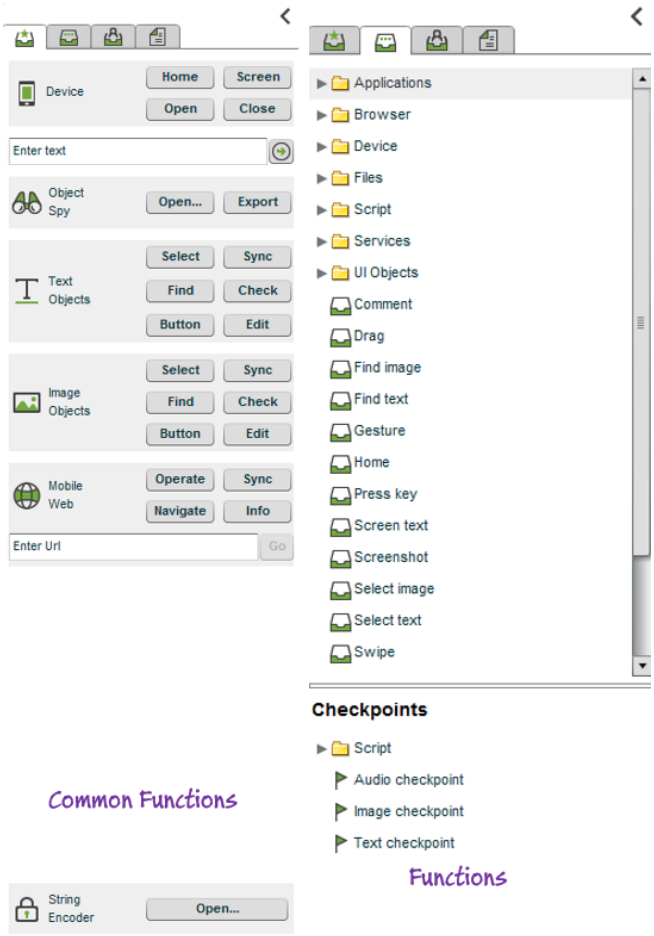
Legacy | Work with Perfecto Automation

Adding commands

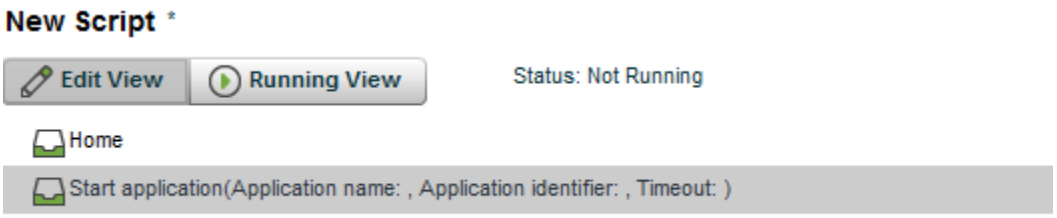
The basic steps in Perfecto Automation scripts are execution of the Perfecto functions. To add a command step to the script double-click on a function in the ***Commands Sidebar*** display.

The Perfecto functions are arranged in two tabs:



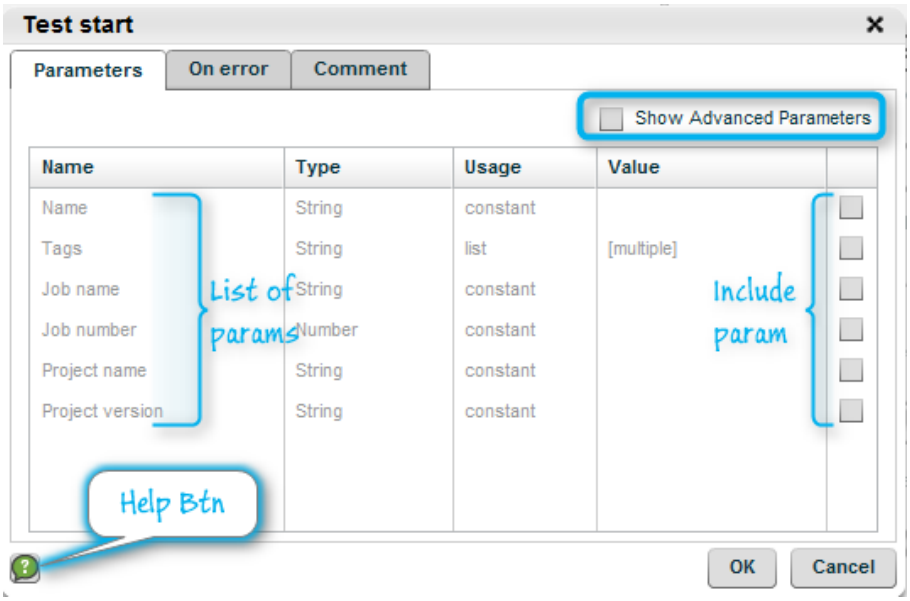
- The Common Functions tab - presents many of the Perfecto functions that are very commonly included in automation scripts, for easy access.
- The Functions tab - presents all of the Perfecto functions in a list format. Many of the functions are grouped into "folders" that can be opened to get the list of related functions.

