Enterprise Architect

User Guide Series

Testpoints

How to use Testpoints? In Sparx Systems Enterprise Architect, apply constraints on model object behavior to applications; one Test Domain verifies many applications, with tolerance to code change, or no change to source code if behavioral rules alter.

Author: Sparx Systems
Date: 2020-09-07
Version: 15.2

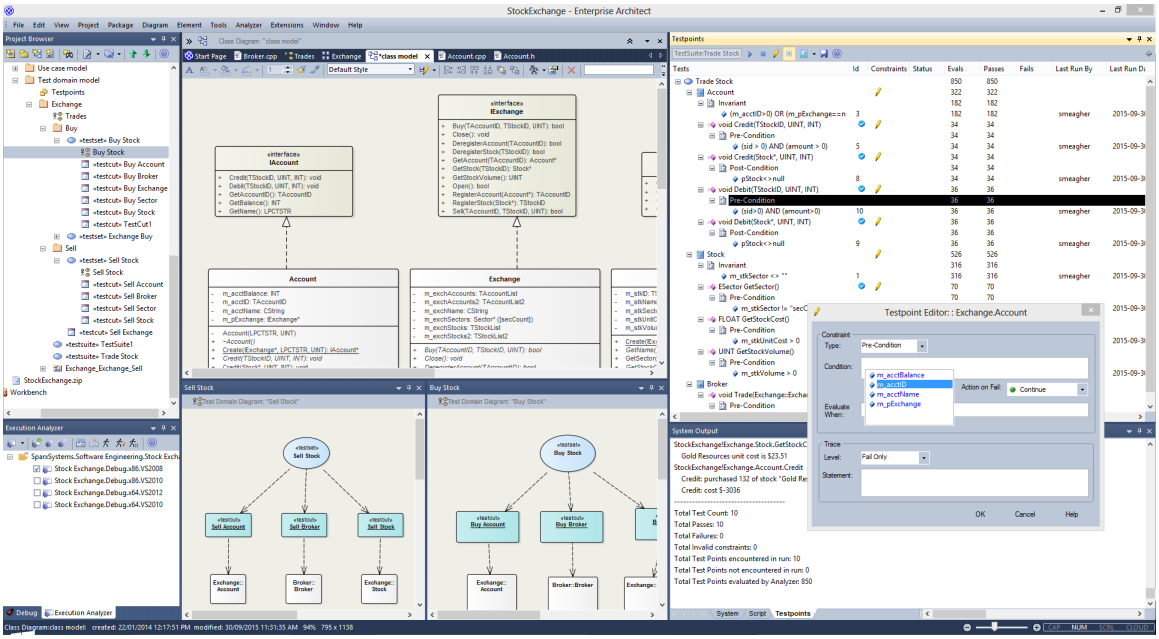
CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Testpoints

Testpoints present a scheme by which constraints and rules governing the behavior of objects can be taken from the model and applied to one or more applications. The advantages that schemes such as this offer are tolerance to code changes - adding and subtracting lines from a function has no effect on the constraints that govern it. Another advantage is that changes to the behavioral rules do not require a corresponding change to any source code; *meaning nothing has to be re-compiled!*

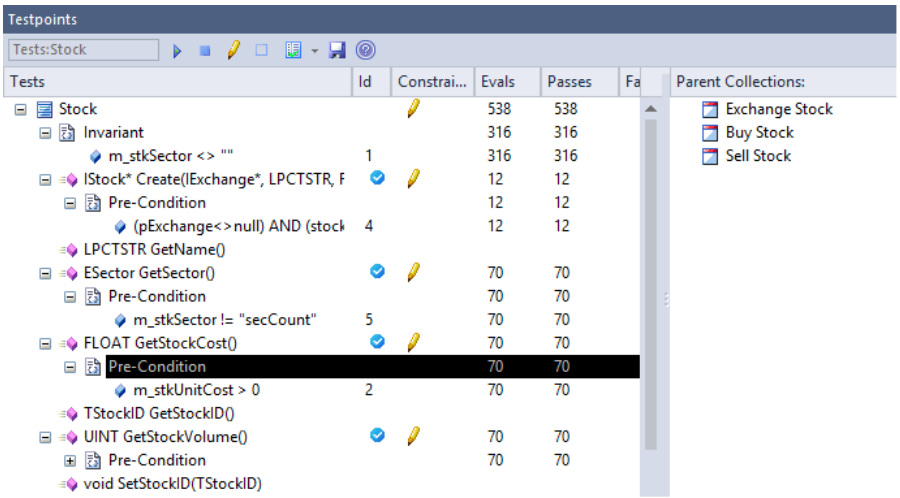
Also, the ability to verify multiple applications using a single test domain is a simple rather than onerous matter. The Test Domain is a both a logical and relational model; constraints in the Class model can be partitioned with Test Cuts. These can be aggregated simply into Test Sets and Test Suites using connectors. Due to the decoupling of the Test Domain from the codebase, it is a simple choice of buttons to run a program normally, or run it for a specific Test Domain. This system also delivers practical benefits in that no instrumentation is required at all. Test results are displayed in the report window during the run, in real-time, as the program runs. These results can be retained, and reviewed at any time in the 'Test Details' dialog or using Enterprise Architect's documentation features.



