



# Automation Best Practices



February 25, 2015

## Table of Contents

Automation Best Practices .....	1
February 25, 2015 .....	1
1 General Scripting Guidelines.....	1
2 Templates and Best Practices - Test Cases, Quick Calls (Procedures) and Response Maps	
Examples .....	6
2.1 Test Case Template .....	6
2.2 Test Case Template - Main Procedure Details .....	7
2.3 Test Case Template - Cleanup Procedure Details.....	12
2.4 Test Case Template - Other Information .....	16
2.5 Debugging a Failed Test Case.....	17
2.6 Debugging using Breakpoints.....	19
2.7 Integrating Devices or Tools - Session Profiles, Quick Calls Libraries, Response Maps .....	20
2.8 Quick Calls Best Practices .....	26
2.9 Creating a Config Quick Call .....	27
2.10 Creating a Show Quick Call.....	31
2.11 Creating a Response Map .....	34
2.12 Test Scripts Reviewing Process .....	37

# 1 General Scripting Guidelines

## Script design

Guidelines for script planning and design:

- Structure of scripting: design flexible scripts, modular scripts, re-usable scripts
- A good scripting design leads to visibility, reusability, scalability and maintainability
- Design scripts that are independent of the test data or test environment
- Script design includes: modularization, parameterization and external definition of global functions and constants
- Define templates, standards and naming conventions for automation scripts; this will make it easier on the long-run to correlate test plans (from RQM) with test scripts, to follow the logic of test steps and to maintain test instructions
- Standard scripting (templates) will result in team consistency during test library design and prevent individuals from following their own coding standards, thus avoiding duplicate coding
- By developing libraries (iTest quick-calls), that can be reused again and again, time is saved for the entire organization/project team
- All scripts developed should have back-up
- All test-beds (setups) used for developing should be stable, available and well documented (all applications needed for development should be available).

## Naming Convention

Whenever a new test script is created and saved, it has to be given a proper name. Test scripts are correlated with test cases. It can be advantageous to use standard, consistent, and meaningful names for scripts and for the variables, constants, functions, and procedures used in those scripts. These naming conventions help make your scripts easier to read, understand, and maintain. It is also a common practice to use the same unique id for the developed scripts and their corresponding test cases.

***E.g. Project Name + Module Name + Test Script Function + Version Number(Id)***

*(same id or version number should be used also when naming corresponding test case in RQM)*

Item	Naming Convention Guidelines	Example
Script file name	As short as practicable, with no spaces or special haracters in the name. Capitalize the first letter of each word in the name.	Project_VerifyAttach_002.fftc
Variable	Meaningful name that reflects the purpose of	_intNumberOfFiles_

	the variable and the data it stores. The first word in the name should be a prefix indicating the type of data stored in the variable. The prefix should begin with a lowercase letter, and all subsequent words should begin with an uppercase letter.	
Constant	All uppercase letters. Separate individual words with underscores, except for constants that are intrinsic to a language or to an automation object.	_DOMAIN_NAME_
Function or Procedure	Verb followed by the noun that the verb acts upon. Capitalize all words in the name.	_DeleteBackupFiles_

### Template for writing Test Cases

Test development should take into consideration the following guidelines that were discussed and approved during planning phase:

- Each test case should verify if the selected **Topology** is complete (all parameters defined should be in place, otherwise the test will not continue)

The test case tries to extract all needed TBML properties and other Parameters:

- If an element is not found, the test will not run on this topology
- If all elements are found, the test will continue

- **Open all sessions to Resources** used by the test
- **Initial Setup** should be configured for all Resources

Need to determine:

- Non Sharable Devices: parameters/configs that we consider to be default
- Sharable Devices: what is common for all users

Need to perform:

- Non Sharable Devices: load to initial setup
- Sharable Devices: remove configuration added during the test execution phase

- **Test steps** should contain re-usable procedures (iTest QuickCalls), Analyze Rules and debug information; Applying test configuration and Verifying results will generate the Pass / Fail result of the test
- **Cleanup** procedure should be present for every test case

Need to remove all configuration added by the test in Step 4.)  
 Need to load configuration and return to the initial setup for Non Sharable Resources

- Last step of the test case implies **closing all sessions** used

	Action	Session	Description
	<b>procedure</b>		<b>main</b>
1	comment		Verify if the topology is correct - Extract topology properties and parameters
	analyze		
	none		
	assert	1	
	When True		
	Eval		set working_directory [tbml property -name PC working_directory]
	Eval		set dir_to_create [param dir_name]
	DeclareExecution		OK:Topology is correct - Starting test case
	When False		
2	comment		Open sessions
	analyze		message Information:Open CMD session
2.1	open	cmd_sp	project://cosmin_common_project/session_profiles/cmd.ffsp
3	comment		Bring devices to initial setup
	analyze		message Information:Setting CMD to minimum config
3.1	min_config	cmd_sp	-working_directory \$working_directory
4	comment		Test case execution
5	comment		Pass test and call cleanup
	analyze		assert 1
5.1	call		cleanup
	<b>procedure</b>		<b>cleanup</b>
1	comment		Extract topology properties and parameters
2	comment		Cleanup steps
3	comment		Close sessions

(sample for test case development guidelines)

### Template for writing Procedures/QuickCalls

All test cases should take into consideration the use of procedures (iTest Quick Calls). Below is presented our proposal (template) for writing quick calls that can be re-used by multiple tests.

**QuickCalls** are:

- a group of commands/actions performed together
- similar to C++ functions
- can receive and use any number of arguments
- can return a value or an output

- used for **automated testing**: can be called from any step of the test case that uses the session associated with the quickcall (hint: everything should be parametrized so that they can be re-used by future test cases)
- used for **manual testing**: after opening the session associated with the quickcall library, all the quickcalls are available for use (hint: ad-hoc manual testing can be performed using the existing quickcalls)

Guidelines for writing QuickCalls:

- **Config / Set QuickCallss** should offer means to validate the configuration:
  - Step 1. **Initialization** - used for:
    - Formatting attributes and parameters
    - Setting a validation variable to TRUE (**set retCode "TRUE"**)
  - Step 2. **Create command** using the formatted attributes (set cmd "set / config keyword \$arg1 keyword \$arg2 ...")
  - Step 3. **Issue command** created in Step 2.
  - Step 4. **Return retCode** (return retCode: \$retCode)

**Note:** The validation of the configuration will be performed by adding **Global Rules** or by **Modifying Global Events** on the **Reference Session Profile**:

- Global Rules are searching for Error patterns
- Global Events are monitoring special situations (like Web target not found)

	Action	Session	Description
	<b>procedure</b>		<b>config igmp</b>
1	eval		set retCode TRUE
2	command	\$session	configure
3	command	\$session	router igmp
4	if		\$version ne "null"
5	if		\$remove_config == "yes"
6	if		\$intf_list ne "null"
7	command	\$session	commit
8	command	\$session	end
9	return		retCode: \$retCode

Step Properties

General Steps **Global Events** **Global Rules** Parameters Reference Files Quality Center

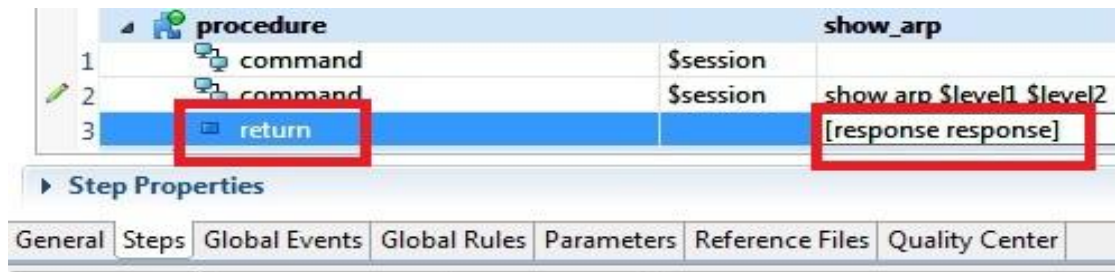
(sample for Config/Set QuickCalls)

- **Show QCs** should return the output needed and should be linked with the corresponding Response Maps (in **Response Map** > Applicability tab)

Step 1. **Initialization** - used for:

- Formatting attributes and parameters
- Step 2. **Create command** using the formatted attributes (set cmd “show keyword \$arg1 keyword \$arg2 ...”)
- Step 3. **Issue command** created in Step 2. Save the output of the command in a local variable using Step Properties > Other Prostprocessing > Store Response
- Step 4. **Return result** (return [response name\_of\_local\_variable])

**Note:** Response maps corresponding to this Quickcall will try to map the returned output. If all Response maps associated with this Quickcall are failing, the test will fail also.



(sample for Show QuickCalls)

This approach brings value to **debugging**:

- Inside the actual test case, after calling each config / set Quickcall we need to verify if the configuration was successful and Pass / Fail the test accordingly
- Inside the actual test case, after calling each show Quickcall we need to add Analyze Rules using the Queries defined in the Response Map

**Note:** Debugging is needed when a test case fails:

- Tester will need to inspect the generated test report (from RQM interface - HTML format)
- Tester will provide traces / logs that need to be started and collected for unstable test cases

This approach brings value to **maintenance**:

- If a CLI command / Web target / SNMP Object / etc. changes from one Firmware release to another, only the QCs / RMs will need maintenance. All the tests should remain valid if functionality is unchanged

**Note:** Tests suites are created and managed by RQM. RQM will exercise each test case, performing each of the steps from the template (check for topology, open sessions, init setup, test case steps, cleanup, close sessions).

## 2 Templates and Best Practices - Test Cases, Quick Calls (Procedures) and Response Maps Examples

### 2.1 Test Case Template

Test Case best practices imply using certain mechanisms by which each scenario can be validated no matter the current state of the devices involved. Also, for each scenario, every particular outcome or result must be treated in order to make sure that the final test result is accurate and predictable.

