* + 1. Introduction

Access Testing is working with <client> to develop a long term partnership where we provide expert advice and guidance through all phases of the software development lifecycle. We are currently working with <client> to understand how testing is conducted and at the same time design a test automation framework to assist the department maximise their testing efficiency.

A test automation framework sets the rules and standards which should be used to guide the test automation. It is an integrated set of components such as function libraries, reusable modules, test data sources, etc. which provides the basis of test automation and simplifies the automation effort. However, what it doesn’t provide is assurance that when implemented it will be adopted by the testing divisions and the selection of tests being automated meet the criteria for automation, have maximum test coverage and deliver a timely and high Return On Investment (ROI).

* + 1. Document Purpose

The purpose of this document is to provide <client> with recommendations and guidance on how to successfully implement and rollout a new test automation framework to ensure it is adopted by the different testing divisions and the benefits are fully realised.

* + 1. Implementation Steps
    2. Design Framework
    3. Identify Testing Scope and Requirements
    4. Identify and Design Business Layer
    5. Identify and Design Data Store
    6. Develop Framework
    7. Deliver With Training and Education Sessions

Documentation

Keywords with their expected UI-Objects and parameters will be documented using this format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Keyword: | Click | | | |
| Category: | UI Testing | | | |
| Description: | Click works with UI based objects by receiving either an Object reference or an object type to click on. The object must be visible on the current screen to succeed. | | | |
| Use map: |  | | | |
| UI\_Object | | | | |
| <OR Reference> | | User defined Object reference from the UFT O.R. or alternate O.R. | | |
| <Object Type> | | One of: [button], [image], [link] | | |
| Parameters | | | | |
| <Label> | | Object label on AUT UI, eg Login, Next, Search etc | | |
| Keyword Sheet Examples | | | | |
| Keyword | | | UI\_Object | Parameters |
| Click | | | btnLogin |  |
| Click | | | [button] | Next |

|  |  |  |
| --- | --- | --- |
| Technical Implementation | | |
| Stored in File: | cls.Actions.vbs | |
| Global Variables | | |
| Global Timeout |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Keyword: | Math | | | |
| Description: | Math works by taking parameters and functions to perform an eval on and store the result for later use | | | |
| Use map: |  | | | |
| UI\_Object | | | | |
| N/A | |  | | |
| Parameters | | | | |
| <Parameter> | | One of the predefined functions for the custom solution. rndBirthDate ( <minYears>, <MaxYears>). Age between min and max  currencyConvert(fromCountry , toCountry , value) converts values to/from au | | |
| Keyword Sheet Examples | | | | |
| Keyword | | | UI\_Object | Parameters |
| Math | | |  | rndBirthDate(14,17) |

Version control

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date | Who | CR Reference | Validated | Approved | Merged into |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| STEP\_ID | ACTION\_NAME | RUN | UI\_OBJECT | Parameter | DATASHEET | ITERATIONS | ON\_FAILURE | Comments |
| 10 | getTemplate | FIRST |  | Template.xml | N/A |  | ExitTest |  |
| 20 | Login | TRUE |  |  |  |  | ExitTest |  |

1. Introduction

Deloitte is working with <clientname> to develop a long term partnership where we provide expert advice and guidance through all phases of the software development lifecycle. We are currently working with <client> to understand how testing is conducted and at the same time design a test automation framework to assist the department maximise their testing efficiency.

A test automation framework sets the rules and standards which should be used to guide the test automation. It is an integrated set of components such as function libraries, reusable modules, test data sources, standards for names, coding and code management etc. which provides the basis of test automation and simplifies the automation effort.

* 1. Document Purpose

The purpose of this document is to provide DIBP with a new test automation framework design.

