**Creating resource library using TestShell Driver Builder**

**Document Agenda**

1. Library main concepts
   1. Creating new Library
   2. Driver workspace
   3. Project variables
2. Driver functions
   1. Template Functions
   2. Init
   3. End Session
3. Driver Compilation
   1. Compilation Settings
   2. Adding driver DLL to TestShell Studio

**Library main concepts**

TestShell library is a .NET DLL which develop using TestShell Driver Builder.

TestShell Studio test is a flow which can interact with different resources during the test, as a best practice approach it is recommended to create a dedicated library for each resource so the Studio test developer could manage and control the relevant resource in a comfortable way.

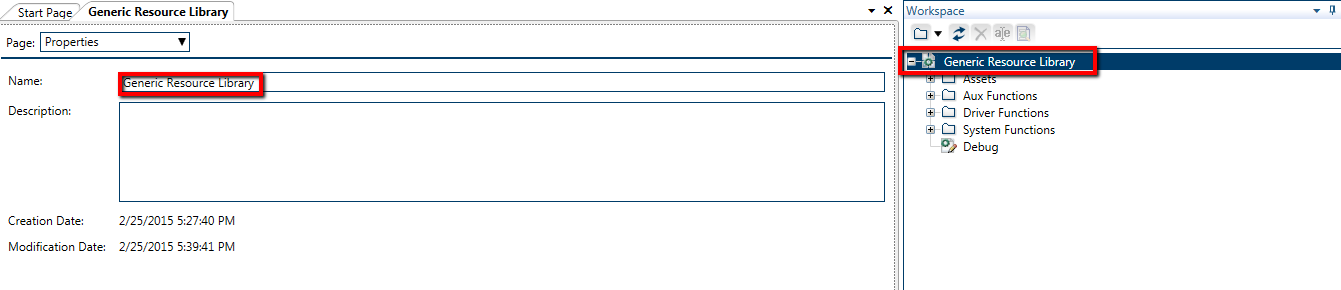
This document provides guide how to create a resource library step by step in Driver Builder and import the library DLL into TestShell Studio as a resource library.

Since each resource has different API (e.g. Telnet, SSH, CLI) – each library needs to manage its own session – according to the resource type. This document simulates the work with Telnet protocol, as working with other protocols can be done using the same idea.

The best practice would be to create a custom library for each resource, the library will demonstrate all the needed resources functionalities to the test developer.

## Creating a New library

1. Open TestShell Driver Builder (Authoring), locate under: Start🡪 All programs🡪 QualiSystems🡪 TestShell Driver Builder (Authoring).
2. From The top left button select open project and 🡪 locate the ‘Generic Resource library.tsdrvproj’ file.
3. Give name to the project – in the Workspace window double click on the ‘Generic Resource Library’, set the solution a valid name - usually the name of the resource (e.g. SDR Base Unit).



1. Save.

## The Library Workspace

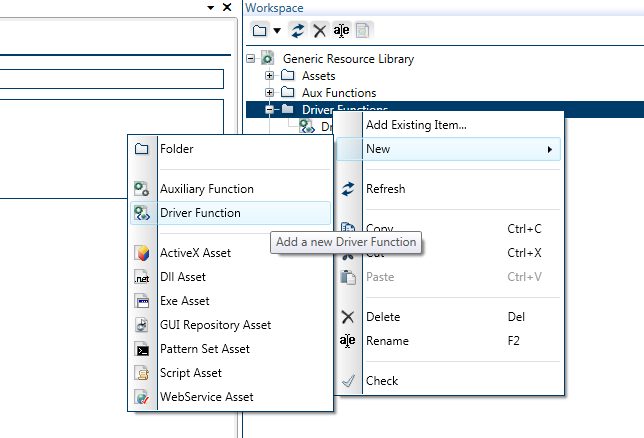
1. The workspace is the Driver Builder solution management window, Under the workspace the user can create and edit the driver entities:
   1. - Driver functions – ‘public functions’ – functions which will be expose to the TestShell Studio user as part of the library.



* 1. - Auxiliary functions –‘private functions’ –internal functions for the driver usage only – will not expose to the Studio user.

