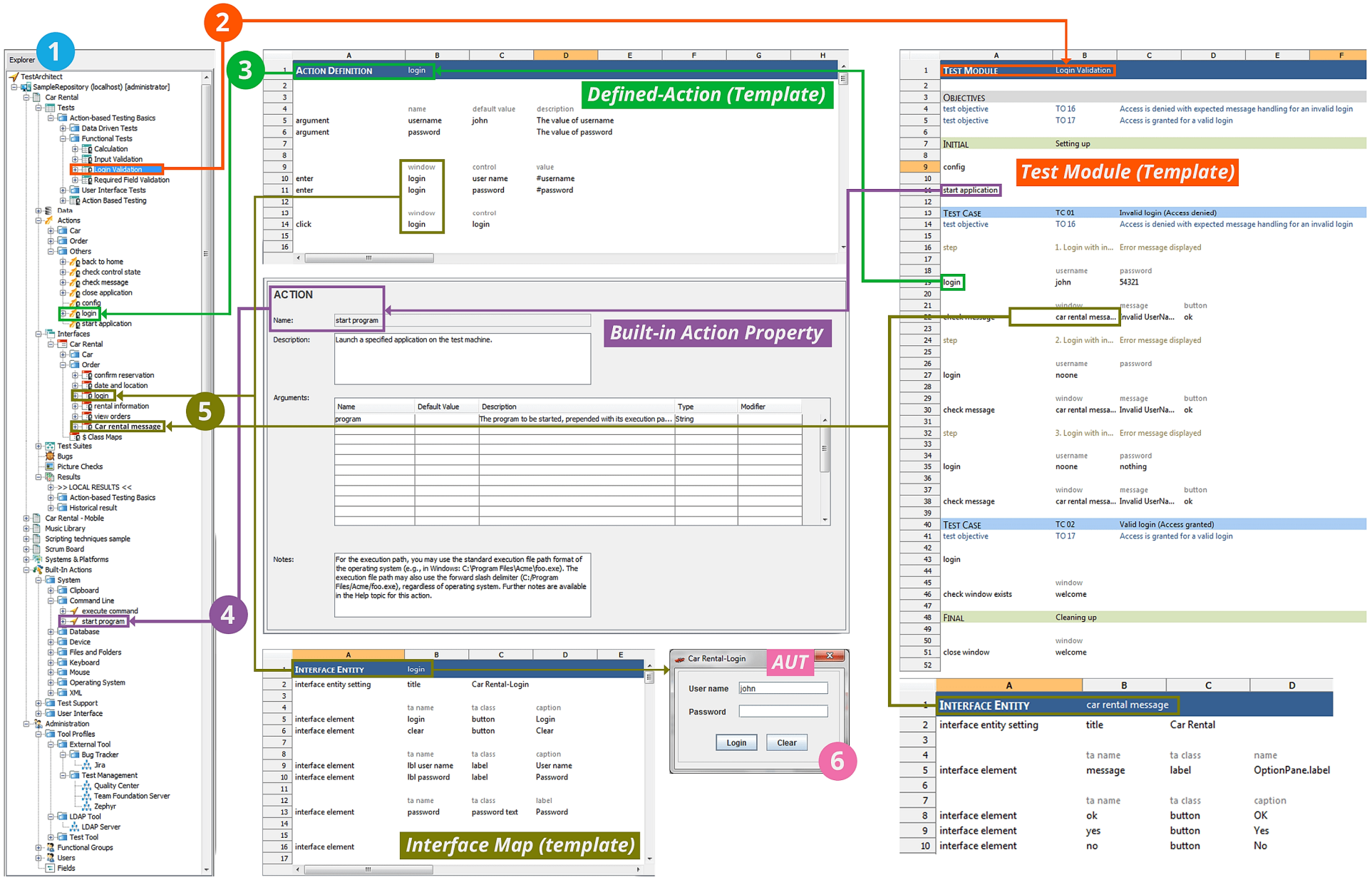


1 TestArchitect Explore Tree

TestArchitect Explorer Tree shows all open repositories, their respective projects, and all the items within each project.