After running the scenario, the script must remove all the configuration added during the test case execution. This step is essential in order to make sure that following testcases will not be affected - also, by removing all added configuration debugging a failed testcase will be restricted only to the current scenario and does not need to take into consideration configuration or changes made by a previous script.

In order to cover all these issues, the following testcase template was implemented:

Each test will use two procedures:

- a. main procedure
  - i. Verify Topology and Extract Parameters
  - ii. Open Sessions needed for test execution
  - iii. Bring Devices to the Initial State
  - iv. Perform Test Case steps (Device configuration, Test Validation)
  - v. Pass Test Case and call Cleanup Procedure
- b. cleanup procedure
  - i. Verify if Execution should Stop, Pause or Continue when the Test Case is considered Failed
  - ii. Extract Topology properties and Parameters used to Revert the Configuration
  - iii. Execute Cleanup steps (Revert configuration added during the Test Case Execution)
  - iv. Close Sessions

The workflow consists of the following actions: For each testcase a series of Devices or Tools will be used in order to perform the required steps. Each of these entities has a list of attributes (IP



Address, Username, Password, eNodeB name, Physical Cell ID, etc.) and these attributes will differ from one Setup (Topology) to another.

In order to make sure that each testcase can run on different Setups (Topologies), all these attributes must be defined either as a Topology property, either as an Parameter (using test case parameter tab or global parameters file).

Before running the Test Case, the correct Topology file and Global Parameters file must be selected:

- In iTTest Enterprise: Right Click on the Topology / Parameter file and click Select as Global
- In iTTestRT: the correct Topology / Parameter file must be specified in the itestrt command

Action	Session	Description
<b>procedure</b>		<b>main</b>
1  comment		Verify if the topology is correct - Extract topology properties and parameters
analyze		assert 1
2  comment		Open Sessions
analyze		message Information:Open sessions
3  comment		Bring Device to the initial state
analyze		message Information:Bring Device to the initial state
4  comment		Perform testcase steps - Use only FailTest action and not PassTest
analyze		message Information:Perform testcase steps
5  comment		Pass test and call cleanup
analyze		assert 1
5.1  call		cleanup
<b>procedure</b>		<b>cleanup</b>
1  comment		Abort, Pause or Continue execution
analyze		assert 1
analyze		assert ([info status] == "Fail") && (\$option == "abort")
analyze		assert ([info status] == "Fail") && (\$option == "pause")
2  comment		Extract topology properties and parameters
analyze		assert [info exists global cleanup_ok]
analyze		message Information:Starting the cleanup procedure
analyze		assert 1
3  comment		Cleanup steps
analyze		message Information:Removing added configuration
4  comment		Close sessions
analyze		message Information:Closing opened sessions

## 2.2 Test Case Template - Main Procedure Details

1. In Step 1, all the Topology Properties and Parameters that are needed inside the test case will be extracted and saved into local variables. Inside the testcase, all procedures or validation steps will be using variables and not hardcoded values. This is very important for the script maintainance and code reusing.

1	comment	Verify if the topology is correct - Extract topology properties and parameters
	analyze	
	none	
	assert	1
	When True	
	Eval	set wifi_nic_name [tbml property -name "pc" wireless_nic]
	Eval	set usb_nic_name [tbml property -name "pc" usb_nic]
	Eval	set mifi_ssid [tbml property -name "mifi" mifi_ssid]
	Eval	set mifi_ip_address [tbml property -name "mifi" ip_address]
	Eval	set mifi_username [tbml property -name "mifi" username]
	Eval	set mifi_password [tbml property -name "mifi" password]
	Eval	set feature_battery [tbml property -name "mifi" "features battery"]
	DeclareExecution	OK:Extracted test case parameters
	When False	

If one of the Topology Properties or Parameters extracted in this step is missing, the test execution cannot continue because not all the needed data is available (this can happen if the incorrect Topology or Global Parameters file is selected as Global). An error message will be displayed in the Test Execution logs and the execution will stop.

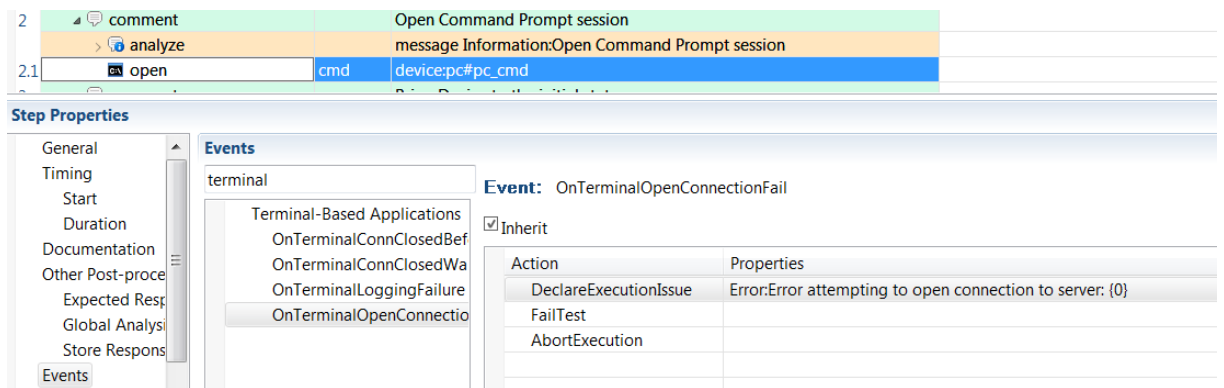
When such a Topology property or Parameter is missing, iTest will trigger an Event called OnInterpreterError. For this Step, the actions executed when the event is triggered were overwritten using the following configuration:

Step Properties		
General	Events	
Timing	oninter	
Start	Execution	
Duration	OnInternalError	
Documentation	OnInterpreterError	
Other Post-processor		
Expected Response		
Global Analysis R		
Store Response		
Events	Event: OnInterpreterError	
Process Filter	<input type="checkbox"/> Inherit	
	Action	Properties
	DeclareExecutionIssue	Error:Could not extract all needed parameters or topology properties. Topology is not correct
	ExitProcedure	

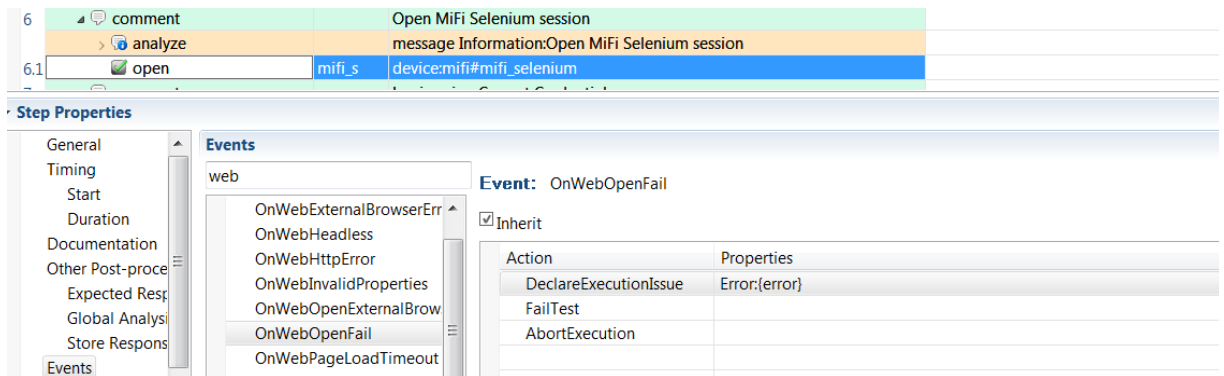
This is the mechanism implemented to ensure that all the data needed by the test case is available. This is the initial validation by which we make sure that the scenario has all the prerequisites to be completely performed.

2. In Step 2, all the Sessions needed for executing the scenario will be opened. This is the second validation step performed before actually starting executing the steps of the Test Case. If opening one of the sessions fails iTest will trigger an Event specific to the type of the session used and based on the actions defined for that event, the script execution will not continue. This step covers L3 connectivity validation or tool configuration (valid TCL interpreter, External Browser availability, etc.)

Example for Command Prompt session (the same Event applies for Telnet and SSH):



Example for Selenium session:



- After Extracting all parameters needed for the test case execution and Opening sessions to all needed entities, the devices and tools can be used and configuration can be changed. In order to make sure that each test scenario will run in a predictable manner, all the devices / tools should be put into an initial state. This state needs to be defined for all the entities used by the Automation Framework and the mechanism by which this initial configuration is applied must be identified (example - load configuration to a predefined configuration file; make sure that the required interfaces are available and up; clear logs in tools used for log capturing; etc.)

Example for Service Testing - Before opening MiFi selenium Session, make sure that the correct Network Interface is enabled on the Client (PC):

3	comment	Bring Device to the initial state
3.1	manage_nic	cmd -nic_name \$usb_nic_name -action disable
	analyze	contains retCode: FALSE, assert \$value == 1
3.2	manage_nic	cmd -nic_name \$wifi_nic_name -action enable
	analyze	contains retCode: FALSE, assert \$value == 1

4. After bringing devices and tools to the initial state, test case execution can begin. For Step 3 and Step 4, all configurations and show actions will be validated in order to identify the result. This way the root cause of a failure can be identified.

If one action fails, the test case will be Failed. When changing the test result from Indeterminate (this is the initial iTest state when starting test case execution) to Failed, iTest will trigger an event called OnFailTestAction. The actions performed by this event were overwritten to execute all cleanup steps and exit the execution. This is the mechanism used to make sure that at the first failure the test case execution will be stopped and the cleanup steps will be processed in order to remove the configuration added by the test case - The test execution will stop when the FailTest rule is applied!

Validation for the configuration Quick calls is performed based on the return code received from the procedure themselves. Each configuration Quick call is required to keep track of the success of the actions performed and return the result to the calling test case.

Example for Service Testing - Changing SSID for MiFi device. The code returned by the change\_network\_wifi\_settings quick call is verified and based on the action success the test will continue or will be Failed.