# Glossary of Acronyms

| Acronym | Full wording |
| --- | --- |
| ALM | Application Lifecycle Management. HP product for defect and test management, including automation testing. |
| AUT | Application under Test |
| CM | Configuration Management |
| HP | Hewlett Packard |
| KWD | Keyword Driven. An automation methodology to drive business processes with business terminology. For example: Logon, Hire Personnel, Lodge Application. |
| QC | Quality Center, produced by HP. QC is now known as ALM |
| QTP | QuickTest Pro. HP Automation test tool. QTP is part of UFT. |
| SOAP | Lightweight messaging API to send and receive messages. |
| UFT | Unified Functional Testing. HP Automation Test Product. Formerly known as QTP. HP have combined Service Test with QTP in this tool. |
| UI | User Interface. Software interface designed to interact with user to perform tasks or report on business activities. |

# Glossary of Terms

| Term | Meaning |
| --- | --- |
| Test Plan | HP ALM test case creation and storage area in ALM. |
| Test Lab | HP ALM test case execution and results storage area in ALM. Includes links to Defects relevant to the test executed. |
| Target System | Refers to the system under test and the environments used. |
| Burnt Record | Some test activities result in changes to a given record that results in a false failure when re-used in either a new execution of the same test or a different test. Example: Registered application is reported deceased. |
| Data Driven Test | Test designed to repeat on a number of rows of data. Each row contains different data and expected results for a given sequence of steps. |
| Iteration | An execution of a test script. The iteration may be one or many repeated operations. |

1. Strategic Test Automation Framework Design
   1. Test Automation Success Criteria

Automation frameworks achieve good ROI by meeting the following criteria:

* Configurable,
* Centralised storage,
* Portability (physical, virtual, cloud),
* Unattended launch,
* Low maintenance,
* Lean Principles, and
* Extensibility
  + 1. Configurable

The scripts are expected to be able to read data from various sources which enable flexible operation on demand to meet the configurable criteria. The environment is supplied by the user or from an automation process before launching the test suite. See Naming Conventions for more detail.

* + 1. Centralised Storage

Storage of test artefacts, such as files or data needs to be accessible from the execution environment. Centralised storage use increases test portability and is discussed in more detail in Storage Structure.

* + 1. Portability

Scripts can be executed on any platform, able to be re-used on multiple environments or run at geographically distant locations without special configuration.

* + 1. Unattended launch

Scripts can be scheduled or linked to automated development activities. Automating test suite executions outside work bandwidth hours provide efficiencies though utilising work bandwidth hours for review, defect management and test script maintenance.

* + 1. Low Maintenance

Externalising data elements (i.e. not hard coding of links, servers, constants etc.) and use of advanced dynamic object locating techniques significantly reduces framework maintenance and extends the lifespan.

* + 1. Lean Principles

Lean principles are well established operational management practices to reduce assets held in stock and reduce costs. The principle drives priorities around highest business value, return on investment and a reusable code base to achieve successful testing.

* + 1. Extensibility

The ability to extend a framework without rewriting the code base is essential to achieving ongoing success over the framework lifespan.

* 1. Test Automation Architecture

Test Automation designs abstract the keyword process from the application under test, thereby isolating the keyword from the specifics of the implementation. The framework in Figure 5 demonstrates layered abstraction. The abstracted layers are:

* Test Definition
* Test Execution
* Test Adaption



Figure 1 – Test Automation Block Diagram

* + 1. Test Definition Layer

Test definition layer contains test cases, test configuration and test data. The location of the dynamic data sources is transparent at this layer.

| Component | Description |
| --- | --- |
| Test Case | ALM Test case with the Keyword Sheet attached. |
| Complex Test Case | Test case for high complexity tests where the Keyword framework is unable to handle. Complex test cases may optionally re-use the keyword framework to reduce coding and maximise re-use. |
| Data and Configuration | Test configuration, test data, execution logs and results. |

The framework test definition is expressed using keywords in a table as shown in Figure 2.