2 Test Module

A worksheet containing test requirements and test cases which narrowly define the scope. Test modules should be designed to run independently from each other, while the test cases within a test module can have dependencies among themselves.

3 Defined-Action

User-defined actions are actions that you create in TestArchitect's Action Based Testing language, and consist of sequences of actions that typically relate to a single business logic function in the application under test.

4 Built-In Action

TestArchitect includes a library of built-in actions to perform a myriad of functions. There are three categories of built-in actions: System actions, Test Support actions and User Interface actions

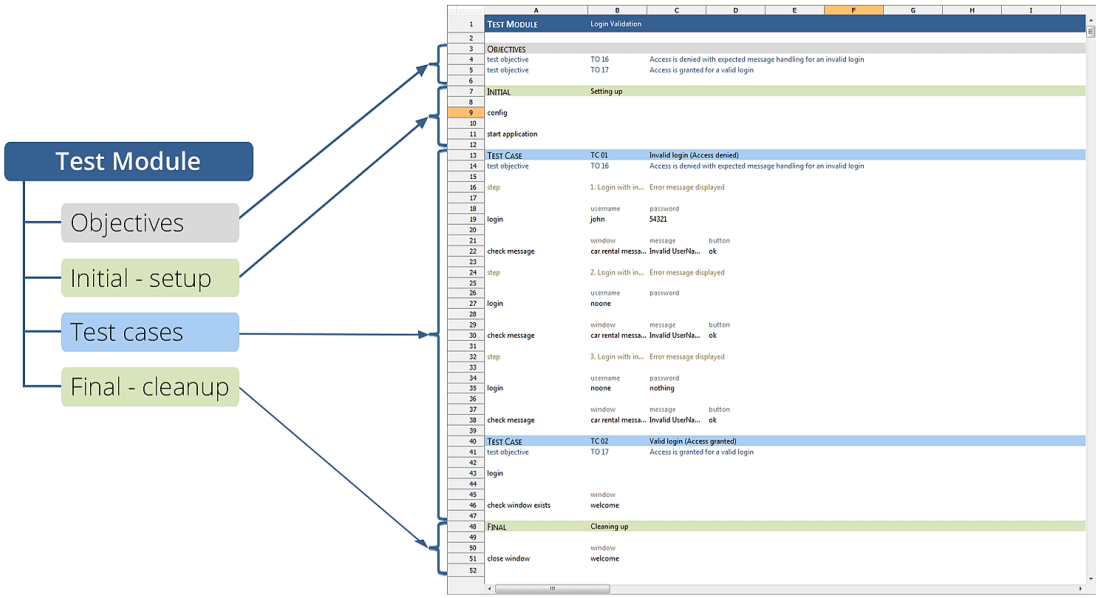
5 Interface Entity

A TestArchitect project item used to represent a single dialog box, window, or full-screen item of the AUT's GUI interface. An interface entity appears in the TestArchitect explorer tree as a child of an interface.

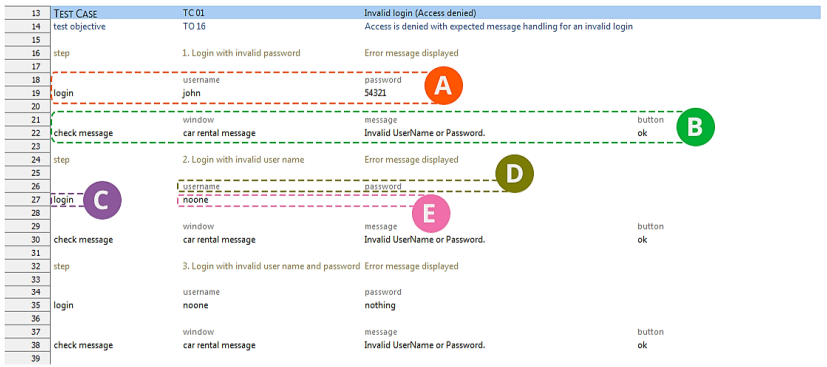
6 AUT

The application that is being tested for correct operation.