7	comment	Change SSID name
7.1	change_network_wifi_sett mifi_s	-ssid \$mifi_ssid_v2
	analyze	
	contains	retCode: FALSE
	assert	\$value == 1
	When True	
	DeclareExecut	Error:Could not change SSID from \$mifi_ssid to the \$mifi_ssid_v2
	FailTest	
	When False	
	DeclareExecut	OK:Changed SSID from \$mifi_ssid to \$mifi_ssid_v2

Validation for the show Quick Calls is performed based on the values present in the output returned by the show command issued or the table / field extracted and returned. Each show Quick call is required to collect the information needed and return it to the calling test case. Response maps can be associated with each quick call and the output will be parsed according to the response map implementation - patterns and queries can be defined. One query will return one value or a list of values based on which the validation can be performed.

If the value / values returned by the query defined in the Response Map associated with the current Quick call are correct, the test execution will continue. If not, the test case can be Failed.

Example for Service Testing - LAN information is extracted from the device. A query named ssid (defined in the response map associated with the show\_lan\_information quick call) will extract the current SSID value. This result (available in \$value) is verified against a parameter defined in Step 1. If the assertion is True, the test execution will continue. If the assertion is False, the test will be Failed - cleanup steps will execute and the test will stop.

11.3	show_lan_information	mifi_s
	analyze	
	query	ssid()
	assert	\$value eq \$mifi_ssid_v2
	When True	
	DeclareExecution	OK:LAN Information output shows the new SSID used: \$mifi_ssid_v2
	When False	
	DeclareExecution	Error:LAN Information output does not show the new SSID. Expected: \$mifi_ssid_v2
	FailTest	

5. After executing all Test case steps and validating configuration and obtained outputs / results, we can end up in one of the two cases:
  - a. Test case Failed -> Cleanup procedure will be executed; Execution will Stop
  - b. Test case steps passed (no FailTest rule was applied)

For the second scenario we need to ensure that the PassTest result is applied and also the configuration added during the previous steps should be now removed in order to avoid affecting the following test cases that will be executed on the same Topology (Devices). These actions are performed in Step 5 (from the Test case Template).

5	comment	Pass test and call cleanup
	analyze	
	none	
	assert	1
	When True	
	PassTest	
	When False	
5.1	call	cleanup

## 2.3 Test Case Template - Cleanup Procedure Details

1. Step 1 should not be modified in any of the Test cases. This is a mechanism implemented in order to permit Pausing, Stopping or Continuing test execution once the FailTest rule was applied (Break on Error mechanism).

The default action when the Test is declared as Failed is continue - Cleanup steps will be executed and the Test execution will end after closing all Sessions






In order to change the default behavior one parameter needs to be added to the Global Parameters file (or to the Test case in the Parameters tab):

test\_params > failtest\_option with the value:

- continue (default action - this is the value that should be used in most of the cases)
- pause (action used for debugging purpose - execution will be paused once the test is failed and all sessions will remain opened and will be available for manual debugging)
- abort (action used for skipping cleanup steps - execution will stop right after the test is failed and no cleanup steps will be executed)

### Parameters

☐ Show the parameter values that will be used for execution.

	Name	Value	Description
	test_params		
	failtest_option	continue	Available values: continue, pause, abort
	wan_info		
	lab_ipv4_subnet	255.255.255.0	
	lab_ipv4_default	172.17.215.21	
	lab_ipv4_dns_server	172.16.56.132	
	lab_ipv4_dns_server	202.138.96.2	

The logic behind this mechanism is as follows: The failtest\_option parameter is extracted from the Global Parameters file or from the Test case Parameters tab:

- If the parameter is available, the value will be saved in the local variable named option
- If the parameter is not available, an Warning message will be issued informing the user that, since no failtest option was specified, the default option will be used - test execution will continue by performing the cleanup steps. This message can be modified from the event called OnInterpreterError

Once the parameter is extracted, the value will be compared against “abort” and “pause”. If the test result is Failed, the corresponding action will be applied.

procedure		cleanup
1	comment	Abort, Pause or Continue execution
	analyze	
	none	
	assert	1
	When True	
	Eval	set option [param test_params/failtest_option]
	When False	
	analyze	
	none	
	assert	(([info status] == "Fail") && (\$option == "abort"))
	When True	
	DeclareExecution	OK:[info procedure] - Aborting execution
	AbortExecution	
	When False	
	analyze	
	none	
	assert	(([info status] == "Fail") && (\$option == "pause"))
	When True	
	DeclareExecution	OK:[info procedure] - Pausing execution
	PauseExecution	
	When False	

- Step 2 is used for extracting the parameters and topology properties needed when reverting the configuration. Users should only modify the last analyze rule by adding the topology properties or parameters that they want to extract and use in the cleanup procedure.

One other mechanism added to this step is the logic used to detect cleanup loops. The FailtestAction event (triggered every time when the test is considered Failed) is calling the Cleanup procedure and Exits Execution after finishing all cleanup steps. However, inside the cleanup steps some outstanding events can lead to triggering the FailtestAction event (session is stuck or other timeouts) which will create a “cleanup loop”.

In order to avoid executing the Cleanup procedure multiple times, the following steps are used:

- If the cleanup\_ok variable does not exist it will be created before executing the cleanup steps
- If the cleanup\_ok variable exists this means that the cleanup procedure was already executed and the execution will be stopped

2	comment	Extract topology properties and parameters
	analyze	
	none	
	assert	[info exists global cleanup_ok]
	When True	
	DeclareExecution	Error:This is a cleanup call loop - it needs to be fixed; exit cleanup and hope for t
	ExitProcedure	
	SkipRemainingRu	
	When False	
	Eval	gset cleanup_ok "finished"
	analyze	
	none	
	message	Information:Starting the cleanup procedure
	analyze	
	none	
	assert	1
	When True	
	Eval	set wifi_nic_name [tbml property -name "pc" wireless_nic]
	Eval	set usb_nic_name [tbml property -name "pc" usb_nic]
	When False	

- In this step the user should add all the Quick calls or procedures used to revert the configuration added during the test case execution.

One important note for all these steps is that no FailTest rule should be used (in order to avoid creating a cleanup loop). The user should only display the result for each configuration change - based on the return code obtained from the Quick call or procedure used.

Example for Service Testing: Changing SSID name back to the initial one after executing the test case



3	comment		Cleanup steps
	analyze		message Information:Removing added configuration
3.1	comment		Enable WiFi connection and disable USB connection
3.2	comment		Wait for device to become available
3.2.1	do_ping	cmd	-ip_address \$mifi_ip_address
3.3	comment		Change SSID name
3.3.1	reload	mifi_s	
3.3.2	login_device	mifi_s	-username \$mifi_username -password \$mifi_password
	analyze		
	contains		retCode: TRUE
	assert		\$value == 1
	When True		
	DeclareExe		OK:Login worked when using correct credentials
	When False		
	DeclareExe		Error:Login failed when using correct credentials
3.3.3	change_network_wifi_s	mifi_s	-ssid \$mifi_ssid
	analyze		
	contains		retCode: FALSE
	assert		\$value == 1
	When True		
	DeclareExe		Error:Could not change SSID from \$mifi_ssid_v2 to the \$mifi_ssid
	When False		
	DeclareExe		OK:Changed SSID from \$mifi_ssid_v2 to \$mifi_ssid

4. The last Step from the Cleanup procedure is used to Close all the sessions used for the Test case execution:

4	comment		Close sessions
	analyze		message Information:Closing CMD session
4.1	close	mifi_s	
4.2	close	cmd	

## 2.4 Test Case Template - Other Information

Note: In order to call the Cleanup procedure when the test case is declared as Failed, one Global Event must be modified for each test case. When reviewing test cases created by other users make sure that the Global Event > OnFailTestAction is configured as in the image below:

### Test Case Global Events

type filter text

Event: OnFailTestAction

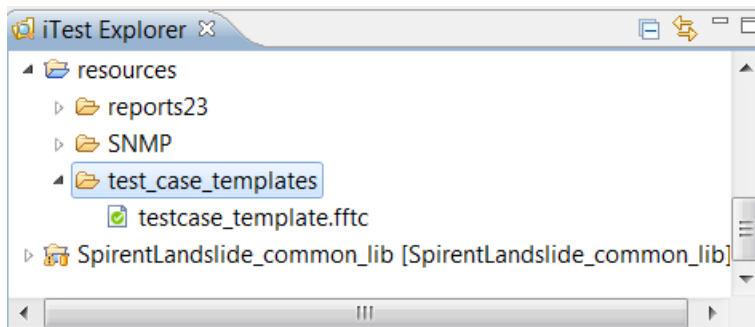
☐ Inherit

Action	Properties
CallProcedure	cleanup
ExitProcedure	

General Steps Global Events Global Rules Parameters Reference Files Quality Center

Note: In order to automatically use the Test Case Template when creating a new Test Case, please follow this steps:

- Open iTTest Enterprise
- Make sure that iTTest Explorer View is opened (Window > Show View > iTTest Explorer)
- Create a new folder called test\_case\_templates under the resources project
- Copy the testcase\_template.fttc file (provided by the Spirent team) into the test\_case\_templates folder.