Figure 2 – Sample Framework Test Definition

* + 1. Test Execution Layer

Test Execution Layer contains the test controller which executes tests, imports data sheets and collects data from databases where required and hands to the keyword and low level drivers. Execution layer supports extensibility both for keywords and supporting functionality.

| Component | Description |
| --- | --- |
| Test Driver | The controller function that reads and executes keywords. |
| Keyword | The underlying function call that performs the actions of the keyword |
| Execution | Code execution from the keyword and supporting components |
| Error Handler | The component that manages, logs and controls flow when errors are encountered |

|  |
| --- |
| Test Driver Pseudocode:  Read initialisation-data  Read target-environment  Download Keyword File  Open Keyword File  For each Row in Keyword File  Read tab  Load datasheet  Execute Keyword  Write Results  Next  Clean up |

|  |
| --- |
| Error Handler Pseudocode:  Read ON\_FAILURE value  If value is ‘ExitTest’ then  exitTest  else  HandleError  End if |

Initialisation Data

Initialisation Data maps storage locations for the test driver. Naming conventions drive selection of databases, ALM or file systems and affect framework operation.

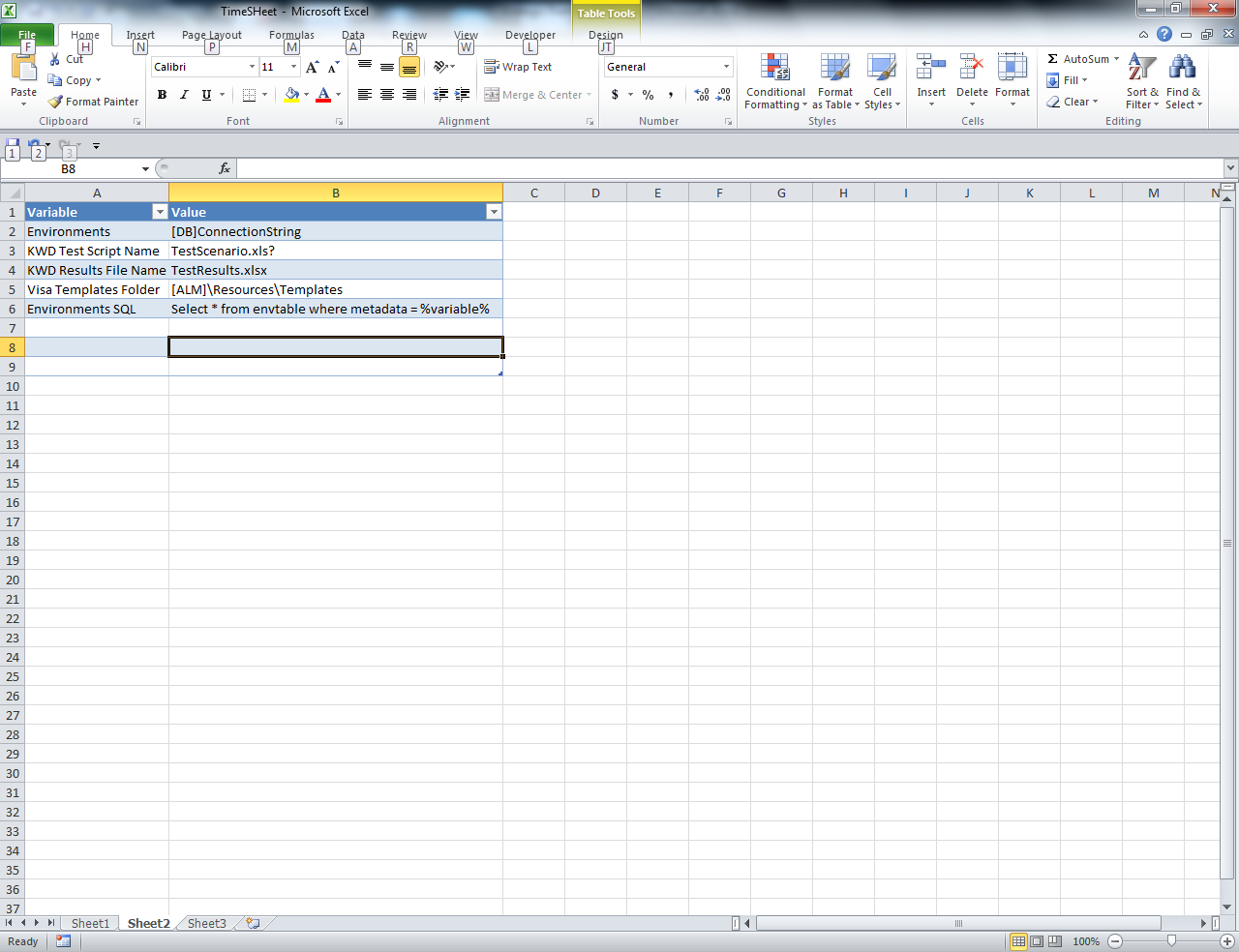


Figure 3 – Sample Initialisation Data

The sample initialisation data sheet in Figure 3 above demonstrates data which sets the controllers’ mapping to files, ALM and databases used in the testing framework. The environment to test is expected to be provided through a parameter read when launched.

* + 1. Test Adaption Layer

The adaption layer interacts directly with the interfaces and objects to run the tests, captures results and logs all actions performed by the test.

| Component | Description |
| --- | --- |
| AUT | Application under test. Such as UI, Databases, and active API’s |

At the test adaption layer, the low level drivers are used to connect to and interact with the AUT. The framework handles the process by:

|  |
| --- |
| ICSE Pseudocode:  Connect to ISCE-Interface  Queue File  WaitForProcessComplete  TestCompleteMessage |

* 1. Name Conventions

Automation frameworks use names to match business processes with coded test solutions. Significant problems occur when naming conventions are either not established or followed. The conventions discussed are utilised in the Test Automation Architecture samples below.

* + 1. Environment Identifier

<client> has multiple test environments, each requiring specific configuration and data. An environment name is used to identify:

* the test environment (s) to connect to
* folders for environment specific templates in ALM resources
* data for specific test cases