1 Create a Test Module with Test Module template



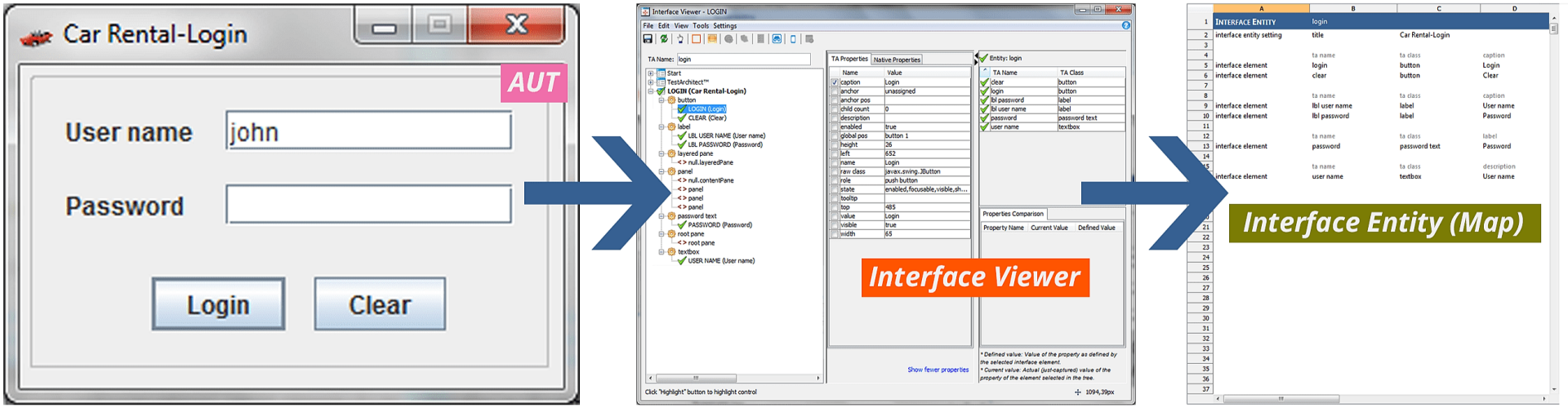
2 Write Action Line using Defined Action or Built-in Action



- A Activity Action Line
- B Verification Action Line
- C Action Name (Verb or Verb + Object)
- D Argument Name
- E Argument Data

3 Action references Interface Entity (Map) to communicate with UI of **AUT**

Use the Interface Viewer tool to capture AUT UI information and store it in Interface Entity (Map)



ACTION BASED TESTING LANGUAGE (ABTL) QUICK REFERENCE CARD

Variables

There are two types of variables in TestArchitect: **global variable** and **local variable**.

Local variables are variables that are declared within a specific section of test modules or actions. They are initiated within a limited scope, and can only be seen in a particular section.

Action to define a local variable, and assign a value to it

		name	value
17			
18	local variable	b	999
19			
20		variable	value
21	set variable	b	888

Variable value

Variable value

Action to assign a value to an existing local or global variable

Global variables are variables with global scope. They can be accessed throughout all test modules and invoked actions within one execution run.

Action to define a local variable, and assign a value to it

		name	value
17			
18	global variable	a	1234
19			
20		variable	value
21	set variable	a	5678

Variable value

Variable value

Action to assign a value to an existing local or global variable

Expressions & Functions

An expression (prefixed by an expression indicator #) is any combination of literal values, variables, operators, operands and functions that follows a set of rules, and which needs to be evaluated before it can be used.

A Function: A predefined, named formula that performs a specific opeartion and returns values needed by your test.