After adding the command to the script it will appear as a new line in the Edit View of the Scripting Canvas.



Setting Function Parameters

After adding the commands to the script, set the values of the parameters for the particular function by double-clicking on the command line. This presents the Parameter window - this window is tailored to the command function that was selected, the following is just an example to highlight the main UI information:



The table in the window lists all of the basic parameters of the function. Some functions support "Advanced" parameters, that can be added to the table by checking the ***Show Advanced Parameters*** checkbox (upper left corner of window). To understand the use and range of values for the different parameters, click the Help button (lower right corner) to display the function description page from the Perfecto REST API documentation.

To set the value of one of the parameters -

- Check the parameter's checkbox - in the rightmost column
- If the parameter is checked, an input field is displayed in the **Value** column.
    - The input will be a Text entry field for most parameters.
    - For parameters that only accept an enumeration input, the field will be a pulldown menu with possible values to select.
- Enter or Select the value for each of the parameters, and click the OK button.

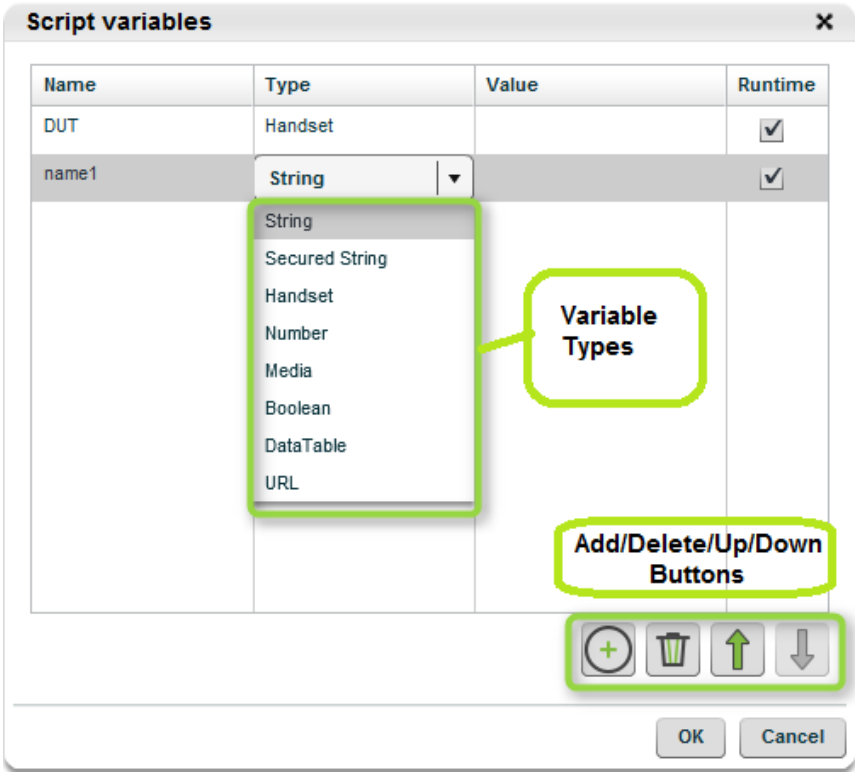
Adding Script Variables

Manage script variables from the **Variables** icon in the Automation toolbar.

Each script variable has:

- Name - any string that contains letters, numbers and the underscore character '_' , and cannot begin with a number.
- Type - script variable types are: *String, Secured String, Handset, Number, Media, Boolean, DataTable*, and *URL*.
- Value - adjusted according to the selected variable type. For example, if Type = Handset, the value will be set using the Select Device dialog box and if Type = DataTable, the value is selected from the Data Table repository.