The environment identifier length is restricted to the maximum ALM folder length and should be between 2 and 8 characters. Testing targets API’s as well as UI’s, requiring interface differentiation between connections and any log in credentials where applicable. DIBP environment identifier takes the form of:

E6

Environment identifier use for templates is discussed in paragraph 2.3.4, where E6 represents an environment identifier. Binding a location folder name (E6) to an environment identifier (E6) improves automation processing and fault finding.

* + 1. Keyword conventions

Keyword Format

Effective keywords have a consistent appearance with a fairly self-explanatory name and not excessive in length. Keywords can be formatted to a number of standards such as:

* Camel Caps, or
* Underscored

|  |  |
| --- | --- |
| Convention | Example |
| Camel Caps | getStockType |
| Underscored | get\_stock\_type |

Camel Caps is shorter than underscored and is used in the remainder of this discussion.

A reasonably understandable keyword length is usually between 10 and 30 characters, with a maximum length of 40 characters. Keywords can be as short as five characters, for example, Login.

Camel Caps convention is represented in a formatting flow diagram in Figure 4. Starting from the left, set and get are in optional paths and the capitalised names are concatenated until either the length reaches the maximum length or the name is complete.



Figure 4 - Camel Caps Format Flowchart

Creating Functional Keywords

Keywords are related to business process or a lower level action. To make the keyword reusable, it optionally accepts objects and/or parameters.

Updating a data template, the designers make a keyword for each type of update, or use one keyword that accepts parameters to re-use the keyword, regardless of what is being updated.

A keyword that retrieves a new account number takes the form:

getNewAccountNumber

To differentiate passport number formats, the keyword requires a parameter to identify the issuing country. The framework implements this keyword with the parameter AU representing the issuing country.



Figure 5 - Sample Keyword Use

* + 1. Test Data

Data for testing purposes can be set up and extracted from various sources, including text, spreadsheets and databases.

Some activities can burn records, rendering the data unusable for further tests. A burnt Record flag schema may be required in the database structure to ensure test data is not re-used by subsequent tests. The data needs to link to the test cases through naming convention, in part driven by the data template type, test identifier (from ALM, the Test Case ID) and, optionally, the target test environment.

Test Data analysis is needed to establish framework conventions for test data.