Features

Feature	Details
Testpoint Composition	Testpoint composition is performed using the Testpoints window. The Testpoints window is context-sensitive and displays the Test Domain for the selected element in either the Browser window or diagram. Selecting a single Class will display the Class structure. A 'pencil' icon is displayed against Classes and methods that have existing constraints. When you select a Test Cut, Set or Suite Test, the Testpoints window displays the entire Domain structure, including all the Classes that make up the domain. Note:

You can navigate the domain hierarchy using the 'Navigation' pane on the right. Testpoints are composed as expressions, using the variable names of the Class members. The Intelli-sense shortcut Ctrl+Space is available within the editor to help you find these. Expressions that evaluate to True are taken to mean a pass. Returning False is taken to mean a fail.



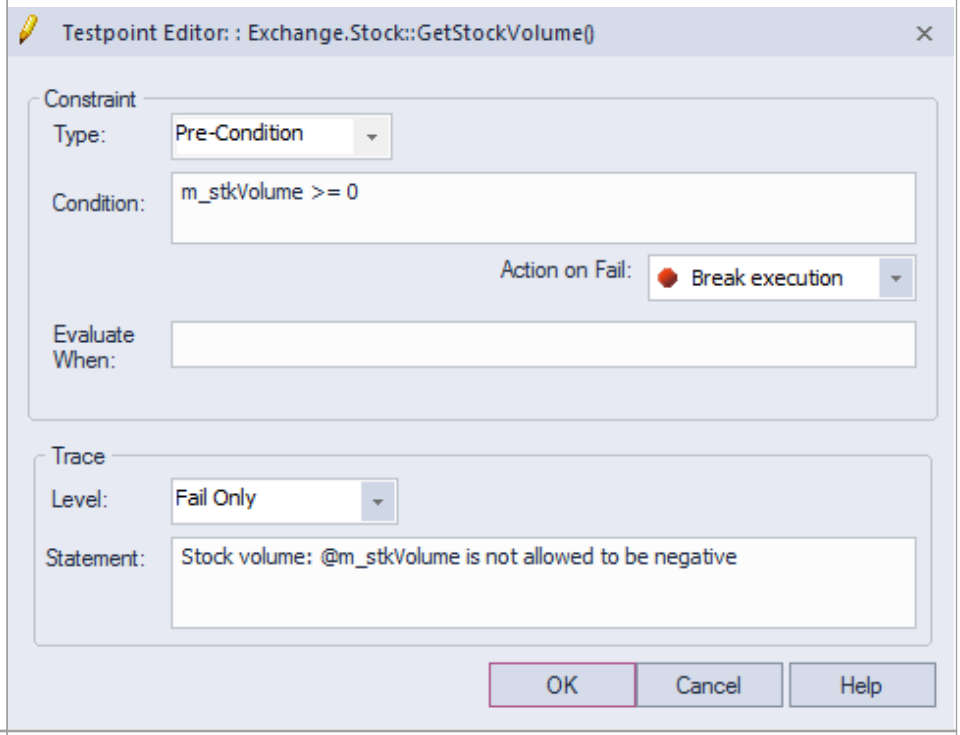
You can add or edit an existing Invariant by double-clicking the Class.