To add/delete/move script variables, use the buttons located at the bottom right side of the window.



Runtime Variables

Runtime variables are defined by the user prior to running the script and their value may change during the execution. The value provided at definition time will be the default.
Deselect the check-box to explicitly define the variable's value which will be constant.

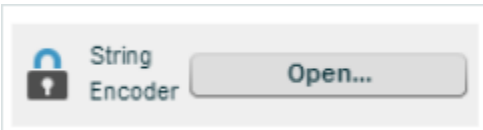
String Encoder

Encode confidential and sensitive information in scripts by encoding the text in following functions:

Type
Edit.set(image)
Edit.set(text)
Application.Element.Set
Application.Edit.Set
Application.Control.Set
Webpage.Element.Set
Webpage.Control.Set
Webpage.Edit.Set

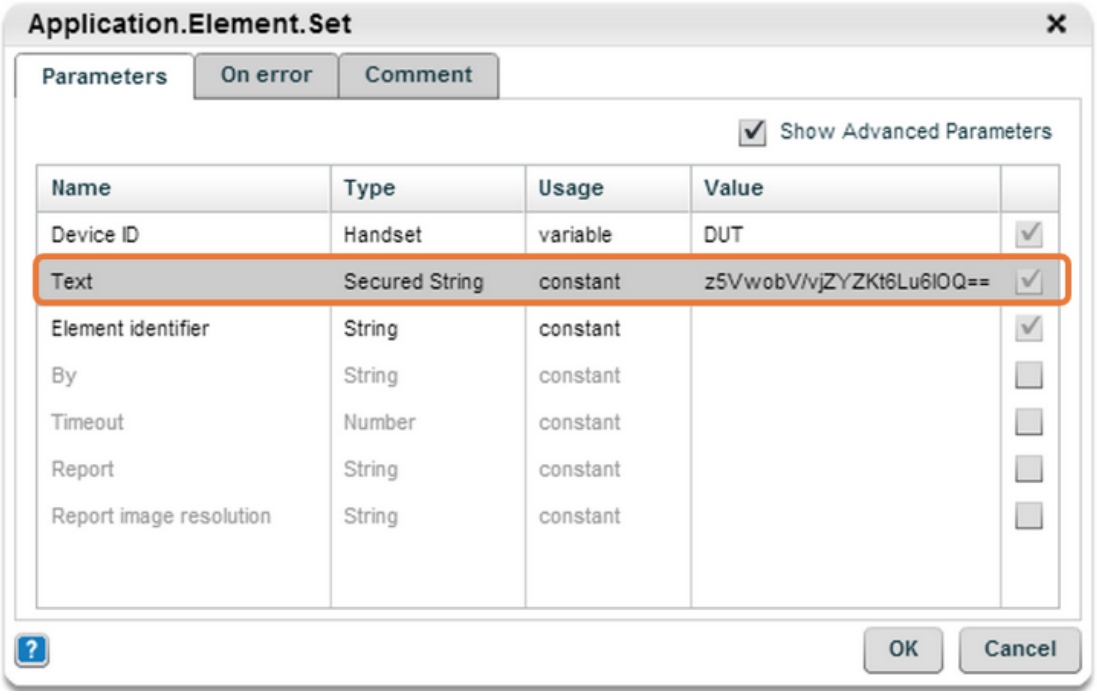
How to use:

1. From the left side bar, click on *String Encoder*.



2. Enter the text to encode, and click on the Lock icon

3. Click on the Clipboard icon to copy the encoded text.
4. In the script, click on the function(s) which will use the encoded text. In the **Text** or **Value** parameter, select Type = ***Secured String***, and paste the encoded text in the value field.



5. Run the script.
6. In the script report, the text will appear as encoded. The text will also be encoded in reports.

String Encoder and Execute Script
When using the Secured String type in a sub-script within script (execute script):

- If a runtime parameter variable in the executed script is defined as a *Secured String* type, then the variable parameter in the calling script MUST be defined as *Secured String*.
- If a runtime parameter variable in the executed script is defined as a *String* type, then the parameter variable in the calling script can be defined as string or *Secured String*. This allows the user to decide whether or not to use a secured string.