* + 1. Template Files

There are approximately <template count> currently in use. Examples of existing messages stored in ALM Resources provide a working template for use in testing these insertions and updates. The file and folder naming convention enables automation to develop a solution to locate and download the file based on:

* Tempalte Sub Class
* Number of applicants in template
* Environment
* Other parameters as required.

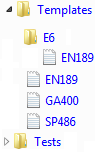


Figure 6 - Handling Environment Specific Templates

Naming the updates templates to show the number of applicants is accomplished by suffixing the name with an underscore and the number.

The framework handles the structure by reading the parameters for the environment and template location (initialisation data) to select the environment specific template:

|  |
| --- |
| getXMLTemplate Selection Pseudocode:  List template sub-folders  If sub-folders contains environment and  environment\template-name exists then  Change template-folder to environment  End if  Read template value  If tempalte-name exists then  Download template  Load file  Else  Raise Error File-not-found  End if |

* + 1. Databases

Using [DB] embedded in an initialisation variable drives the framework towards a database as opposed to ALM or file system storage. The framework handles this by:

|  |
| --- |
| Data Source Selection Pseudocode:  If Environment-location contains ‘[DB]’ then  Set environment-source = openDB  elseif Environment-location contains ‘[ALM]’  Set environment-source = ALM  Else  Set environment-source = file-system  End if  environment-source.DBQuery stringQuery |

* + 1. Framework Script Standards

Standardising the framework scripts improves readability and maintainability. Software engineering practices for revision control, commenting and Code style are discussed in detail here.

Revision Control

A revision control tool with functionality to control updates through branching, version rollback and managed code merges is strongly advise to help recover from errors getting into the framework. ALM has built in limited revision control, use of a git, mercurial or subversion based version control system is recommended.

Commenting

Commenting for code is essential to describe the components intended function and parameters. Function description comments are placed before the function or subroutine and take the form:

|  |
| --- |
| ‘@FUNCTION: Click  ‘@DESCR: Takes one or two parameters from the keyword sheet  ‘ and works out how to click on the designated object  ‘@PARAM: UI\_Object – object name (btnLogin) or type name eg: [button]  ‘@PARAM: Parameter – label on object  ‘@RETURN: 0 = Pass, 1 = Fail  ‘@VER: 0.1 |

For clarity, optional comments may be embedded in the code to explain algorithm used.

Code

Coding standards ensure readability and maintainability. Indents, spaces and returns (aka whitespace) reveal the structure. For example, this code shows the use of whitespace to reveal structure of an IF statement. Optional commenting may be added if the function is complex.

In a similar fashion to Keyword creation, function names are built using the control flow chart in Figure 4. Classes utilising Properties are exempt from get and set prefixing.

|  |
| --- |
| Function myFrameworkKeyword ( byVal UI\_Object , byVal Parameter )  If UI\_Object <> “” then  ‘If object is named  Else  ‘handle no UI\_Object  End If  End Function |

Use of scoping such as Private, Default etc. is determined by the purpose and intent of the function. Keywords are automatically default to public.

Classes are also coded with similar use of whitespace and commenting. Classes for ADO or API connections are useful because clean up and connection closing is handled automatically through the terminate event.

|  |
| --- |
| Class name  Sub class\_initialise  ‘init code here  End Sub  Sub class\_terminate  ‘terminate clean up here  End Sub  Sub run  ‘code to run here  End Sub  End Class |

* 1. Storage Structure

The selection and use of storage has a direct impact on maintainability, test stability, portability and re-use. Test Automation framework storage applies contextually to:

* File Systems
* ALM
* Databases

The data update frequency and logged granularity required provides guidance on which storage is the appropriate choice for a framework to meet the strategic criteria. Figure 8 applies framework storage to DIBP systems, with most artefacts stored in ALM. A database is used to maintain dynamic test data.



Figure 8 - Artefact Storage

* + 1. File System Storage

File systems:

* Can be disrupted by users working on a file system. Eg accidental removal or move of a folder
* Inaccessible for certain users, eg user-specific file paths
* Lack control granularity for content changes, ie version tracking

File system storage is acceptable for a temporary execution scratch pad.