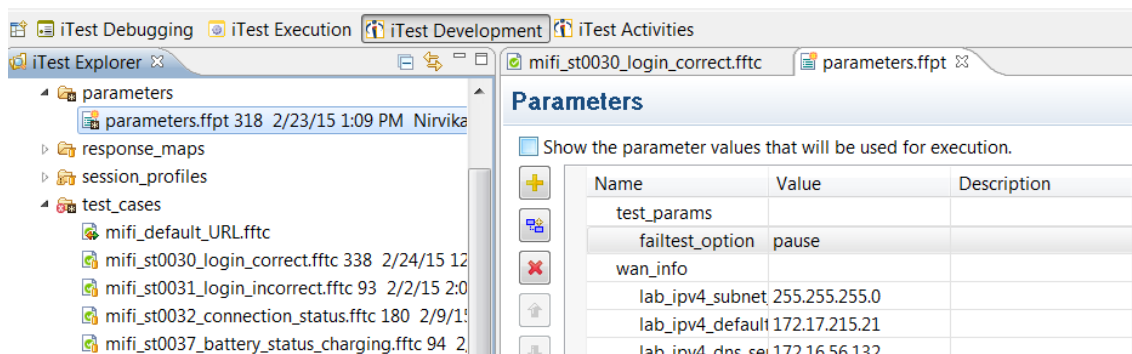
Automation Workflow best practices:

- All Test cases created by the Spirent team or Reliance Jio engineers should respect the presented template. This is needed for automation consistency and to ensure that the training provided to new hire is relevant and reflects the automation assets already created.
- Every improvement added to the test case template should be implemented into all existing test cases. Also, the documentation needs to be updated with the new information.

- All Test cases created by each engineer should be reviewed by the Spirent team or an experienced Reliance Jio engineer. Feedback should be provided and required changes should be implemented as part of the automation process. After implementing the changes the test case can be considered automated and is ready for integration in the regression suite.

## 2.5 Debugging a Failed Test Case

Debugging a Failed Test case is an important task in maintaining the automation assets and also in identifying device or network issues. In order to ease this task we added the Break on Error mechanism (Step 1 from the Cleanup procedure) which uses the **test\_params/failtest\_option** parameter to determine if the execution needs to continue, pause or abort when the test fails. In order to make use of this mechanism the user needs to add this parameter to the Global Parameters file and set the correct value.

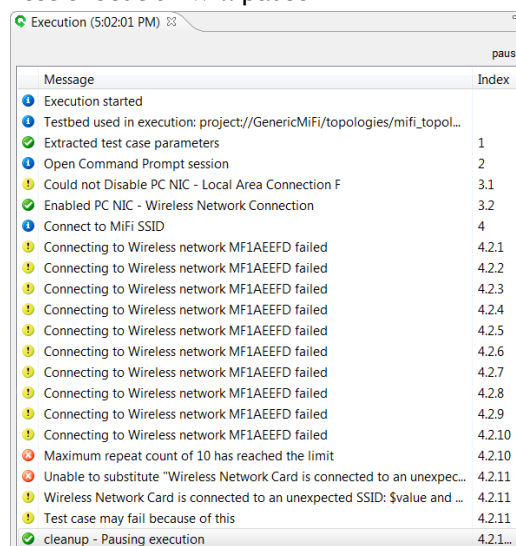


The screenshot shows the iTest Development window with the 'Parameters' tab selected. The 'Parameters' table lists various parameters used in the test case.

Name	Value	Description
test_params		
failtest_option	pause	
wan_info		
lab_ip4_subnet	255.255.255.0	
lab_ip4_default	172.17.215.21	
lab_ip4_dns_server	172.16.56.132	

If the **failtest\_option** parameter is set to **pause** and the test case failed, the following actions will be performed:

- Test execution will pause



The screenshot shows the iTest Execution window with the 'Execution (5:02:01 PM)' tab selected. The execution log shows the following messages:

Message	Index
Execution started	1
Testbed used in execution: project://GenericMifi/topologies/mifi_topol...	2
Extracted test case parameters	3.1
Open Command Prompt session	3.2
Could not Disable PC NIC - Local Area Connection F	4
Enabled PC NIC - Wireless Network Connection	4.2.1
Connect to Mifi SSID	4.2.2
Connecting to Wireless network MF1AEFFD failed	4.2.3
Connecting to Wireless network MF1AEFFD failed	4.2.4
Connecting to Wireless network MF1AEFFD failed	4.2.5
Connecting to Wireless network MF1AEFFD failed	4.2.6
Connecting to Wireless network MF1AEFFD failed	4.2.7
Connecting to Wireless network MF1AEFFD failed	4.2.8
Connecting to Wireless network MF1AEFFD failed	4.2.9
Connecting to Wireless network MF1AEFFD failed	4.2.10
Maximum repeat count of 10 has reached the limit	4.2.11
Unable to substitute "Wireless Network Card is connected to an unexpect...	4.2.11
Wireless Network Card is connected to an unexpected SSID: \$value and ...	4.2.11
Test case may fail because of this	4.2.11
cleanup - Pausing execution	4.2.1...

- All sessions opened during the test case execution will remain opened
- All logs and commands issued during the test case execution will remain
- Manual execution or manual steps can be performed to debug the issue in the failed state

```

mifi_st0030_login_correct.fttc  parameters.fttc  pc1
C:\Program Files\Spirent Communications\iTest 4.4>netsh wlan connect ssid="MFLAEEFD"
The network specified by profile "MFLAEEFD" is not available to connect.

C:\Program Files\Spirent Communications\iTest 4.4>netsh wlan connect ssid="MFLAEEFD"
The network specified by profile "MFLAEEFD" is not available to connect.

C:\Program Files\Spirent Communications\iTest 4.4>netsh wlan connect ssid="MFLAEEFD"
The network specified by profile "MFLAEEFD" is not available to connect.

C:\Program Files\Spirent Communications\iTest 4.4>netsh wlan show interface

There is 1 interface on the system:

    Name                : Wireless Network Connection
    Description          : Intel(R) Centrino(R) Ultimate-N 6300 AGN
    GUID                 : c62ed5f7-227b-4a9c-b03f-303f7580844b
    Physical address     : 00:24:d7:b1:ef:50
    State                : connected
    SSID                 : AR5510_64B777
    BSSID                : 58:1d:91:64:b7:77
    Network type         : Infrastructure
    Radio type           : 802.11n
    Authentication       : WPA2-Personal
    Cipher               : CCMP
    Connection mode       : Profile
    Channel              : 9
    Receive rate (Mbps)  : 72
    Transmit rate (Mbps) : 72
    Signal               : 99%
    Profile              : AR5510_64B777

    Hosted network status : Not available

C:\Program Files\Spirent Communications\iTest 4.4>

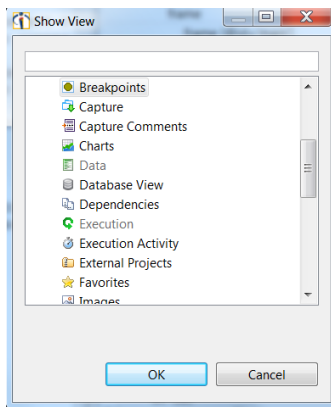
```

- The user can change the perspective to Debugging and inspect all the local or global variables used inside the test

The screenshot displays the iTest Debugging interface. On the left, a command window shows the same netsh commands and output as the previous image. On the right, the 'Data View' pane is open, showing a hierarchical tree of variables and their values. The tree structure is as follows:

- stack [@id='1']
  - frame
    - frame [@id='main']
      - index: 0
      - value: AR5510\_64B777
      - wifi\_nic\_name: Wireless Network C...
      - usb\_nic\_name: Local Area Connect...
      - mifi\_ssid: MFLAEEFD
      - mifi\_ip\_address: 192.168.15.1
      - mifi\_username: administrator
      - mifi\_password: administrator
      - wifi\_connected: 0
      - i: 1
      - frame [@id='cleanup']
        - index: 0
        - value:
        - option: pause
- parameters
  - profiles
  - test\_params
  - wan\_info

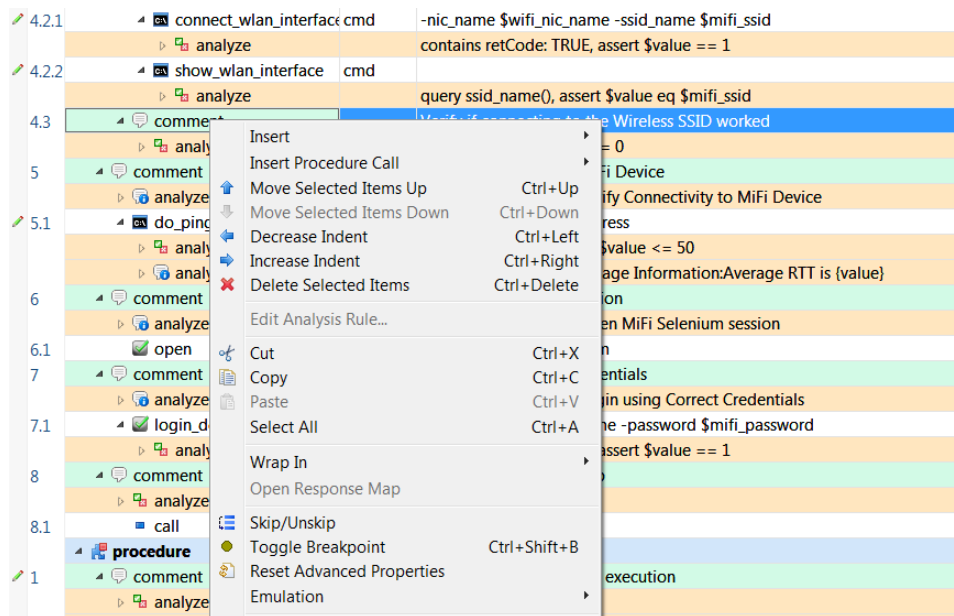
Note: If the Data View is not opened, the User should open it from Window - Show View - Other - Data.



**Important:** The `failtest_option` parameter should be changed back to `continue` after Test Development and Debugging is Completed.