String Encoder and User Function

If a parameter is defined in a user function as *Secured String* type, when the user function is executed from within a script, the parameter type MUST be defined as *Secured String*.
If a parameter is defined in the user function as *String* type, when the user function is executed from within a script, the parameter type can be defined as either *String* or *Secured String*.

String Encoder and Selenium/Appium

When using the secured string in a Selenium/Appium automation script, prefix the encrypted string with "**secured.**". This provides indication to the Selenium/Appium Server that the string should be decrypted before transferring it to the device.

Execution Control Constructs

Conditions

To include a condition into the script, use the **Condition** icon in the Automation toolbar. Use a condition to change the flow of the script according to a set criteria.

Each ***Condition*** splits the execution to two possible branches:

- **On Success** - if the condition evaluated is successful
- **On Failure** - if the evaluated condition was an error or unsuccessful

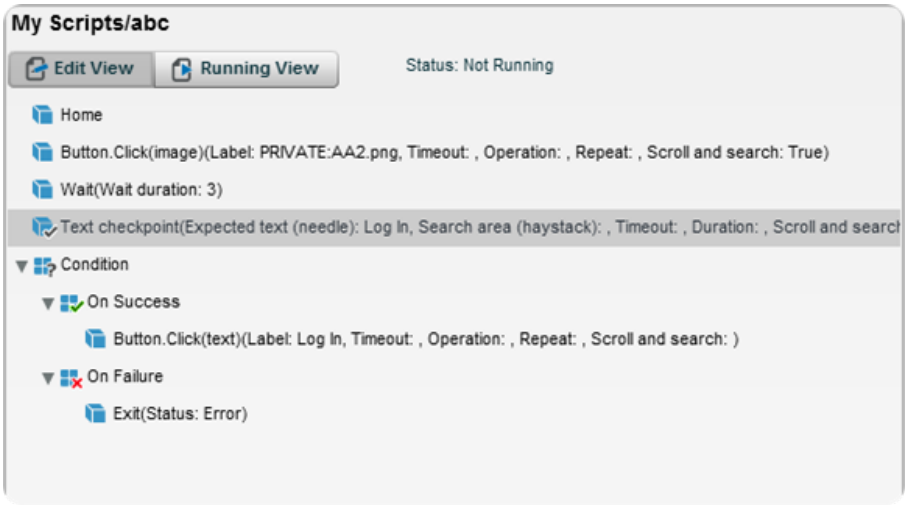
Conditions are evaluated according to the result of the preceding command, which should normally be a validation (such as ***Find*** or ***Text checkpoint***). Therefore, ensure that the function to be evaluated is located directly above the condition in the script. (Notice that the results used to evaluate the condition can be from any function, including *Wait*.)

> **Note:** When using ***Find***, the *On Error* policy (see below) is by default "Catch". When using a checkpoint, change the *On Error* policy to "Catch" rather than the default -exiting the script. This will avoid marking it as a pass or fail in the report, leaving it as the condition input only.

Either conditional set of commands (On Success or On Failure) can be left blank, and if met will cause the script to continue to the next step after the condition statement.

It is possible to enter any set of commands inside the condition path - including nested conditions and *groups* (see below).

In the following script, the ***Condition*** command uses the result of the *Text checkpoint* to either click on the Login button or exit the script with an error.



Loops and Data-tables

Use loops in the script to repeat a set of commands - either a predetermined number of times or use a data table to create iterations and keep the script data driven.To add a loop to your script use the **Loop** icon on the Automation toolbar.

You can select what type of loop control from the *Loop* dialogue:

When repeat is selected, the system will iterate the command according to the defined number. When data table is selected, the system will perform the commands and in each iteration the data of the active row in the data table will be used.

How to use:

1. From the IDE's *Repository* tab, create a data table or upload a CSV.
2. Add a data table variable to the script and use your Repository table as its value.
3. Select the data table in the Loop command.
4. Define the values to be retrieved from the data table, in any of the functions (e.g. Edit.Set) used inside the loop.

5. Select the Usage as data table.
6. Select the Value as the data table and column name.

> **Note:** A script can include multiple data tables. However, each loop can refer to only one data table.

Error policies such as *Continue* and *Break* change within a loop.