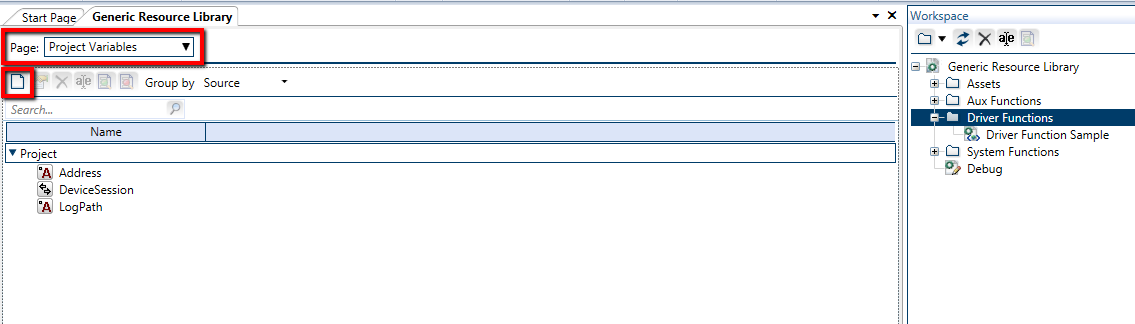


* 1. Assets – any Dll, EXE, Script which the driver\library use as part of its execution.



## Project variables:

1. Beside the function variables (variables which the library creator needs to define under each function) the Driver Builder project can manage project variables – the project variables are like ‘Global Variables’ – once a specific project variable populated with value – all the other functions in the library can use it.
2. Usually the project variables will use to hold variables which need to be used more than once in different functions – for example – the ‘resource address ,’SSH username’...
3. The best practice to manage project variables would be to initialize them when the driver is launched from Studio (Done in the ‘Init’ function- see below).
4. In order to create/edit project variables see the following steps
   1. Double-click on the Workspace root -the driver project name, (In the screenshot above, it is ‘Generic Resource Library’).
   2. From the Page drop-down list, select **Project Variables:**



1. Once the project variables defined - the driver creator could use them using the following syntax: ‘Project.VARIABLE\_NAME’ – for example, if the driver creator would like to use the project ‘Address’ variable – he should use ‘Project.Address’ variable.

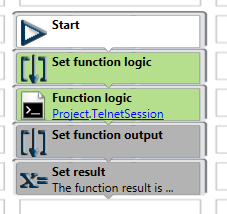
## Driver Functions

Each library should implement different set of functions – depend on the library role, as each library **must** implement 2 functions:

‘Init’ and ‘End Session’, which locate under the ‘Systems Functions’ folder.

When we would like to create custom function we could use the ‘Driver Function Sample’ function

## Template Function



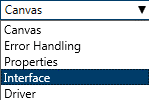
The ‘Driver Function Sample’ function locate under the Driver functions folder.

The template function is type of ‘Driver function’ meaning the Studio end user could call it from TestShell Studio.

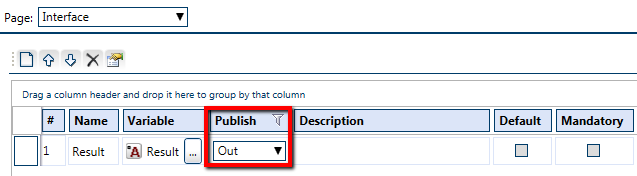
1. Assume the driver creator would like to create new function – he should copy&paste this function and change its name- for example ‘SetDeviceVersion’.



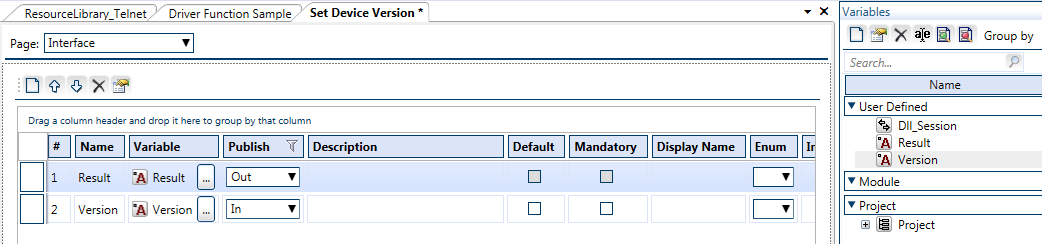
1. Function interface - The interface define the function in\out parameters, drill the Canvas dropdown and locate the interface section



* 1. The template has one variable define as the function result – this variable is type of OUT –meaning this variable populate value when the function finish its execution.