More information: *testarchitect.logigear.com/onlinehelp/#TA_Automation/Topics/The_test_language_functions.html*

```
15  // declare local variables and initialize its value
16  // Use two forward slashes to denote a comment.
17                                  name        value
18  local variable                  a           Logigear Corporation
19
20  // report the substring containing 8 characters from the left
21                                          Arguments
22                                  text
23  report                          # left (a, 8)
24                          Function name
25  // declare local variables and initialize its value
26
27                                  name        value
28  local variable                  width       20
29  local variable                  height      40
30
31                                  text
32  report                          #" The area of the rectangle is: " & width* height
33
                        Expression indicator        Expression operators
```

```
⊖ 18: local variable  [name:a] [value:Logigear Corporation]
        a -> Logigear Corporation
⊖ 23: report  [text:# left (a, 8)]
        # left (a, 8)->Logigear
        Logigear
⊖ 28: local variable  [name:width] [value:20]
        width -> 20
⊖ 29: local variable  [name:height] [value:40]
        height -> 40
⊖ 32: report  [text:#" The area of the rectangle is: " & width* height]
        #" The area of the rectangle is: " & width* height->The area of the rectangle is: 800
        The area of the rectangle is: 800
```

QUICK REFERENCE CARD

Conditional actions

Begin a block of action lines which are executed only if a specified condition is satisfied.

```
15
16  get system date   year >> y   month >>m   day >>d   weekday >>wd
17
18                    condition
19  if                # y ="2018"
20
21                    text
22  report            The FIFA World Cup 2018 will be held in Russia
23
24                    condition
25  else if           #y="2020"
26
27                    text
28  report            The UEFA Euro 2020 will be held in thirteen cities in thirteen different European countries during the summer of 2020.
29
30  end if
31
32                    condition
33  if                # d=24 and m=12
34
35                    text
36  report            Merry Christmas.
37
38  end if
39
40                    condition
41  if                #wd ="Saturday" or wd="Sunday"
42
43                    text
44  report            Happy weekend.
45
46  else
47                    text
48  report            Have a nice day.
49
50  end if
```

```
⊖ 16: [13:50:24/ 15ms]  get system date  [year: >> y] [month: >>m] [day: >>d] [weekday: >>wd]
      y -> 2016
      m -> 12
      d -> 13
      wd -> Tuesday
⊖ 19: if  [condition: # y ="2018"]
      # y ="2018" -> false
⊖ 25: else if  [condition: #y="2020"]
      #y="2020" -> false
  30: end if
⊖ 33: if  [condition: # d=24 and m=12]
      # d=24 and m=12 -> false
  38: end if
⊖ 41: if  [condition: #wd ="Saturday" or wd="Sunday"]
      #wd ="Saturday" or wd="Sunday" -> false
  46: else
  48: Have a nice day.
  50: end if
```

Loop actions

A loop is a statement, or set of statements, that are repeated for a specified number of times or until some condition is met

while / end while

Denotes the beginning of a **while/end while** loop. Evaluates a conditional expression to determine whether execution is to continue with the action lines directly below it, or with the lines following the matching **end while**.

```
15
16  local variable    name: temp count    value: 1
17
18                    condition to run:
19  while             #temp count < 3
20
21                    text:
22  report            #temp count
23
24                    variable: temp count    value: #temp count + 1
25  set variable
26
27
28  end while
29
```

```
⊖ 16: [14:07:29/ 15ms]  local variable  [name: temp count] [value: 1]
      temp count -> 1
⊖ 19: while  [condition to run: #temp count < 3]
      #temp count < 3 -> true
---- WHILE LOOP, START ----
  22: 1
⊖ 25: [14:07:29/ 1ms]  set variable  [variable: temp count] [value: #temp count + 1]
      #temp count + 1 -> 2
      temp count -> 2
⊖ 28: end while
---- WHILE LOOP, NEXT CYCLE ----
  22: 2
⊖ 25: [14:07:29/ 1ms]  set variable  [variable: temp count] [value: #temp count + 1]
      #temp count + 1 -> 3
      temp count -> 3
⊖ 28: end while
---- WHILE LOOP, DONE ----
```