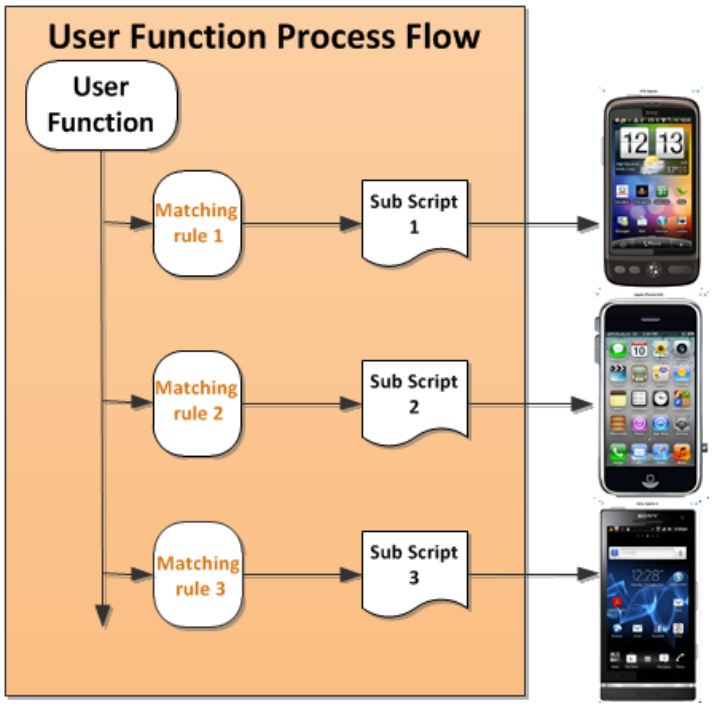
- *Continue* - Inside a loop jumps to the next iteration; outside it acts as "Abort".
- *Break* - Inside a loop exits to next command outside the loop; outside it acts as "Abort".

User Functions

User functions enable branching in a script by different device criteria. The branches are packaged into reusable functions defined by the user – i.e. user functions. User functions "wrap" scenarios where there are differences across devices, maintaining one script for multiple devices.

How it Works

The user function contains an implementation script for each criteria. The function is run and when the rule is matched, the appropriate implementation script is run on the specific device.



Create a User Function

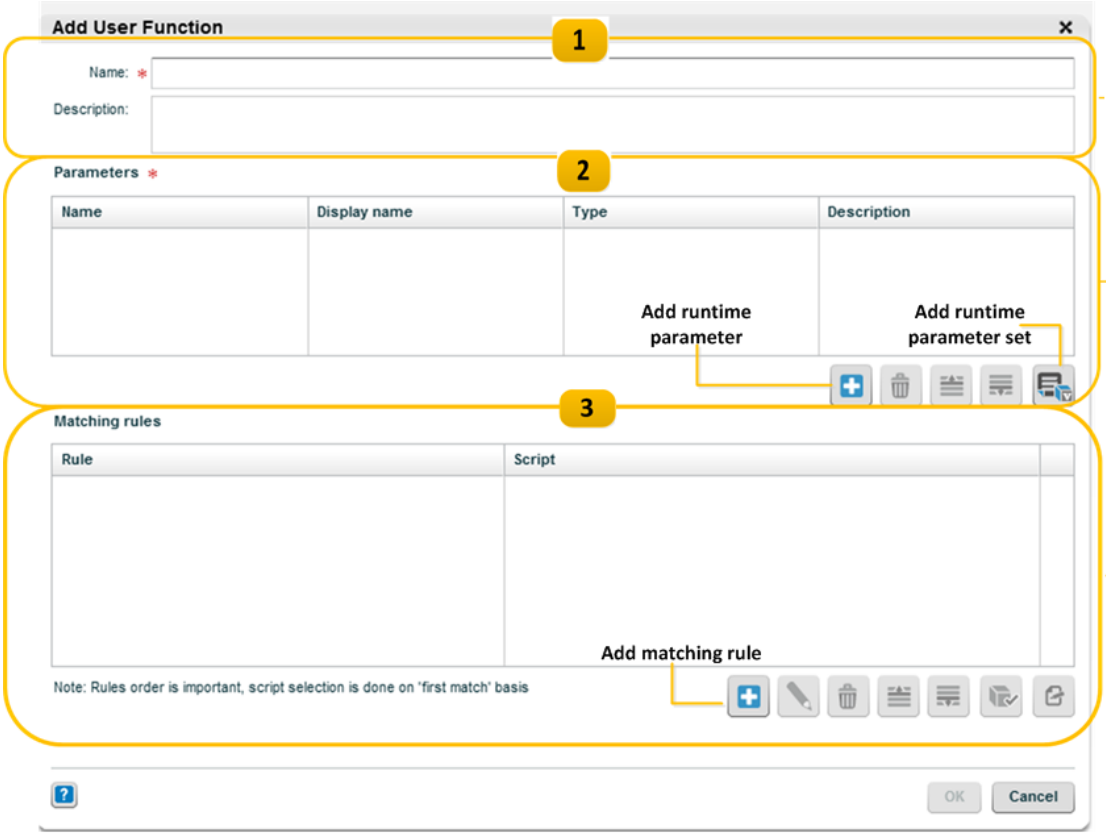
- Click on the ***User function*** tab in the *Commands Sidebar*.
- Select a folder location
- Click Add to create a new user function