* 1. The ‘Set Value’ function should **get** version from the Studio, we can use the interface to ask for this inputs from the user:
     1. Create new variable – name: ’Version’, type: string, scalar.
     2. Drag the variable from the variable window to the to the interface window, see that the version type is ‘IN’ – now the user of this function will need to provide the version value when he would like to use the function.



* + 1. Save and open the canvas again.

1. Let’s review the basic function steps
   1. Set function logic - this part should implement the logical part of the function – since this sample present telnet session resource – probably the logic will used the Telnet session – double click on the ‘Telnet session’ step will open the Telnet window.

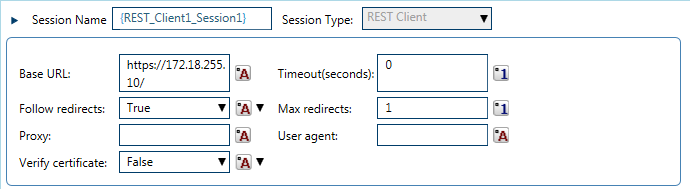
In the Telnet window you can record the requested steps, set transforms for validation, etc.

If your driver use different type of session – use it.

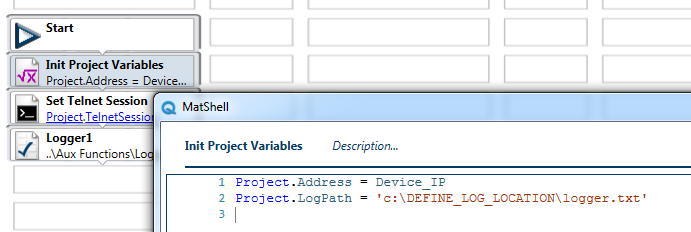
* 1. Set function output – the template has one output – the user who called this function will get this value as a result.

## Init function

1. The ‘Init’ function is a function which will called **automatically** by TestShell Studio each time **Studio** test run the driver ‘Init’ step.
2. The **Init** function **cannot** be called explicitly from the TestShell studio.
3. The ‘Init’ interface variables will be exposed to the Studio user in the driver session creation.
4. The following (example) image taken from TestShell Studio after the user drag the ‘REST Client’ library into the test canvas and open the step mini canvas, notice to all the Session input parameters: ‘Base URL’, ‘Follow redirects’ etc. – all these variables are part of the ‘Rest Client library- Init’ function and defined in the function interface as **IN parameters**.



1. Implement the function
   1. Save session parameters - The function get as inputs all the session parameters – using MathShell tool the driver creator can save these values into matching project variables, so the other driver functions could use these parameters, in our case we will ask from the user the resource IP – and will store it in the relevant variable.



* 1. Define driver log file - set the path of the driver log file, all the other driver functions would write to this file.
  2. Initiate driver session – for example the ‘Generic Resource Library’ initialize Telnet session, each driver should load the required session type – base on the resource communication API, the session should be ‘project Variable’ – once this session initialized all the other functions could use this session
  3. What else the driver must do during its initialization?

## End Session

1. Similar to the ‘Init’ function the ‘End Session’ function will be called from TestShell Studio each time the test get to the ‘End Session’ step.
2. The **End Session** function cannot be called explicitly from the TestShell studio.
3. The ‘End Session’ is recommended to clean up the driver activity for examples: close open sessions, write to the driver log etc.