* + 1. ALM Storage

ALM’s storage ensures accessibility within the client’s site, enables version control and manages accessibility. The framework has access to all the <client> ALM storage spaces ensuring audit trails are maintained when executing automated tests.

ALM storage spaces are:

* Resources
* Test Plan
* Test Lab

Resources

Resources are any files utilised by tests, such as automation function libraries, Object Repositories, templates and data. Data stored in ALM is harder to maintain but is acceptable if relatively static. Resources can be stored in a folder structure giving quick reference to users of the test architecture. Resource structure sample Figure 9 demonstrates how to organise the resource.

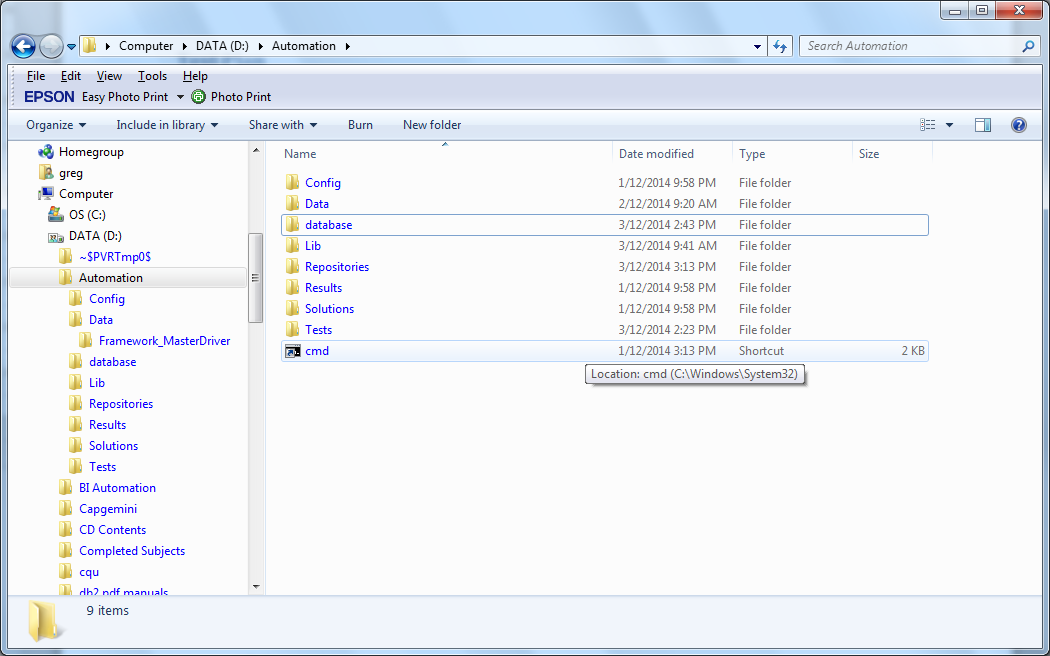


Figure 9 - Sample Folder Structure

Files can be stored in Excel or XML format in the resources. <client> use of MS Office ensures the test staff can perform maintenance on stored files as required.

Test Plan

Test plan stores test cases, automation scripts and attachments. A KWD Test is attached to the test case along with localised data in tabs. The test is accessible to be downloaded and read by the test driver when the tests are executed.

Test Lab

Test lab stores Test Instance Run and Step data, results attachments (including images and files) and automation logs from UFT, linked defects. Test Lab structures promote test case re-use in test suites. When a test is executed from ALM, the driver must locate the attached Keyword File before executing the steps:

|  |
| --- |
| Test Driver Keyword File Pseudocode:  With currentRun  Download-Attachment testScriptName  load testScriptName  End with |

Database Storage

A database is more flexible and enables quick updates to test data; logging ensures a change audit trail is maintained. Using a custom database schema for testing meets centralised storage and portability criteria. These data stores are easy to maintain and log changes if required.