It is recommended at this point to create the scripts you will be using for the user function.

Define the User Function

- Enter the Name and Description
- Parameters - Define each parameter of the branching scripts or import them automatically by selecting a script to be used as a baseline.

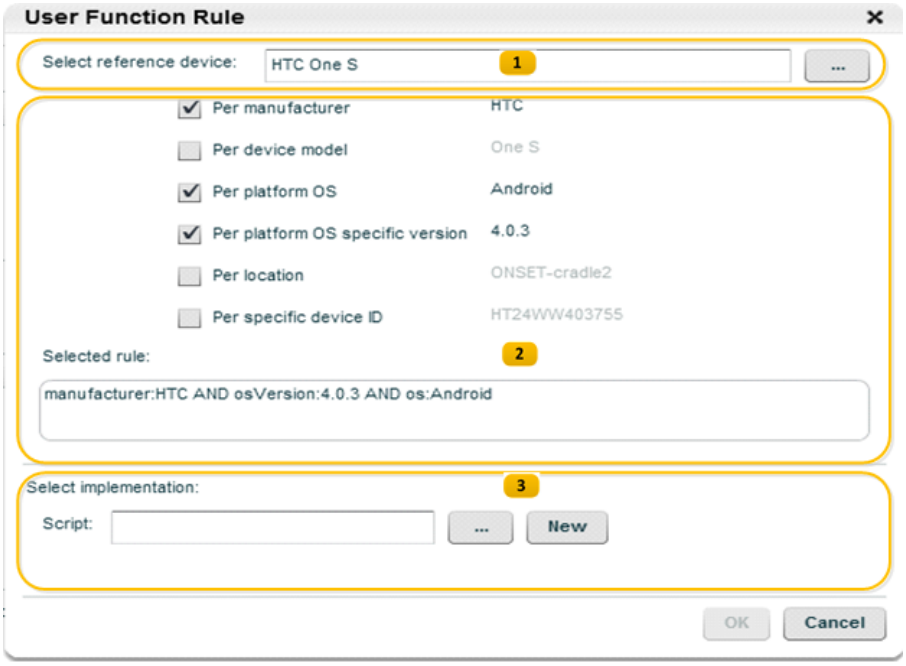
Important: The parameters defined in your user functions are the run-time variables defined in your implementation script. The run-time variables of your implementation script should be identical and in the same exact order.



Define the rules of the user function

- Select the reference device
- Select criteria - Selecting multiple criteria requires all criteria to be met. For example, manufacturer Samsung, Android platform OS 6.2.1 will be true only if the device is manufactured by Samsung AND on the Android platform AND running Android 6.2.1
- Select implementation script - Select an existing implementation script or click **New** to create a new implementation script.