Loop actions

Repeat / until

Denotes the beginning of a **repeat/until** loop.

```
15
16  local variable      name          value
                        temp count    1
17
18  repeat
19
20                      text
21  report              #temp count
22
23                      variable      value
24  set variable        temp count    # temp count + 1
25
26                      condition to stop
27  until               #temp count = 3
28
```

```
16: [14:12:30/ 1ms]  local variable  [name: temp count] [value: 1]
      temp count -> 1
18: repeat
---- REPEAT LOOP, NEXT CYCLE ----
  21: 1
24: [14:12:30/ 1ms]  set variable  [variable: temp count] [value: # temp count + 1]
      # temp count + 1 -> 2
      temp count -> 2
27: [14:12:30/ 1ms]  until  [condition to stop: #temp count = 3]
      #temp count = 3 -> false
---- REPEAT LOOP, NEXT CYCLE ----
  21: 2
24: [14:12:30/ 1ms]  set variable  [variable: temp count] [value: # temp count + 1]
      # temp count + 1 -> 3
      temp count -> 3
27: [14:12:30/ 1ms]  until  [condition to stop: #temp count = 3]
      #temp count = 3 -> true
---- REPEAT LOOP, DONE ----
```

Operators

Comparison

Symbol	Operation	Priority
=	equal to	4
<> , !=	not equal to	4
>	greater than	4
>=	greater than or equal to	4
<	less than	4
=<	less than or equal to	4

Logical

Symbol	Operation	Priority
not	Value is **TRUE** if its operand is **FALSE**	5
and	Value is **TRUE** if and only if both sides of the **and** operator are **TRUE**	6
or	Value is **TRUE** if either side of the **or** operator is **TRUE**	7

Settings

Settings are used to steer the automation process. They control how your action lines are handled by the TestArchitect interpreter or automation.
More information: *testarchitect.logigear.com/onlinehelp/#TA_Automation/Topics/bia_setting.html*

```
7   INITIAL                 Setting up
8
9   // set the maximum wait time for a control or HTML element to become available or, depending upon the action involved, unavailable.
10
11                          setting         value
12  setting                 object wait     30
13
```

Dynamic identifiers

A value for a window/control argument which, instead of using a TA name, directly identifies a UI element through its TA class and TA property values.
More Information: *testarchitect.logigear.com/onlinehelp/#TA_Help/Topics/The_test_language_dynamic_identifiers.html*

```
15  // Enter the name "john" into the 'User name' field
16                          window          control                          value
17  enter                   login           [ta class = textbox , description = user name]   john
18
19  // Click the 'Login' button
20                          window          control
21  click                   [title = Car Rental-Login]   [ta class = button , caption = Login]
22
23  // If successful, 'Login' window disappears
24                          window
25  check window not exists [title = Car Rental-Login]
```

A dynamic identifier is surrounded by a pair of square brackets.

```
17: [11:06:31/ 453ms]  enter   [window: login] [control: [ta class = textbox , description = user name]] [value: john]

21: [11:06:32/ 187ms]  click   [window: [title = Car Rental-Login]] [control: [ta class = button , caption = Login]]

⊖ 25: [11:06:32/ 31ms]  check window not exists   [window: [title = Car Rental-Login]]

    check       window not exists
    expected    [title = Car Rental-Login] not exists
    recorded    [title = Car Rental-Login] not exists
    result      passed
```

Wildcards

A specific regular expression pattern that can be used to substitute for any other character or characters in a string, allowing for flexibility in pattern matching.
More Information: *testarchitect.logigear.com/onlinehelp/#TA_Tutorials/Topics/Wildcards.html*

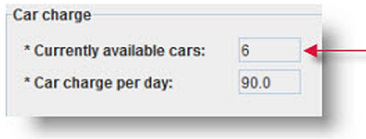
```
16  //Click a text whose string value begins with 'Car - 5'
17  Wildcards ".*", represent "matching any contiguous set of characters".
18                          window          control
19  click                   scrumboard      [ta class=li , text= {Car - 5.*}]
```

```
19: [14:14:37/ 546ms]  click   [window: scrumboard] [control: [ta class=li, text={Car - 5.*}]]
```

Checks

Any point in a test procedure in which any type of check action exists. check actions are the only actions that register pass/fail results.