## 2.6 Debugging using Breakpoints

Breakpoints are used to pause the execution at a certain step. The breakpoints should be added before starting the execution and iTest execution will pause as soon as the step with the breakpoint is reached, without executing the step. In order to add a new breakpoint the User needs to Right Click on the step and select Toggle Breakpoint:

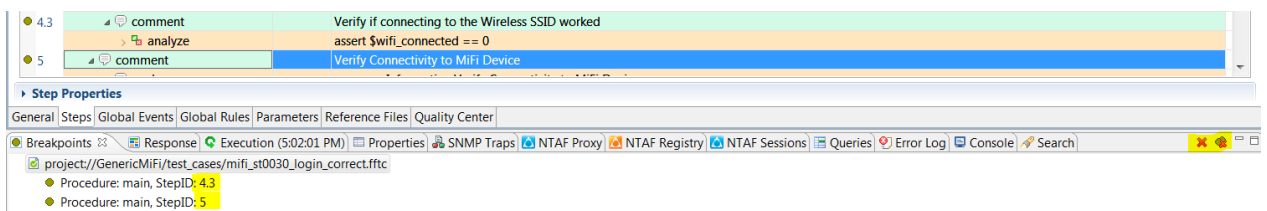


After adding a Breakpoint, a yellow circle will appear next to the Step on which the Breakpoint was added on:

4.2.2	show_wlan_interface	cmd	
	analyze		query ssid_name(), assert \$value eq \$mifi_ssid
4.3	comment		Verify if connecting to the Wireless SSID worked
	analyze		assert \$wifi_connected == 0
5	comment		Verify Connectivity to MiFi Device
	analyze		message Information:Verify Connectivity to MiFi Device
5.1	do_ping	cmd	-ip_address \$mifi_ip_address
	analyze		query loss_ratio(), assert \$value <= 50
	analyze		query average_rtt(), message Information:Average RTT is {value}

The execution will pause at the selected step and the User can change the Perspective to Debugging in order to inspect the local and global variables and their values. Also, the user can change the values for any of the local and global variables and can resume the execution. The new data added for each of the variables will now be used inside the test execution.

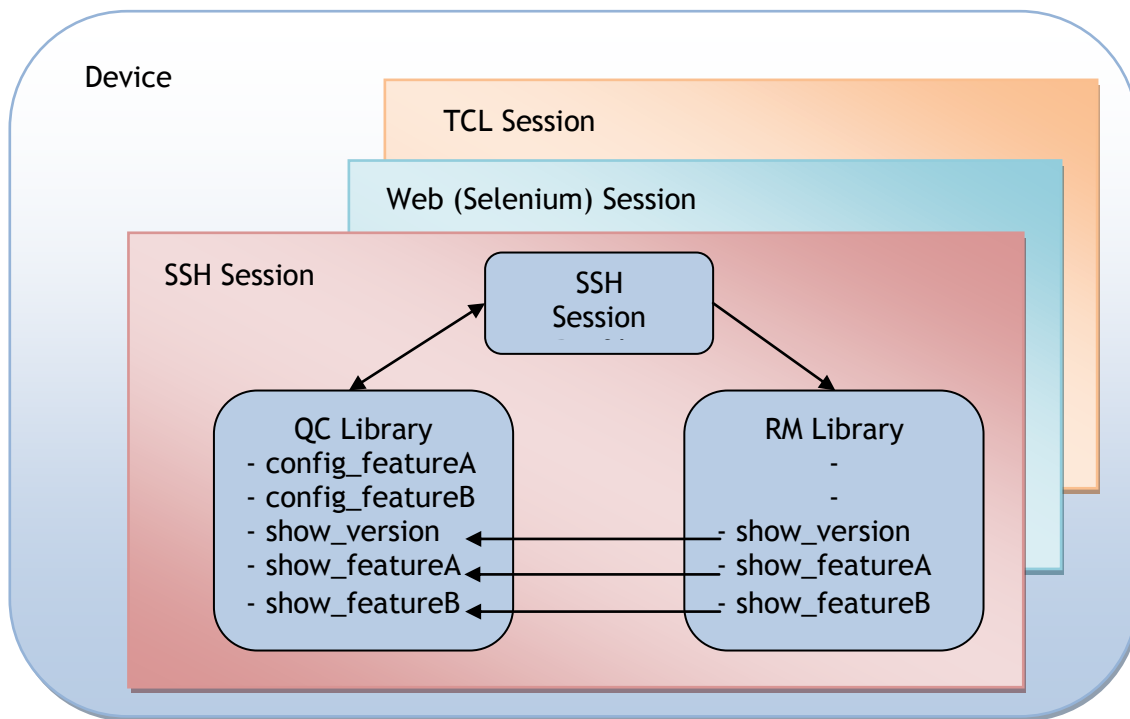
In order to remove a Breakpoint the user needs to Right Click on the same step and use Toggle Breakpoint again. One other way to clear Breakpoints is to open Window - Show View - Other - Breakpoints. All added Breakpoints will be visible inside this Tab and can be cleared using the buttons from the Top Right corner of the Tab.



**Important:** All breakpoints should be removed after Test Case Development and Debugging is completed

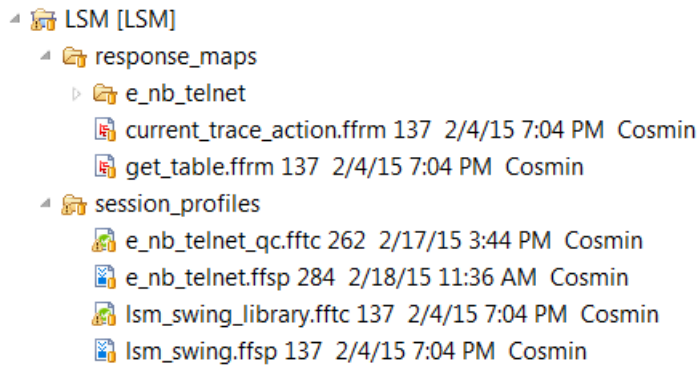
## 2.7 Integrating Devices or Tools - Session Profiles, Quick Calls Libraries, Response Maps

In order to integrate a new Device or Tool with the iTest Framework new Session Profiles must be created. If the Device supports multiple types of sessions (SSH, Web Gui, TCL, etc.), one Session Profile for each type of Session should be created. For each of the Session Profiles defined, the user must create a Quick call Library containing procedures associated with this type of Session and Response Maps (one Response Map is associated with one procedure from the Quick call Library).



Creating the assets for each Device or Tool should be done using the following Procedure:

- Create a new iTest Project using (the name of the Project should contain the name of the Device or Tool; the iTest Project should contain the following structure of subfolders: session\_profiles, response\_maps)
- In the session\_profiles subfolder:
  - Create one Session Profile by specifying the correct Session type and modify the Session parameters (by opening the More menu under the Start tab) to reflect the Device or Tool Configuration (example: For Telnet / SSH session profiles the Size of the Terminal and the list of known Prompts should be configured correctly).
  - Naming convention for the session profile is <name\_of\_device>\_<session\_type>.ffsp
  - Check "This is a reference session profile" checkbox from the Misc tab.
- In the session\_profiles subfolder:
  - Create one test case - this test case will be associated with the Session Profile created in the previous step and will become the Quick call Library for this Session Profile.
  - Naming convention: <name\_of\_session\_profile>\_qc.fftc



- Tie the Session Profile to the Test case (Quick call Library) using the following steps:
  - o Open Test case General Tab: Check “Include this test case when listing QuickCall Libraries” checkbox from the QuickCall sub-tab.  
Browse to the Session Profile or Device: select the corresponding Session Profile

At this point, connection from the Quick Call to Session Profile is complete

- o Open Session Profile Misc Tab: In the QuickCall sub-tab Browse to the corresponding Test case (the one used in the previous step)

At this point, connection from the Session Profile to Quick Call is complete.  
Since both files are configured to point to the each other, no Error should appear.  
All procedures created in the Quick Call Library will be associated with the Sessions opened using this Session Profile and will be available for calling inside the test cases.

- Tie the Session Profile to the Response Map Library
  - o Open Session Profile Misc Tab: In the Response Map Library sub-tab add the root path for this Project: project://<project\_name>/



e\_nb\_telnet.ffsp e\_nb\_telnet\_qc.fttc

## General

### General Information

Headline:

Owner:

Description:

☐ Include this test case when listing procedure libraries for call steps and CallProcedure actions

### Execution Behavior

Entry Point:

☐ Generate an execution message for each comment step that is executed

☐ Force all variables to be global

☒ Discard session profile parameters when sessions close

Execution time limit:

Default step timeout:

Estimated execution time:

☒ Update "Estimated execution time" value after execution

### Procedures

[Add to or edit the steps and procedures in this test case](#)

### Advanced

☒ [Parameters](#): Define settings that can be accessed by the test case

☒ [Global Events](#): Define the default behavior when various unusual events arise during execution

☒ [Global Rules](#): Define analysis rules that will apply to all steps

### Test Suite Reporting

☒ Include test cases executed by this test case (via EXEC.run)

Include this test case in response to summarize steps:

Include execution issues in response to summarize steps:

### Emulation

☐ Enable emulation for the test case

☐ Enable emulation duration for the test case

### QuickCall

☒ Include this test case when listing QuickCall Libraries

Session profile or device:  [Browse...](#)

General Steps Global Events Global Rules Parameters Reference Files Quality Center

**Miscellaneous**

**General Information**

Headline:

Description:

☐ This is a reference session profile

**Response Map Library**

Select the response map library to use to map responses from the session during capture or execution

**Note:** You can create a response map library using the [Response Map Library wizard](#)

Response map library:

**Form Map Library**

Select the form map library to use when selecting targets during a session

**Note:** You can create a form map library using the [Form Map Library wizard](#)

Form map library:

**Emulation Source**

Select the response map library that should provide emulated responses for the session

**Note:** You can create a response map library using the [Response Map Library wizard](#)

Emulation source:

**QuickCall**

[View QuickCall library](#)

Initialization QuickCall:

Start Global Events Global Rules Parameters Response Filters Misc

- Associate show Quick calls to Response Maps:  
For Quick calls that are returning output from the Device or Tool, predefined queries can be created in order to parse the output and extract data - this data can be used for validation inside the Test Case. iTest can take decisions based on the action that is performed at a certain time and can be configured to attempt to parse the output once it is received.