ADO connectivity enables the framework to access a variety of data sources, using this psuedocode:

|  |
| --- |
| ADO Pseudocode:  ADO.ConnectionString = DBConnectionString  ADO.DBQuery SQLStatement  Return ADO.Record(1) |

* 1. Keyword Framework
     1. Functional Map

Functional complexity limits keyword driven framework capability, where native/simplified scripting takes on more complex test activities. The intention of the map is based on testability and keyword limitations expected to be encountered during development phase. Table 1maps the expected functionality level for native and keyword levels within the context of the application under test.

Table 1 - Test vs Functional Limits Map

|  | Script coding | Keyword Driven |
| --- | --- | --- |
| API | 🗸 | Limited |
| DB | 🗸 | Limited |
| UI | 🗸 | 🗸 |
| XML | 🗸 | Limited |

Limited in this context means the automation process will handle several steps within the keyword. This simplifies using the specific API’s and makes keywords more useful. For example, a keyword to select a data type would:

* Download the template, and
* Read template into the handler

Table 2 below maps the Functional areas of Business, UI, XML etc., to the keywords for the required functionality. Math keyword is a generic item used to manage calculations such as dates or currency.

Table 2 - Functional Area to Keyword Map

| Functional Area | Keywords |
| --- | --- |
| Business | Logon, Logoff, lodgeAppplication, fillScreen, getApplicationTemplate, setTemplateData |
| UI | set, get, click, getStaticText, checkpoint, Synchronise, Navigate, OpenBrowser |
| XL | - |
| XML | selectElement, updateElement |
| API | queueMessage, waitForComplete |
| DB | dbQuery, checkpoint |
|  | Math |

* + 1. Keywords

Keywords used by the simple test sheet enable business users to create their own automated test cases. The list in Table 3 is not exhaustive and additional keywords maybe identified during implementation.

Table 3 - Keyword Description

| Keyword | Description |
| --- | --- |
| checkpoint | Checks resultants against expected values, writes pass/fail. |
| click | Clicks buttons and links |
| dbQuery | Sends Query to database |
| FillScreen | Fills the UI screen with mandatory data from data source |
| get | Gets the value for TextBoxes, DropLists, RadioButtons and Checkboxes. |
| getResponse | Retrieves and scans return message for errors. |
| getStaticText | Extracts text in UI screens not in an editable location. |
| getDataTemplate | Identify the template to download and prepare for sending. |
| lodgeApplication | Submits Application of specified type with message to API |
| Logoff | Logoff and close the interface |
| Logon | Logon to the UI |
| Math | Add, subtract or run a calculation on currency or date. |
| navigate | Navigate Browser to supplied URL |
| openBrowser | Opens a browser window for testing |
| queueMessage | Sends the file to the API |
| screenFill | Fill the screen with mandatory default values |
| set | Sets the value for TextBoxes, DropLists, RadioButtons and Checkboxes. |
| setTemplateData | Assumes *getTemplate* and *getTemplateData* was used in the test in an earlier step. Changes the value if the node is found |
| Synchronise |  |
| WaitForComplete | Waits for API to respond to queueMessage with completion or error message. |

Additional business oriented keywords identified during implementation phase. Two examples are:

* *getPassportNumber*, a keyword that accepts a parameter for the issuing country, and returns a value formatted to that country’s specifications.
* *setTransactionData*, changes details such as Date of Birth, address or change the answer to a questionnaire, after *getTransactionTemplate* retrieves a valid transaction and *getTrasnactionApplicant* selects a valid applicant.

Implementations for Click and setTransactionData are demonstrated in these pseudocode blocks:

|  |
| --- |
| Click Pseudocode:  Scan UI for labelname  If labelname is-in UI then  Object(labelname).Click  Else  Raise error object-not-found  End if |

|  |
| --- |
| setTransactionData Pseudocode:  if selected-transaction is invalid then  raise error no-transaction-or-node-selected  end if  if selected-transaction.node exists then  selected-transaction.node = newValue  else  raise error transaction-node-invalid  end if |

* 1. Sample API Test

Figure 7 below shows a sample API test uncovered during the client site discovery phase. The overview shows the steps to replicate in the framework:

1. Reading the template and data from the sources,

2. Applying updates to the message,

3. Storing the message as an audit trail,

4. Sending the message and receiving confirmation,

5. Testing the database contains sent message data, and

6. Write the results back to ALM.



Figure 10 - Sample API Test

The test shown is the basis for selecting API as a high value return on investment due to the volume of tests based on Transaction subtypes alone. The database tests are expected to have similarities and differences need to be added as small modules to supplement the general approach.