Example



Action Lines

```
19
20                          window          control
21  check control exists    view cars       available car
22
```

```
⊖ 21:  check control exists   [view cars] [available car]

    check       control
    expected    available car
    recorded    control found
    result      passed
```

Error handling & recovery

TestArchitect provides a number of mechanisms to support error handling and recovery to allow for tests to continue to run after encountering unanticipated errors, warnings or test failures.

on error

Specify the execution path to take in the event of an error.

For example, in the event of an error, you would like TestArchitect to abandon the current test case and continue with the next test case in the same test module, specify the 'exit test case' argument in the test procedure.

7	INITIAL	Setting up	
8			
9		behavior	
10	on error	exit test case	
11			
12	TEST CASE	TC 01	User is able to select car
13	test objective	TO 01	User is able to select car
14			
15		car	number of cars
16	select car	Compact/Toyot...	1
17			
18		car	
19	check car selected	Compact/Toyota Prius	
20			
21			
22	TEST CASE	TC 02	User is able to search order
23	test objective	TO 02	User is able to search order
24			
25		value	
26	search order	RCV0	
27			
28		field name	expected value
29	check order information	First Name	John
30			

```
⊖ INITIAL, Setting up
   10: on error  [behavior:exit test case]
⊖ Test case: TC 01, User is able to select car
   Error(s) at line(s):16:9
   13:  test objective TO 01 - User is able to select car
⊖ (E) 16: select car  [car:Compact/Toyota Prius] [number of cars:1]
      (the remaining lines of the test case will be skipped)
   ⊖ (E) 9: click tree node  [window:car selection] [tree:car select tree] [node path:# "Car Types/" & car]
         # "Car Types/" & car->Car Types/Compact/Toyota Prius
      ERROR:  No window found matching "car selection" within the timeout of "20" seconds.
              Please check the interface definition for "car selection", or set a longer timeout with the "window wait" setting.
      <<<stopping action, returning to test module>>>
⊖ Test case: TC 02, User is able to search order
   23:  test objective TO 02 - User is able to search order
⊕ 26: search order  [value:RCV0]
⊕ 29: check order information  [field name:First Name] [expected value:John]
```

on error action

Specify the action to be invoked in the event of an error.

As an example, the following is a simple error handler that, when called, merely captures the screen at the time of the error, saving it to a designated jpg file

7	INITIAL	Setting up	
8			
9	//set notice level to 0 to show no warning message in case of UI navigational problem		
10		level	
11	set notice level	0	
12			
13	//specify the action to run when error happens		
14		action	
15	on error action	error handler	
16			
17			
18	TEST CASE	TC 01	User is able to view order
19	test objective	TO 01	User is able to view order
20			
21		window	control
22	click	welcome	view orders
23			
24		window	
25	check window exists	view orders	
26			

1	ACTION DEFINITION	error handler
2		
3	//Capture screen	
4		
5		image location
6	capture screen	D:\Images\failure.jpg
7		
8	// close application	
9		
10		window
11	close application	welcome

```
⊖ [13:48:58/ 93ms]  INITIAL, Setting up
   11: [13:48:58/ 1ms]  set notice level  [level: 0]
   15: [13:48:58/ 93ms]  on error action  [action: error handler]
⊖ [13:48:58/ 10359m:19s:245ms]  Test case: TC 01, User is able to view order
   Failed at line(s): 25
   Error(s) at line(s): 22
   19:  test objective: TO 01 - User is able to view order
⊖ (E) 22: [13:48:58/ 20s:296ms]  click  [window: welcome] [control: view orders]
ERROR:  No matching UI object found for "view orders" within the timeout of "20" seconds. Please check
        the interface definition for "view orders", or set a longer timeout with the "object wait" setting.
   Executing action 'error handler', as specified in 'on error action'.
   >>> EXECUTED ACTION: error handler <<<
⊖ 22: [NA/ 10358m:38s:747ms]  error handler
   6: [13:49:18/ 125ms]  capture screen  [image location: D:\Images\failure.jpg]
⊕ 11: [13:49:18/ 2s:340ms]  close application  [window: welcome]
⊖ 25: [13:49:20/ 20s:202ms]  check window exists  [window: view orders]
   check      window exists
   expected   view orders exists
   recorded   view orders not exists
   result     failed
```