All the logic used to parse the output and extract (populate) the queries is comprised in a Response Map. Each Response Map covers all output variations for one command / show Quick call and should apply only when that command / show Quick call is used.

The general workflow for this mechanism is the following:

- A show Quick call collects output directly from the Device and returns this output to the test case
- A Response Map configured to apply to this Quick call will attempt to parse the output received from the Quick call in order to extract the defined queries In order

to configure one Response Map to apply when a certain Quick call is used please follow these steps:

- Create a new Response Map in the response\_maps subfolder
- Naming convention: The name of the Response Map should be identical with the name of the Quick call on which it will apply
- Add Response Map Sample and Parsing logic (details in the Response Map Best Practices section)
- In Applicability Tab:
  - Select the type of session for which this Response Map should apply (example: Telnet or SSH or Selenium...)
  - In the Action field enter the Quick call name for which this Response Map should apply
  - Change Priority to the required value (this is optional and should be used only if multiple Response Maps are used for the same Quick call / Action)

**Automatic Mapping Applicability**

**Response Map Applicability**

Specify when to use this response map. You can use wildcards and regular expressions to enhance your Action, Target, and Command settings.

☒ If this response map is in a library, then apply the map to a response when all of the following conditions are met.

Select from the list only if you want to restrict this response map to the selected applications.

Session types: ☐ Tcl Shell ☒ Telnet ☐ UDP ☐ VMware vSphere Client

Action: change\_algorithm\_eia\_eea

Target:

Command:

Specify how to interpret the Action, Target, and Command settings.

Compare properties using: Wildcard

**Map Priority**

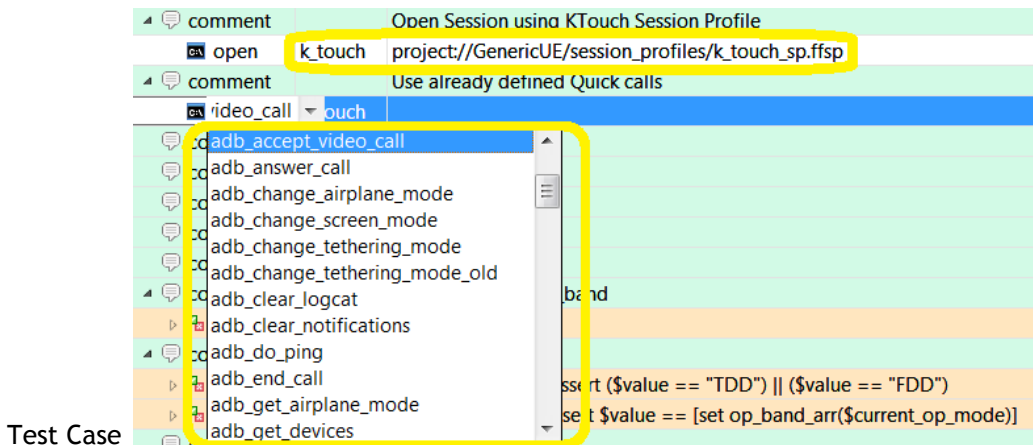
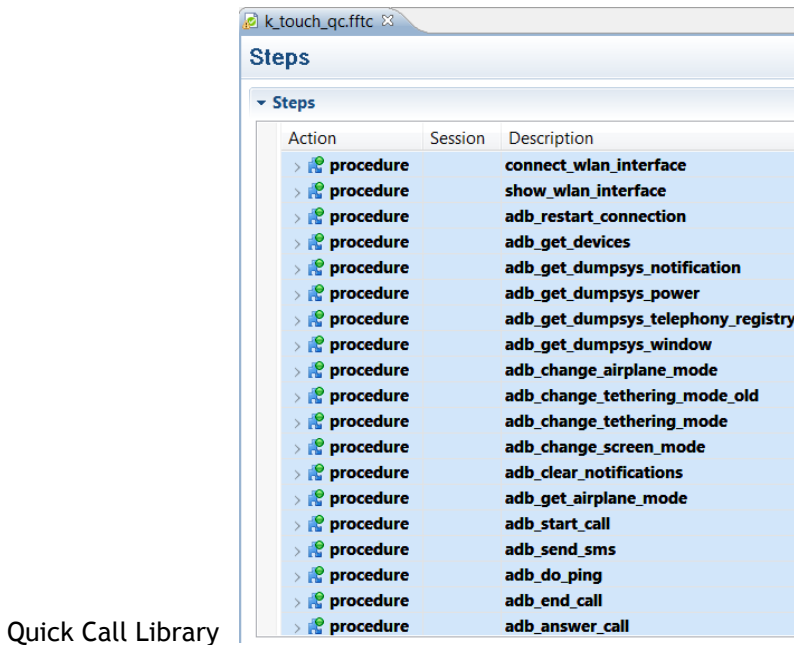
When multiple response maps apply, iTest applies the maps in priority order. Lower number are higher in priority (for example, 1 is higher priority than 100).

Priority: 50

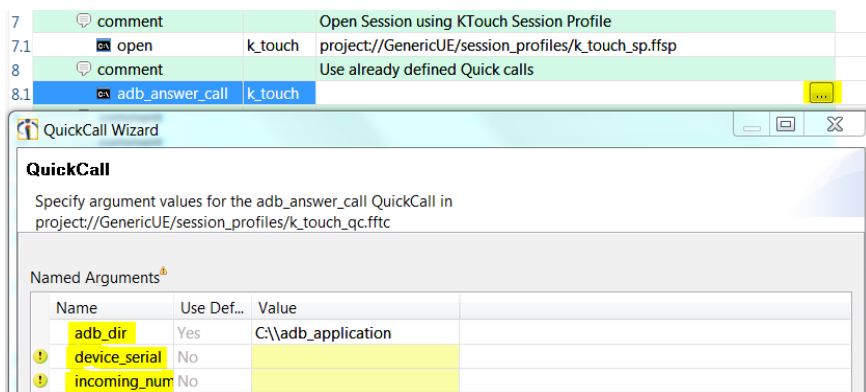
Overview Samples Applicability Queries Parsers Response Filters Pattern

## 2.8 Quick Calls Best Practices

iTest Quick calls is the term used to identify procedures that are associated with one type of session. All procedures that are created in the Quick call library of a Session Profile (inside the Test case associated with the Session Profile) will become available to call from other Test Cases when an instance of that Session is used.



After selecting the Quick call, the Attributes interface can be accessed by pressing on (...):



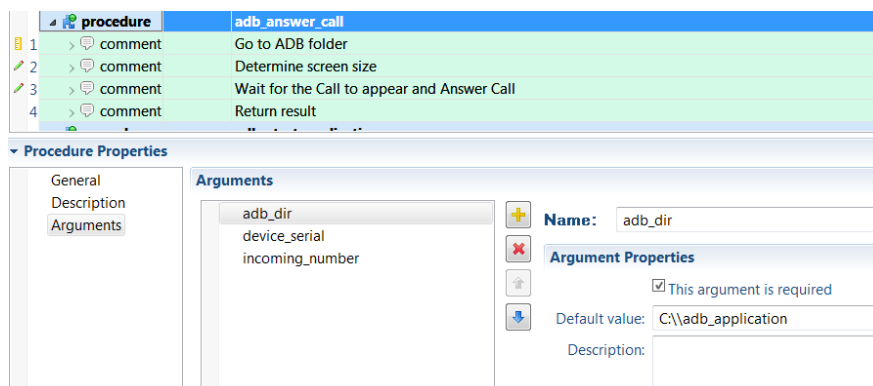
The Quick calls can be divided into two main categories: The first category is related to Procedures that are used to change configuration or set certain parameters (config) and the second one consists of Procedures used to extract data or information (show) from the Device or Tool.

When creating a new Quick call procedure the user should take into account the type of procedure, either config or show, and develop the logic of the procedure in such a way that:

- The config Quick call should return one success code based on the configuration result
- The show Quick call should return the output collected from the Device or Tool

## 2.9 Creating a Config Quick Call

1. Each procedure should only implement one independent action / feature (using one or multiple commands) so that the complexity of different features can be isolated from one procedure to another
2. Each procedure should define an interface (list of Arguments). Using these Arguments the procedure should treat multiple scenarios so that every possible configuration for the specific feature can be constructed using different values for each of the Arguments. The Arguments are defined under the Procedure Properties tab, Arguments sub-tab:



3. The procedure should maintain the success code throughout the execution. In implementing this requirement, the best practice is to start with a local variable (retCode) for which we set the “TRUE” value. During the configuration some steps may fail and this will lead to changing the local variable (retCode) to “FALSE”. At the end of the procedure the success code will be returned as an output of the procedure. Based on this success code the user can take action inside the test case by:
  - FailTest (if this configuration is critical for the test scenario, the test should be failed)
  - Display Warning (if this configuration is not critical a Warning message can be displayed and the test case can continue)
  - Ignore Failure

**Example 1:** k\_touch Command Prompt Quick call library: The procedure adb\_answer\_call is used to answer an incoming call by performing the following actions:

- Command Prompt changes directory to the ADB application folder. In this step we are setting the success code to “TRUE” and we initialize flags used during the procedure to 0.

procedure	adb_answer_call
comment	Go to ADB folder
analyze	
none	
assert	1
When True	
Eval	set retCode "TRUE"
Eval	set call_in_progress 0
Eval	set call_is_received 0
Eval	set call_incoming_number 0
When False	
command	\$session [string range \$adb_dir 0 1]
command	\$session cd \$adb_dir

- A loop is used for verifying if the UE is in ringing state and if the caller is the expected incoming number. If both of these conditions are met, the call\_in\_progress flag is changed from 0 to 1 and the loop is stopped.

