call writeReportLog("Failed in Step/Function : clickButton_Level1PBwindow_DP","Failed", Err.Description , "")
End If
```

AT Function:

```

◇ //Ignoring the Loan Processing
◇ //Example Call clickButton_Level1PBwindow_DP("Process Loan Validation","No")
▢ Boolean clickButton_Level1PBwindow_DP(String windowTitle,String buttonName)
  ◇ Boolean bFlag = false
  ◇ List<String> lsDialogTitle = {}
  ▢ if (BranchPlus.DialogBox[windowTitle].PushButton[buttonName].WaitWhileExists(5))
    ◇ BranchPlus.DialogBox[windowTitle].PushButton[buttonName].Click()
    ◇ bFlag = true
  ▢ else
    ▢ for (AppObject ao in DialogBox[all])
      ◇ lsDialogTitle.Add( ao.GetAttr("Title"))
      ▢ if (lsDialogTitle.Find(windowTitle)>0)
        ◇ BranchPlus.DialogBox[windowTitle].PushButton[buttonName].Click()
        ◇ bFlag = true
    ▢ if (BranchPlus.DialogBox[windowTitle].PushButton[buttonName].WaitWhileExists(5))
      ◇ BranchPlus.DialogBox[windowTitle].PushButton[buttonName].Click()
      ◇ bFlag = true
  ◇
  ◇ //Error Handling
  ▢ if GetErrorCount() > 0
    ◇ LogError("Failed in Step/: clickButton_Level1PBwindow_DP")
  ◇ return bFlag
  
```

Challenges

- Indentation of Converting UFT Actions to AT Steps.
- Handling parameters and function return datatype.
- UFT native methods are not straight forward as AT, hence need to develop own functions/utilities.

Eg.:

AddDate ()
 Currency () etc.,

Tests to Tests

UFT Testcases are in the form of a flowchart or descriptive language, calling its actions in sequence and every test is associated with the inbuilt Datatable either local or global sheet and also has its own Start and End. In AT, we have a concept of AppState that has OnStart and OnFinish which works similar way of UFT and also datatable are in-built in both the tools. The challenges faced in the test were related to Function call, Indentation and Parametrization. All these were handled by the test utilities that were written by us in order to convert the tests of UFT to AT Test format. The successful transition of the tests is as shown in the below screenshot.

UFT Tests

```

launchBranchApplication x
Main
36 If isDirectLoanApplicableForCurrentState Then
37
38   'Test Data path for Execution Test data
39   globalTestData_testDataPath = Environment.Value("BranchPlusSharedPath") & "Global_TestData\Create_NewLoans_LoanType_SuiteType.xlsx"
40
41   'Importing Test Data from Excel Shet(specific sheet)
42   Call importSpecificSheet(globalTestData_testDataPath,"DirectLoan")
43
44   'Reading Data Which is required for test execute
45   SuiteName = DataTable.Value("SuiteName")
46   loanType = DataTable.Value("Loan_Type")
47   customer_Type = DataTable.Value("Customer_Type")
48
49   'if Suite To Execute and Suite Name is equal to Execute test case
50   If Ucase(Environment.Value("SuiteToExecute")) <> Ucase(SuiteName) Then
51     ExitGlobalIteration
52   End If
53
54   'Customer Type is not validate exit the current Test Iteration
55   If Not(Trim(customer_Type) = "Joint" or Trim(customer_Type) = "Cosigner" or Trim(customer_Type) = "Individual" or Trim(customer_Type) = "Joint_Cosigner") Then
56     Call writeReportLog("Customer Type: "&customer_Type&" is not valid in Global Test Data sheet","")
57     ExitGlobalIteration
58   End If
59
60   'Selecting the Row where used data is empty
61   For itr = 1 To DataTable.GetSheet("Individual").GetRowCount
62     If DataTable.Value("UsedData","Individual") = "Y" Then
63       DataTable.GetSheet("Individual").SetNextRow
64     Else
65       Exit for
66     End If
67   Next
68
69   'Logger
70   Call writeReportLog("<b><font color='blue'> Use Case Scenario : Creation of '" & loanType & "' for '" & customer_Type & "'</font></b>","Passed",FailComments,"")
  
```

AT Tests