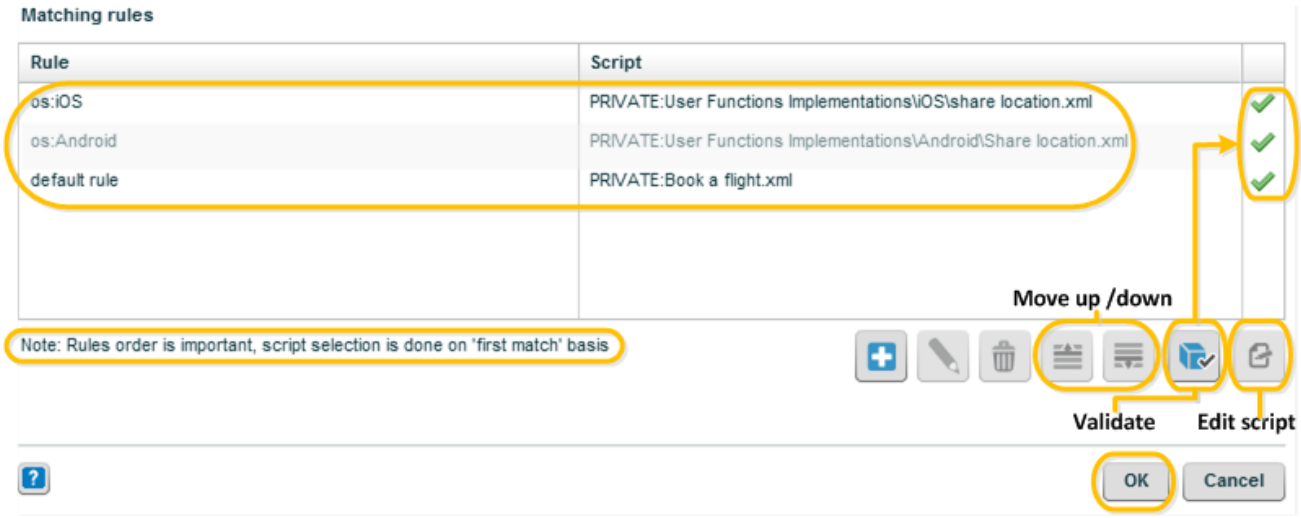
It is possible to add a default rule, that will run for all cases. To implement, open the User Function Rule and select an implementation script without selecting a device.



Validate the implementation script parameters

- Click on the *Validate* icon to confirm the parameters are defined correctly
- Click **OK** to save the User Function.

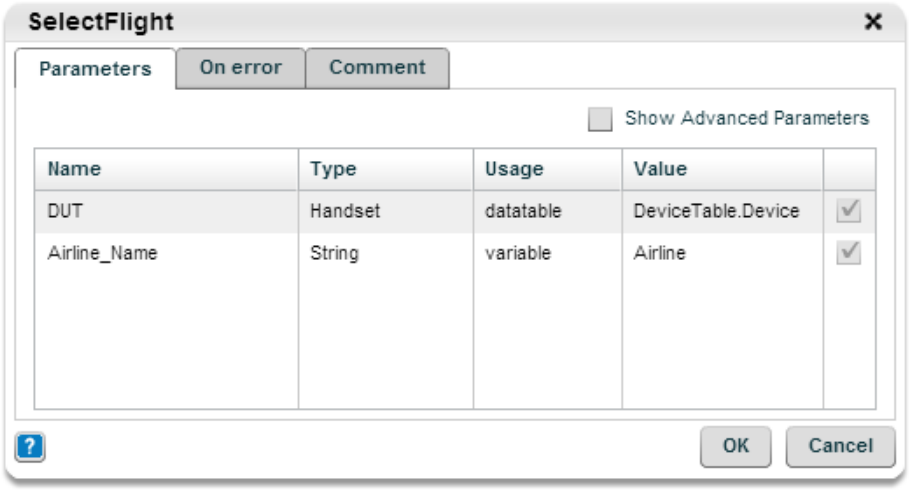
Note: You can access your implementation script in one click by using the *Edit Script* button.



Adding User Function to Script

- Insert your user function into the script by dragging it from the User Functions pane, just like regular commands.
- Open the user function in your script to define the parameter values**.**

Note: The parameters which appear in the user function are the run-time variables used in the implementation scripts.



Using the String Encoder in a user function:

- If the parameter is defined in the user function as *Secured String type*, when the user function is executed from within a script, the parameter type MUST be defined as Secured String.
- If the parameter is defined in the user function as *String type*, when the user function is executed from within a script, the parameter type can be defined as either String or Secured String.

Groups

Group existing script lines together by:

- Highlighting script lines and clicking on the **Group** icon in the Automation toolbar, or
- Directly insert script lines into an existing group by highlighting the location first.

Edit the group name to describe the group. A group can also be included within another group.

A group is treated as a single unit. Groups may be used within branches of *Condition* statements, or as the iterated commands in a *Loop*.