comment	Wait for the Call to appear and Answer Call
for	{set i 0} {\$i < 10} {incr i}
comment	Wait for call to appear
sleep	2
comment	Verify if the call is in progress
adb_get_dumpsys_telephony_registry	\$session -adb_dir \$adb_dir -device_serial \$device_serial
analyze	
query	call_state()
assert	\$value == 1
When True	
Eval	set call_is_received 1
When False	
analyze	
query	incoming_phone_number()
assert	\$value == \$incoming_number
When True	
Eval	set call_incoming_number 1
When False	
comment	Verify if the correct call is received
analyze	
none	
assert	(\$call_is_received == 1) && (\$call_incoming_number == 1)
When True	
Eval	set call_in_progress 1
Break	
When False	
DeclareExecutionIssue	Information:Waiting for call from \$incoming_number

- If the call is in progress and the incoming number is correct, the UE will attempt to answer the call by swiping on the screen from on coordinate to another. After this action, an ADB dumpsys command is use to determine if the call is established on this end. Based on the result, the success code (retCode) is kept as “TRUE” or changed to “FALSE”
- If the call is not in progress or the incoming number is not correct the UE will not attempt to answer the call and will change the success code (retCode) to “FALSE”

- The procedure returns the success code (retCode: \$retCode) to the test case that called it

comment	If the call is in progress - Answer Call
if	\$call_in_progress == 1
then	
command	\$session adb -s \$device_serial shell input swipe [math.round [expr \$screen_width / 2]] [math.round
sleep	5
adb_get_dumpsys_telephony_registry	\$session -adb_dir \$adb_dir -device_serial \$device_serial
analyze	
query	call_state()
assert	\$value != 2
When True	
Eval	set retCode "FALSE"
DeclareExecutionIssue	Warning:Call is not in progress after answering
When False	
else	
comment	Call did not appear so cannot answer
analyze	
none	
assert	1
When True	
Eval	set retCode "FALSE"
When False	
comment	Return result
return	retCode: \$retCode

Example 2: For Command Prompt session - connect\_wlan\_interface procedure is used to initiate netsh connect command for a specific SSID and performs the following actions:

- The Initialization step is used for setting the success code to "TRUE" and verifying if a profile name was specified
- In the second step the netsh connect command is issued and the success code will be changed to "FALSE" based on the response (if "completed successfully" message is missing from the command's response)
- The procedure returns the success code (retCode: \$retCode) to the test case that called it

procedure	connect_wlan_interface
comment	Initialization
analyze	
none	
assert	1
When True	
Eval	set retCode "TRUE"
When False	
analyze	assert \$profile_name == "null"
comment	Issue netsh connect command
command	\$session netsh wlan connect ssid=\"\$ssid_name\" name=\"\$profile_name\" interface=\"\$nic_name\"
analyze	
contains	completed successfully
assert	\$value == 0
When True	
Eval	set retCode "FALSE"
When False	
comment	Return result
return	retCode: \$retCode



## 2.10 Creating a Show Quick Call

1. Each procedure should only implement one independent show command or extract output from a single web page.
2. Each procedure should define an interface (list of Arguments). Using these Arguments the procedure should treat multiple scenarios so that the required information will be extracted or the show command uses the correct parameters
3. The output from the show command or the output from the web page should be saved in a local variable and returned to the test case that called the Quick Call.
4. Every show Quick call should have one ore multiple Response Maps. The Response Map needs to be configured to apply only to this Quick call Action.

**Example 1:** For Command Prompt session: Procedure adb\_get\_devices is used to issue “adb devices” and send the output to the test case that called it:

- Initially the command prompt working directory is change to the ADB application folder

procedure		adb_get_devices
1	comment	Go to ADB folder
1.1	command	\$session [string range \$adb_dir 0 1]
1.2	command	\$session cd "\$adb_dir"

- The “adb devices” command is issued and the entire output is saved in the local variable named adb\_devices. Note: Saving the entire command output is performed by navigating to Step Properties > Other Postprocessing > Store Response. If “Store only text of the response” checkbox is checked, the return action will use \$local\_variable\_name. If this checkbox is unchecked, the return action will use [response local\_variable\_name], command which actually extracts the text from the local variable

2	comment	Issue all ADB commands
2.1	command	\$session adb devices
<b>Step Properties</b>		
General		
Timing		
Documentation		
Other Post-proce		
Expected Resp		
Global Analysis		
Store Respons		
<b>Store Response</b>		
Store the response in a variable that can be accessed later		
Store the response in variable: adb_devices		
<input type="checkbox"/> Make it global		
<input type="checkbox"/> Store only the text of the response		

- The last step is used for returning the collected output to the test case

3	comment	Return result
3.1	return	[response adb_devices]

Note: For this show Quick call we have create one block Response Map that extracts all serial numbers for the authorized devices.

The Response Map has the same name as the Quick call on which it is applied and has the following structure:

- Block: daemon\_header containing the first two lines of the output



Note: The Port Number is declared as Variable - Configuration in Token Properties

Note: This Block should appear at least once **is Required** - Configuration in Block Properties

- Container: devices with the following components inside:
  - o Block: header containing the third line of the output



- o Block: entry containing one entry sample



Note: The device word is devlared as Variable so this line should match also entries for which the state is “unauthorized” or different from “device”

Note: The list of serial numbers (\*)will be extracted under the name device\_serial - Configuration in Token Properties

Note: This Block should appear at least once **is not Required**

Note: This Block may appear multiple times **is Required** - Configuration in Block Properties

In order to make this response map to apply to the correct quick call please ensure that the following requirements are met:

1. The Response Map is placed in the response\_maps sub-folder of the correct project (in this case GenericUE)
2. Verify that the corresponding Session Profile (in this case k\_touch\_sp.ffsp) is correctly configured related to Response Maps location in Misc tab, Response Map Library sub-tab

3. Verify Applicability tab in the Response Map:
  - a. Session type should be correct (in this case Command Prompt)
  - b. Action field should be correct - the name of the Quick call on which we expect this Response Map to be applied (in this case adb\_get\_devices)

**Automatic Mapping Applicability**

**Response Map Applicability**

Specify when to use this response map. You can use wildcards and regular expressions to enhance your Action, Target, and Command settings.

☒ If this response map is in a library, then apply the map to a response when all of the following conditions are met.

Select from the list only if you want to restrict this response map to the selected applications.

Session types: ☐ Chat (XMPP) ☒ Command Prompt ☐ Database Client ☐ File

Action: adb\_get\_devices

Target:

Overview Samples Applicability Queries Parsers Response Filters Block

Once all these configurations are done, every time when we are using this Quick call (adb\_get\_devices) from a Test Case for the k\_touch session, the output of the Quick call will be parsed and the defined token (device\_serial) will extract all the serial numbers present in the obtained output.

All the matched values can be identified in the Response tab - a blue rectangle will be surrounding each extracted value:

Response Execution Console Properties SNMP Traps NI

```
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
D5971A1772400918      unauthorized
```

Response Execution Console Properties SNMP Traps NI

```
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
D5971A1772400918      device
D5971A1772402228      device
D5971A1772400320      device
```

## 2.11 Creating a Response Map

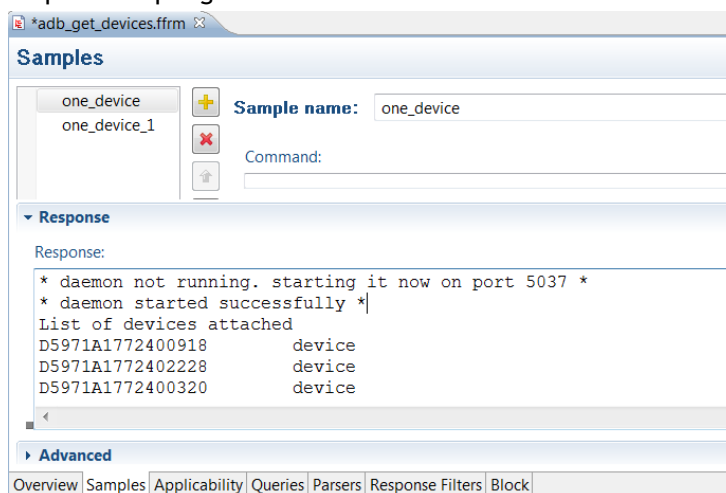
When creating a Response Map the user should start by observing the output that will be parsed inside the Response Map. Different types of output are calling for different types of Response Maps (Table, Pattern, Block) and selecting the best match may lead to significant less work when creating and maintaining the Response Map.

Usually, if the output obtained looks like a table, a Table Response Map will be recommended. If the output is large or we are interested only in extracting values from certain lines the best match will be the Pattern Response Map (this option is very similar with Regular Expressions) - by using Patterns, iTest will try to find all outputs that match each of the patterns defined and will ignore the rest of the output. When using complex queries (for example `igmp_packets_by_igmp_ip_address_and_interface_name`) or dealing with well structured outputs Block mapping may be the best approach. One drawback for Block mapping is that if the output structure changes the mapping will fail - this is why all possible types of outputs should be taken into account when creating a Block Response Map.

When creating a new Response Map, please follow this procedure:

1. Place the Response Map file in the corresponding Project under the `response_maps` subfolder
2. Naming convention: The Response Map should use the same name as the Quick call for which is created
3. After creating the Response Map file, add outputs specific to the corresponding Quick call or command and save them as Samples in the Response Map Sample Tab. These outputs will be used to verify the correctness of the Response Map logic. If an Error icon appears after modifying a Response Map this means that the mapping failed for one of the Samples and this may lead to Test Case failure.

Maintaining a set of Samples is essential in keeping a history of the outputs types obtained since the addition of the Response Map until the current time and after each change in the Response Map logic the user needs to make sure that no Error is present.