```

class CreateDirectLoan : Test
  ♦ %TestInfo[Desc="Createa direct Loan",Group="Loan Creation",Table=""]
  ♦ parameter String RowID
  Main()
    if (isDirectLoanApplicableForCurrentState())
      ♦ String SuiteName = r_DirectLoan.SuiteName
      ♦ String DirectloanType =r_DirectLoan.Loan_Type
      ♦ String customer_Type = r_DirectLoan.Customer_Type
      ♦
      ♦ //if Suite To Execute and Suite Name is equal to Execute test case
      if (sSuiteToExecute.ToUpper() == SuiteName.ToUpper())
        ♦ LogError("Halt the Global Iteration")
        ♦
        ♦ //Customer Type is not validate exit the current Test Iteration
      if (!(customer_Type.Trim() == "Joint" || customer_Type.Trim() == "Cosigner" || customer_Type.Trim() == "Individual" || customer_Type.Trim() == "Joint_Cosigner"))
        ♦ LogError("Customer Type: {customer_Type}is not valid in Global Test Data sheet")
        ♦ LogError("Halt the Global Iteration")
        ♦
      for (Integer i =1; i<=lr_Individual.Count(); i++)
        if (lr_Individual[i].UsedData == "Y")
          ♦ // do nothing (Update need to be done)
        else
          ♦ break
          ♦
      ♦ //Logger
      ♦ Print(" Use Case Scenario : Creation of {DirectloanType} for {customer_Type} ")
  
```

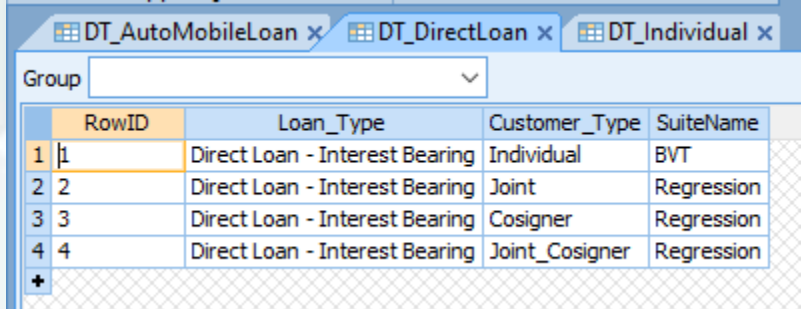
Challenges

- Indentation of Converting UFT Tests to AT Tests.
- Handling parameters.
- Converting test data and passing the input values from the data table to the test.

DataTable to DataTable

UFT has an inbuilt datatable which has local or global sheet parameters. These are converted to AT inbuilt datatable using the product 'Data Editor' of Zeenyx. This creates a Datatable of the same format with the column names and the values. Here in AT, a unique column is automatically created as "RowID" which enables the test to take different set of parameters (Data Testing) input as shown in the example below:

	A	B	C
1	Loan_Type	Customer_Type	SuiteName
2	Direct Loan - Interest Bearing	Individual	BVT
3	Direct Loan - Interest Bearing	Joint	Regression
4	Direct Loan - Interest Bearing	Cosigner	Regression
5	Direct Loan - Interest Bearing	Joint_Cosigner	Regression



The screenshot shows the AT Data Editor interface with three tabs: DT_AutoMobileLoan, DT_DirectLoan, and DT_Individual. The DT_DirectLoan tab is active, showing a datatable with columns: RowID, Loan_Type, Customer_Type, and SuiteName. The data rows are:

RowID	Loan_Type	Customer_Type	SuiteName
1	Direct Loan - Interest Bearing	Individual	BVT
2	Direct Loan - Interest Bearing	Joint	Regression
3	Direct Loan - Interest Bearing	Cosigner	Regression
4	Direct Loan - Interest Bearing	Joint_Cosigner	Regression

Conclusion:

Over years of effort built over UFT project which is currently unstable/unreliable to maintain larger tests can be migrated quickly and efficiently to AscentialTest using the utilities created by Matryxsoft in a very short span of time which improves the reliability and quality of the tests in turn improves the software quality, the utmost priority for one and all.

For more information on migrating UFT to AT test automation tool.
 Visit Matryxsoft Tech at www.matryxsoft.com.