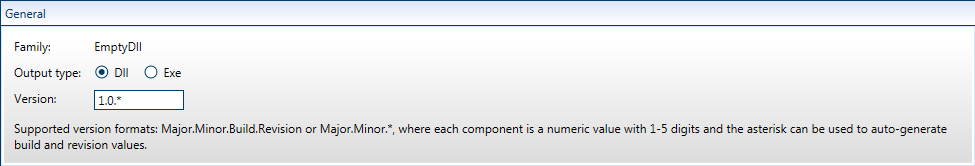
## Driver Compilation

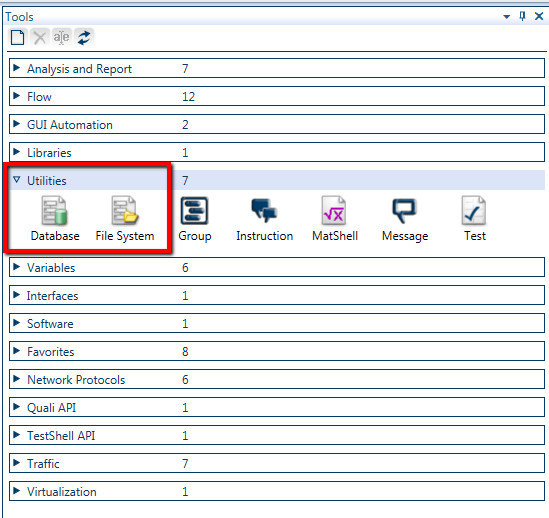
Once the library is ready the next step would be to compile it and add it to Studio as library.

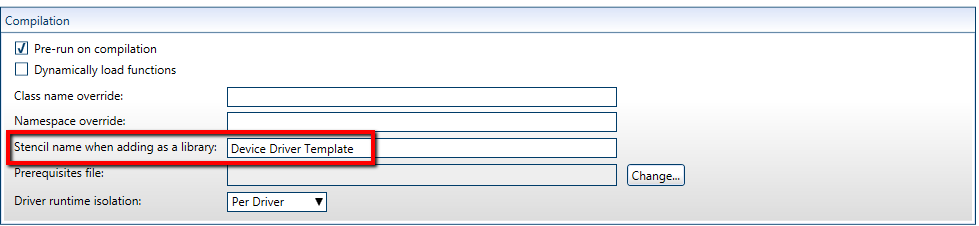
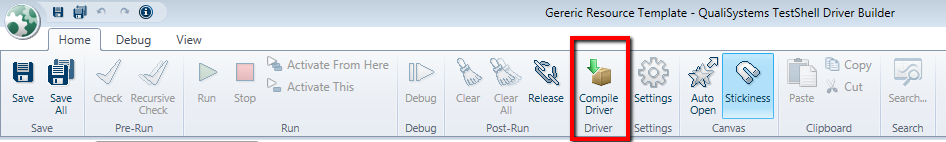
Let’s review the completion definitions:

Under the Workspace window double click on the project name – this will open the driver configuration pane.

## Compilation settings

1. DLL version:
   1. The library creator can define the driver final version.
   2. The version numbers can be specified in the following way:
      1. The user may manually specify a version number in the format Major.Minor.Build.Revision (example 1.2.3.4).
      2. The user may ask Driver Builder to automatically generate a version number by using the format Major.Minor.\* (example 1.2.\*).
   3. When option 2 is used, Major.Minor.\*, the last 2 numbers are generated in the following way:
      1. A. The third number, build, is the number of days since Jan. 1st 2013.
      2. The forth number, revision, is the number of seconds since midnight divided by 2. 7
   4. This way every time that a project is compiled you'll automatically get a new number (compilations during the same day will change only the fourth number, revision).
2. Library Stencil name:
   1. The Stencil name define where the library will be presented in TestShell Studio.
   2. See in the following image – the ‘Database’ and the ‘File System’ libraries compiled as their stencil name set to: utilities

.

* 1. In order to set the library DLL under custom Stencil define request name in the ‘Stencil name when…’ 
  2. Now we ready to compile the library – in the home ribbon find the ‘Compile Driver’ button and click it.
  3. 

1. Once the compilation process finish pop up window appear referring you to the compilation results folder, click on the ‘Here’ link – this will open the results folder. The folder contain few files but our main interest is with the DLL file.

## Adding Driver library DLL to TestShell Studio

Once the DLL is compiled the TestShell Admin can add it to TestShell Studio

Let’s review the adding library to TestShell Studio steps:

1. Open TestShell Studio (Admin user only).
2. Go to Admin tab 🡪click on the Libraries bottom.
3. Right click on the ‘Libraries’ root folder 🡪new 🡪 Library.
4. Locate the just compiled DLL file and open it.
5. Udder the Library tree locate the new DLL with blue ‘plus’ sign – It mean that the new library exist only in the current machine, in order to make it available for all the Studio users – locate the ‘Check In’ button in the button of the screen and click it- this action will place the library in the TestShell server.