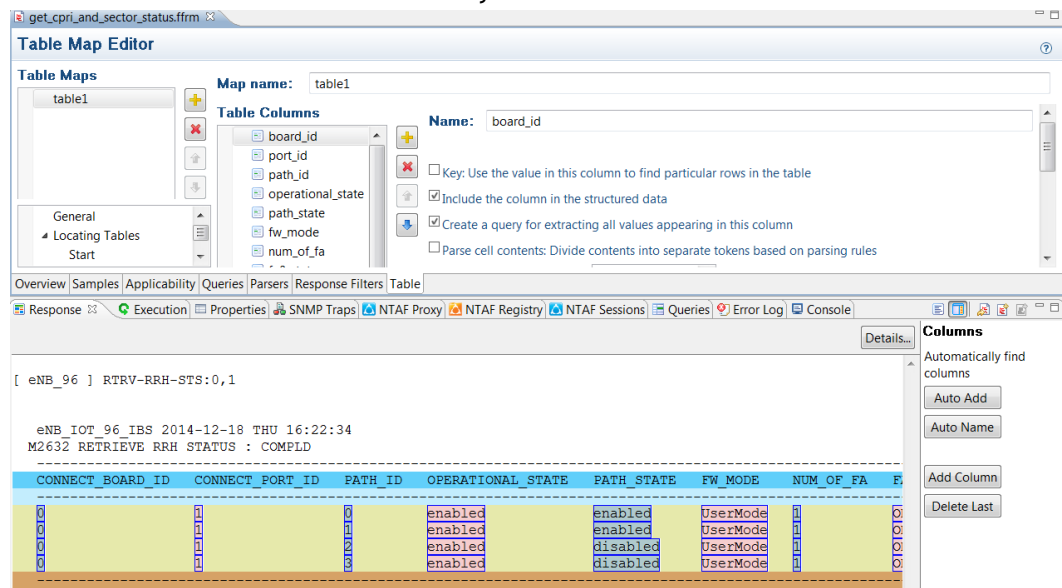


4. After adding the Sample outputs, the user must select the Response Map type from the Overview Tab:
  - a. Table
  - b. Pattern
  - c. Block
5. After selecting the type of the Response Map, a new Tab will appear (named Table, Pattern or Block), based by the user's selection. Response Map logic needs to be added by the user based on the output and on what queries it needs to create.

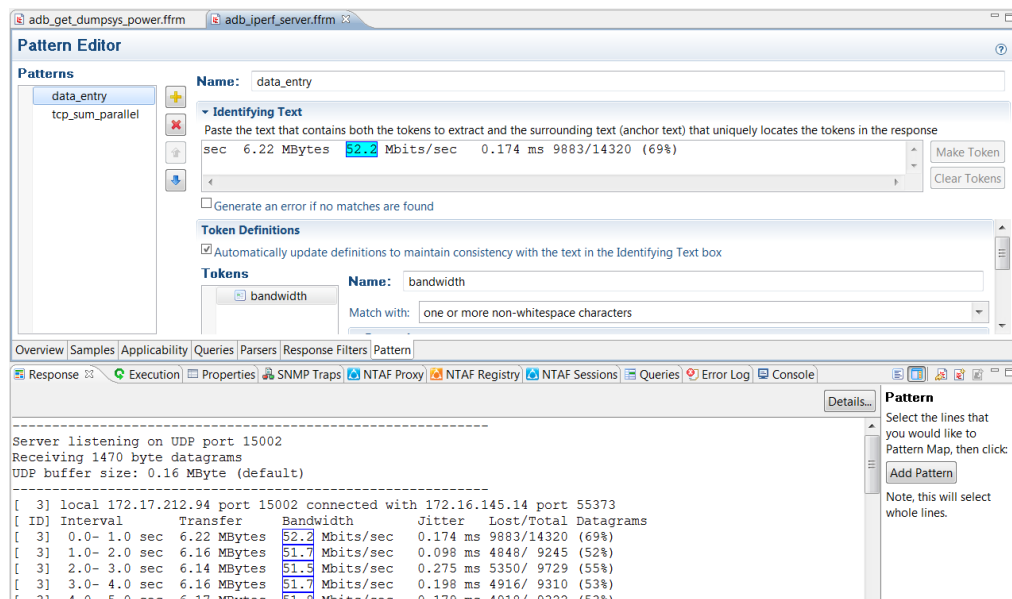
General information:

- a. Table: Add one or more tables and specify table Header / Footer or Start and End location; Define columns by using Whitespace, Special Characters or Fixed Length as Delimitation criteria.

Column names will become Queries;  
One Column can be identified as the key

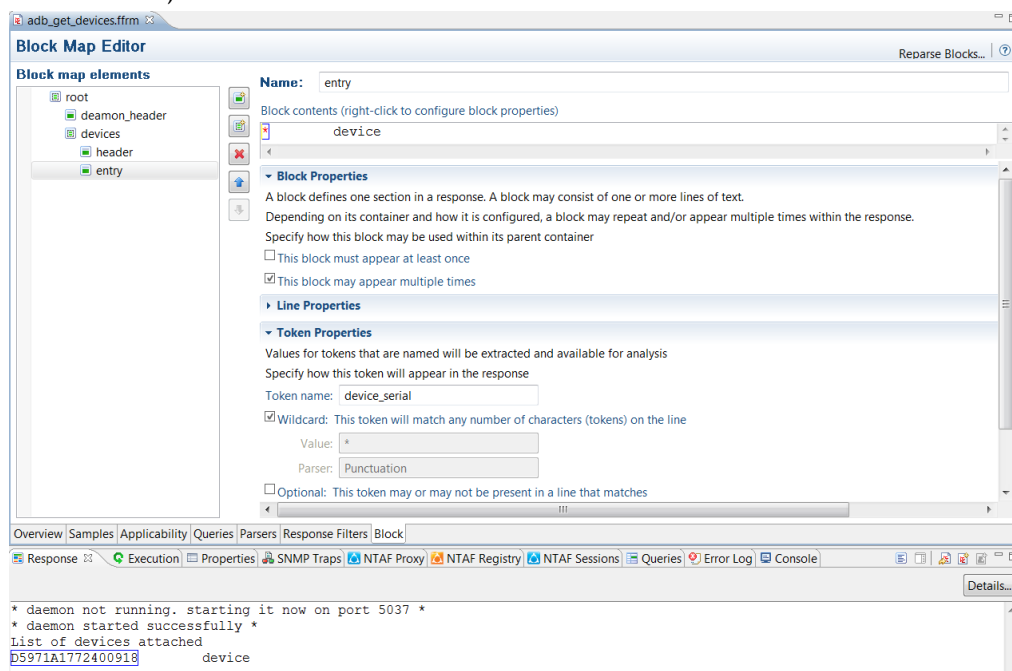


- b. Pattern: Add one or multiple patterns and create Tokens by highlighting the required values and clicking Make Token  
Inside each Pattern can be defined multiple tokens and one of these tokens can be identified as the key (note that this will be the key only for tokens extracted using this pattern. Tokens extracted using other patterns will not be affected).  
Pattern matching is very similar to Regular Expressions so the User can modify Matching criteria for both the anchors and the extracted token.



- c. Block: The block mapping requires creating a structure for the entire output that will be matched. Tokens can be extracted and the matching criteria can be modified by using predefined token properties (Wildcard, Variable, Optional, Key).

There are multiple levels for controlling the output form: Block Properties, Line Properties (controls one line from the Block), Token Properties (controls one token from one line).



6. The last step in creating a Response Map is to fill in the Applicability information available in the tab with the same name: Applicability

The user needs to select the correct Session type, the correct Action Name and can change the Priority if multiple Response Maps were defined for the same Action (lower numbers have higher priority)

**Automatic Mapping Applicability**

**Response Map Applicability**

Specify when to use this response map. You can use wildcards and regular expressions to enhance your Action, Target, and Command settings.

☒ If this response map is in a library, then apply the map to a response when all of the following conditions are met.

Select from the list only if you want to restrict this response map to the selected applications.

Session types: ☐ Tcl Shell ☒ Telnet ☐ UDP ☐ VMware vSphere Client

Action:

Target:

Command:

Specify how to interpret the Action, Target, and Command settings.

Compare properties using:

**Map Priority**

When multiple response maps apply, iTest applies the maps in priority order. Lower number are higher in priority (for example, 1 is higher priority than 100).

Priority:

Overview | Samples | Applicability | Queries | Parsers | Response Filters | Pattern

## 2.12 Test Scripts Reviewing Process

Whenever a new test script gets developed it should get officially approved. This is usually done following a technical and formal review process. The work done on the automation solution is reviewed in order to identify issues or limitations and provide feedback. This will help further development and test database maintenance.

Whenever a test script fails there could be 2 main reasons:

- a) script failure
- b) device/feature/test expected result failure

The first of these problems can be prevented by planning a formal review for all new test scripts before integration in the regression suite happens. Our proposal is considering a formal process for the technical review for every new test case that will be integrated.

Technical review details:

- a) It is led by the trained moderator but can also be led by a technical expert;
- b) It is often performed as a peer review without management participation;

- c) In practice, technical reviews vary from quite informal to very formal.

A formal review process consists of six main steps:

- a) Planning
- b) Kick-off
- c) Preparation
- d) Review meeting
- e) Rework
- f) Follow-up

Reviewing scripts process workflow:

- a) A new test case is developed in respect to the existing test documentation; every test step from description should be written in clear with very strictly defined validation criteria;
- b) After the test completion make sure that the test works on the developing machine;
- c) Before integration in the regression suite make sure that the test can run on the Automation Manager machine (testbed is different > topology will be different);
- d) Plan a review meeting via email when every aspect from above is prepared;
- e) During the review meeting the following should be covered:
  - both the script logic (in respect to the test description) and scripting best practice (in respect to script template) should get approval;
  - at least 2 team members should attend this meeting: the owner of the test case and the automation engineer who is responsible for test script maintenance or regression integration (during the initial phase a Spirent engineer can fill this position);
  - Send review meeting minutes email with changes that need to be done;
  - make sure to follow all updates after meeting;
- f) Test script author should take accountability for implementing the review comments;
- g) After updating the script (if required) a second informal review meeting (could be via email) should make this test case getting approved for running in regression (usually a test script is considered completed when it gets integrated in the regression suite).