ACTION BASED TESTING LANGUAGE (ABTL) QUICK REFERENCE CARD

Error handling & recovery

on failure action

Specify the action to be invoked in the event of a check failure from any check-type action.

The following example, when called, captures the screen in the event of a check failure, saving it to a jpg file

```
7   INITIAL                    Setting up
8
9                              action
10  on failure action          failure handler
11
12  TEST CASE                  TC 01          User is able to login
13  test objective             TO 01          User is able to login
14
15                             username       password
16  login                      john
17
18  //check if there is a Welcome link in Home page.
19
20                             window         control
21  check control exists       home           welcome
22
```

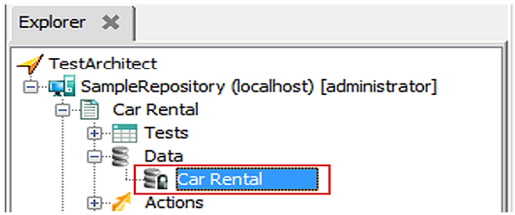
```
1   ACTION DEFINITION          failure handler
2
3
4   //Capture screen on failure
5
6                              image location
7   capture screen             D:\Images\failure.jpg
```

```
[13:57:53/ 437ms] INITIAL, Setting up
  10: [13:57:53/ 437ms] on failure action [action: failure handler]
[13:57:53/ 10367m:56s:716ms] Test case: TC 01, User is able to login
  Failed at line(s): 21
  13: test objective: TO 01 - User is able to login
  16: [13:57:53/ 1s:467ms] login [username: john] [password:]
  21: [13:57:55/ 20s:779ms] check control exists [window: home] [control: welcome]
      check      control exists
      expected   welcome exists
      recorded   welcome not exists
      result     failed
    Executing action 'failure handler', as specified in 'on failure action'.
    >>> EXECUTED ACTION: failure handler <<<
  21: [NA/ 10367m:34s:470ms] failure handler
    7: [13:58:16/ 453ms] capture screen [image location: D:\Images\failure.jpg]
```

Data set

A Data set is a collection of data. It contains rows of values that can be retrieved by an automated test and acted on sequentially.

Data sets are stored in the Data subtree of the TestArchitect explorer tree, and can be organized into folders and subfolders.



A data set worksheet typically resembles the following.

Car Rental

Data Lines | Information

	A	B	C	D	E
1	DATA SET	Car Rental			
2					
3		duration	country pickup	state pickup	city pickup
4	row	3	Canada	British Columbia	Kamloops
5	row	1	United States	Texas	Houston
6	row	2	United States	Florida	Brossard
7	row	5	United States	California	San Diego
8	row	4	Canada	Manitoba	Dauphin
9					
10		name	criterion		
11	filter	long	duration >= 4		
12					

Sample test script:

```
14
15  //Use a data set with a specified filter
16
17                             name           filter
18  use data set               /Car Rental    long
19
20                             text
21  report                     # city pickup
22
23  repeat for data set
24
25  //Use a data set with a filter expression directly within the 'filter' argument
26
27                             name           filter
28  use data set               /Car Rental    duration >= 4
29
30                             text
31  report                     # city pickup
32
33  repeat for data set
```

Sample test result:

```
  18: use data set [name: /Car Rental] [filter: long]
    21: San Diego
  23: repeat for data set
    21: Dauphin
  23: repeat for data set
      ---- end of cycle
  28: use data set [name: /Car Rental] [filter: duration >= 4]
    31: San Diego
  33: repeat for data set
    31: Dauphin
  33: repeat for data set
      ---- end of cycle
```