Figure 11 - Test Framework Component Relationship

* + 1. Sample Test Case

The sample API test is implemented using the keywords in Table 3, shown in Figure 12. The test definition keyword sheet .



Figure 12 - Sample KWD Sheet

Step 10 – selects Transaction Type GA400 template from the designated template storage.

Step 20 – Focuses on specified applicant’s data

Step 30 – calculates a random birth date between 14 and 18 years of age

Step 40 – changes applicants DoB to the calculated value from step

Step 50 – changes applicant’s country to UK

Step 60 – changes applicant’s City to Leicester

Step 70 – queue the message, wait for completion and confirm the response.

Step 80 – queries the database to confirm the message was processed and is present in the database

Step 90 – writes the final results back to ALM.

* 1. Identified Modules

Table 4 maps the required low level functionality to support the framework within the execution and test adaption layer. Some functions are keywords; others support keyword functionality.

Table 4 - Modules and Functionality

| Module | Functions | Descriptions |
| --- | --- | --- |
| ALM API | UploadFile  DownloadFile  AttachFile  WriteResult | Uploads a file to ALM  Downloads a file from ALM  Uploads and attaches a file to ALM Object  Writes a Result to ALM |
| XML Message Manager | ReadFile  SelectElement  UpdateElement  SaveFile | Reads an XML file from file system  Selects nominated element using XPath  Updates nominated element  Saves XML file to file system. |
| XL Manager | Open  GetConfigurationVariables  GetMatchingRow  NextRow  Close | Open File and select a sheet Configuration Variables binds script to XL data Matching column value returns related column  Steps through an XL based Test Script |
| SOAP API Messaging | OpenConnection  QueueMessage  WaitForComplete  CloseConnection | Connect to specified API  Sends message into API Queue  Waits for the queue response |
| Database Component | DBConnect  ExecuteSQL  ExecuteProcedure  ErrorMessages  RecordCount  MoveNext  Close | Database connections  Execute SQL and get a recordset |
| Function Library | Unique Application Key  Date Calculations  Passport Number Validation  String Manipulation  Basic Numeric Calculations  Select Country of Origin Data (eg Town etc) | Initialisation  Eg age expected to be less than 18 years |
| Additional Functionality | Language/Localisation  Cross Browser Testing  Specify message for environment  Persistent parameter data  General purpose generators/counters | Additional functionality is a lower priority for the initial development phase of the test framework. |

* 1. Preparing Framework for first use
* Decide initial locations of the artefacts in ALM and the databases
* Update the initialisation variable file.
* Upload framework resources
* Load Transaction Templates for first automation batch
* Design and implement database for framework to retrieve data from.
  1. Building Framework Tests
     1. Is transaction template required and stored in ALM?
* If not, acquire a sample that meets the data requirements (eg number of applicants) and upload.
  + 1. Additional Test Data?
* If not, create the data in the test data sheets or database as applicable to the test
  + 1. Automate ALM Test Case
* Change the ALM test type to QTP Automated Test
* Open the test in UFT
* Type the automation test keyword into the new script: RunTest
* Associate this list of function libraries files with the test:
  + Keywords.vbs
  + Handlers.vbs
  + Support.vbs
    1. Write Keyword Test
* Download the keyword template spreadsheet to the local drive
* Write the automated keyword test
* Upload as an attachment to the test case
* Add a parameter to the test for Test Environment, set the value to E6
  + 1. Validate Automation Test
* Execute and validate the automated script
* Verify the test is correct and useable
* Check off the automation test as completed