Repository

The Perfecto Lab *Repository* is used to store files such as scripts, data tables, test flows, and images in the Perfecto Lab. Files, such as applications, can be stored in the Repository and later used within automation scripts.

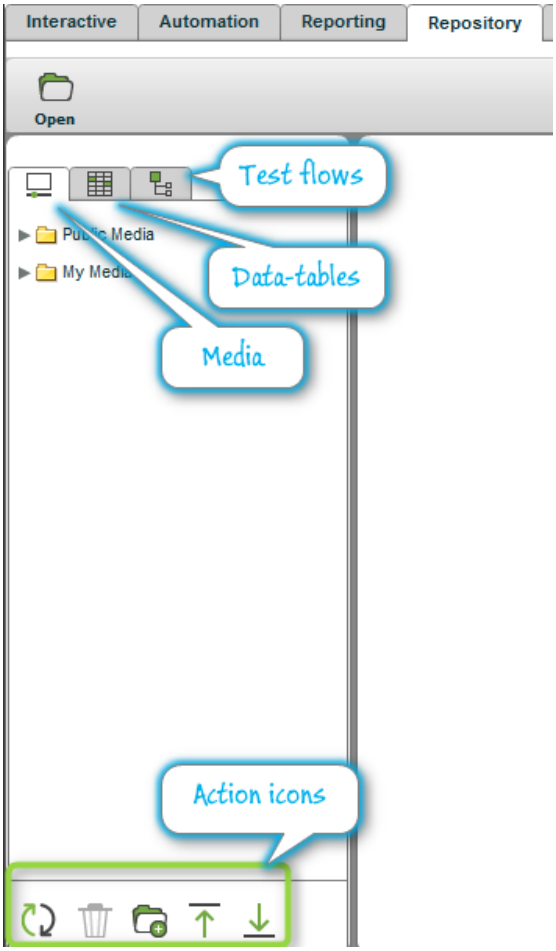
The Repository is divided into the following areas:

Repository Area	Description
Data Tables	CSV or XML files that contain data for scripts
Media	Images, videos, text files, and applications
Scripts	Perfecto scripts
Test Flows	Test flows

Each area is further divided into the following subareas:

Repository Area	Description
Public	Accessible by all users of the cloud
Group	Accessible by all users of a group. A single cloud can have multiple groups.
My	Private to the owner only

Files can be uploaded, downloaded and deleted, by using the action icons at the bottom of the sidebar:



Repository Keys

When using functions that utilize files from the repository, the paths to the files are specified as repository keys.

A repository key consists of the subarea and the path to the file. For example, to specify a file called myImage,jpg stored in the images folder located in the PRIVATE subarea, specify the key as *PRIVATE:images/myImage.jpg.*

A repository key does not need to contain the repository area in the path as it is inferred from the context. For example, in the *repository key parameter* of the *Install application* function, the repository area is automatically set to media.

Error Policy

Specify the command behavior if an error occurs during the command execution.

To do this:

- Double-click on the command line to open the command definition,
- Click on the ***On error*** tab,
- Select the desired error policy from the drop-down menu.

It is possible to apply the error policy to multiple commands. All selected commands will receive the same error policy.

Available error policies:

- **IGNORE** (default)
  Continue with the script flow even if the command fails. Marked as 'failed' in the report.
- **ABORT**
  Terminate the execution of the script and any other calling script.
- **BREAK**
  Inside a loop, stop the current loop and continue to the next command after the loop. Outside of a loop, acts as 'abort'.
- **CONTINUE**
  Inside a loop, stop the current iteration and continue to the next iteration. Outside of a loop, acts as 'abort'.
- **CATCH**
  Use the result as input to condition statement. Marked as 'Catch' in the report (not success or failure).

Keyboard Shortcuts

The following keyboard shortcuts are available while working with the Automation tab in the Perfecto Lab IDE:

Shortcut

Shortcut	Description
HOME	Top of Script
END	End of Script
CTRL S	Save Script
DEL	Delete selected Line
CTRL G	Insert Group
CTRL O	Open Script
CTRL N	New Script
CTRL L	Insert Loop
CTRL R	Record
CTRL P	Play (Execute Script)
CTRL V	Open Variables
CTRL F	Open Functions
CTRL D	Open Devices

Note: When using IE use CTRL+ALT, for the shortcuts that appear as "CTRL" in the table, to avoid conflicts with the browser's own shortcuts.