www.qatestlab.com
contact.ua@qa-testlab.com

Automated QA TestComplete Script

PROJECT A

General information	
Customer	<Project name>
Created by (Author)	
Preparation date	
Version	
Status	

Revision History					
Version	Description	Author	Date	Approved by	
				Author	Date

Summary

1. General information

The provided script demonstrates the test automation of native applications for Windows by using the means of TestComplete.

Imagine that we have an archiver and several its compression algorithms are under performance optimization. The activities should not influence the final algorithm output. In consequence, checking each product version, we should make sure that:

- after unpacking the archive affected by an optimization algorithm, the files fully correspond to their originals
- archive-output of algorithm coincides with some control sample. The sample was checked manually by using special means and it was confirmed to be a correct algorithm output.

The provided script performs the checking of exactly such a type by interacting with the application under test via GUI and therefore realizing one of the possible end2end scenarios for the product of this type. 7 zip is the application under test.

As in course of time, another archive type can require testing and the test steps do not depend on the type of archive under test, the decision to remove the name of archive extension from scripts code was made, realizing data-driven test (DDT). To store the tested archive types, the storage of TestComplete project variables of a table type is used.

2. Test script

```
var TEST_DATA_FLD_PATH = ProjectSuite.Path + 'test_data\\';
var FILES_TO_COMPRESS_PATH = TEST_DATA_FLD_PATH + 'files\\';
var TEMP_DIR_PATH = Sys.OSInfo.TempDirectory + 'demo\\';
var DEF_TIMEOUT = 5000;

function main(testData) {
 // run AUT, create temp folder
 init();

 // do the test for each specified archive extension
 for (var r = 0; r < testData.RowCount; r++) {
 // get an archive extension from test data storage
 var EXT = testData.Item(0, r);
 //make archive file name
 var archiveName = EXT + '.' + EXT;
 Log.Message('Tested archive format: ' + EXT);
 var autWnd = Aliases.autProc.autWnd;

 // go to the folder with files to put into an archive
 autWnd.workArea.navBar.filepathTextField.SetText(FILES_TO_COMPRESS_PATH);
 autWnd.workArea.navBar.filepathTextField.Keys('[Enter]');
```

```
var content = autWnd.workArea.content;

 // select each item inside the folder, by clicking the first one and simulating a
click on
the last one holding SHIFT key
 content.ClickItem(0);
 content.ClickItem(content.wItemCount - 1, 0, skShift);

 // add selected files clicking "add" button
 autWnd.toolbar.ClickItem("Add");
 var newItemDlg = Aliases.comprProc.addToArchiveDlg;

 // wait until the new archive dialog is shown
 newItemDlg.WaitProperty('Exists', true, DEF_TIMEOUT);

 // set archive destination path
 newItemDlg.destFilepathTextField.SetText(TEMP_DIR_PATH + EXT);

 // select required archive type, according to the value specified in test data
 newItemDlg.archiveExtSelector.ClickItem(EXT);
 // confirm the dialog
 newItemDlg.confirm.Click();

 // wait until the dialog leaves
 newItemDlg.WaitProperty('Exists', false, DEF_TIMEOUT);

 // compare newly created archive with baseline
 var isArchivesEqual = aqFile.Compare(TEMP_DIR_PATH + archiveName,
TEST_DATA_FLD_PATH +
archiveName);
 expect(isArchivesEqual, "Newly created archive is equal to baseline");

 // go to the folder with new archive
 autWnd.workArea.navBar.filepathTextField.SetText(TEMP_DIR_PATH);
 autWnd.workArea.navBar.filepathTextField.Keys('[Enter]');

 // open extract dialog for the archive
 content.ClickItem(archiveName);
 autWnd.toolbar.ClickItem("Extract");

 // uncheck flag that is for unpacking to subfolder, if it is checked
 var extractDlg = Aliases.comprProc.extractDlg;
 if (Aliases.comprProc.extractDlg.extractToSubfolderOption.Enabled) {
 Aliases.comprProc.extractDlg.extractToSubfolderOption.Click();
 }

 // making path to the folder where files will be extracted to
 var extractToPath = TEMP_DIR_PATH + EXT;
 extractDlg.extractDestinationTextField.SetText(extractToPath);
 extractDlg.confirm.Click();
 extractDlg.WaitProperty('Exists', false, DEF_TIMEOUT);

// comparing original files against extracted copies
 var files = aqFileSystem.GetFolderInfo(extractToPath).Files;
```

```
 while(files.HasNext()) {
  var baseFileObj = files.Next();
  var baseFileName = baseFileObj.Name;
  var baseFilePath = baseFileObj.Path;
  var testedFilePath = extractToPath + '\\' + baseFileName;
  if (expect(aqFile.Exists(testedFilePath), baseFileName + '. extracted from
archive')) {
  expect(aqFile.Compare(testedFilePath, baseFilePath), 'Tested copy of the"' +
baseFileName
+ '" file is equal to baseline');
  }
  }
  }
}
function init() {
  aqFileSystem.CreateFolder(TEMP_DIR_PATH);
  TestedApps.aut_7zip.Run();
}
function finalize() {
  aqFileSystem.DeleteFolder(TEMP_DIR_PATH, true);

TestedApps.aut_7zip.Close();
}
function expect(expr, description) {
  var msg = "Expectation: " + description + '. Result: ';
  if (expr) {
  Log.Checkpoint(msg + 'passed');
  } else {
  Log.Warning(msg + 'failed');
  }
  return expr;
}
```

3. Execution results

The results of script execution are available from a specific tool menu but they can also be exported as a web page. A screenshot of the results is provided below