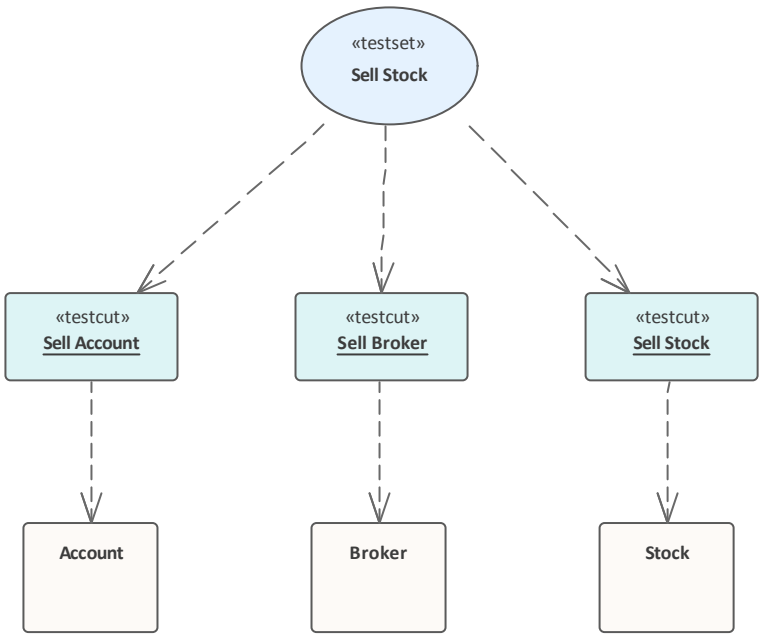
You can add or edit an existing pre- or post-condition similarly by double-clicking the method.

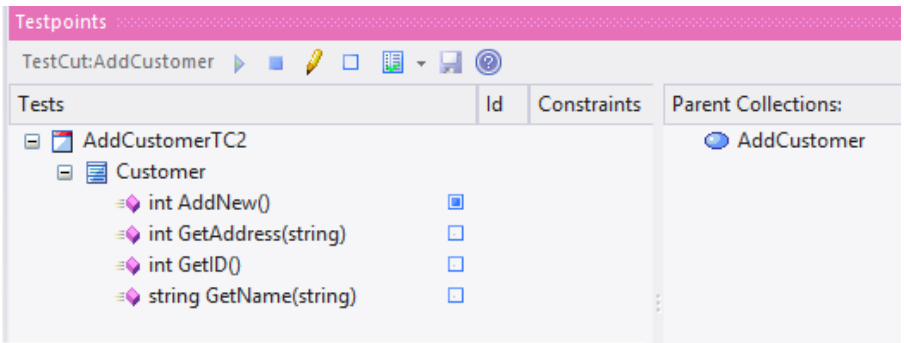
Double-clicking a Testpoint will automatically display the source code if it is available.

Line conditions are best added from within the code editor using its shortcut menus.

This image is of a pre-condition in the

	Test domain. 
Testpoint Trace Statements	Each Testpoint can have its own Trace statement. The Trace statement is a dynamic message that can reference variables in its object or local scope. They are output during the evaluation of a test. They can be configured to be output every time a constraint is evaluated, or more usually when a test has failed. Trace statements can be directed to the 'Testpoints' tab of the System Output Window, or to an external file. You can configure this in any Analyzer Script.
Test Domain Composition	The Test Domain diagram is a dynamic medium where Testpoints are assembled to test Use Cases. Use Cases in a Test

	Domain diagram are provided in three different stereotypes: Test Cut, Test Set and Test Suite. Management of the domain is as easy as modeling on any diagram. The toolbox and shortcut menus provide access to any Test Domain Artifacts. In brief, Testpoints from multiple Classes are aggregated into Test Sets. Test Sets are then linked to form Test Suites. Both Test Cuts and Test Sets are re-usable assets. Linking the same Test Set to the one or more Test Suites is a matter of drawing connectors. 
Test Domain and the Class Model	Rarely would a Use Case involve all the methods of a single Class. Most likely it is realized using a variety of methods from collaborating Classes. We call this

	subset of methods a 'cut', and the Test Cut Artifact is the tool we use to make these cuts. The Testpoints window will adapt depending on the context, to be that for a Test Domain or Class element. This image shows the Testpoints window when a Test Cut has been selected. Note the checkboxes, which are only visible for a Test Cut. They denote the methods (Test Cuts) that are contributing to a Test Set. In this example the Test domain was generated by the Execution Analyzer, which did the method identification work for us. 
Testpoint Evaluation	The Testpoints window is used to evaluate Test domains. The window has a toolbar for starting or attaching to the target application. The domain to test is always reflected by the element that has context, so if you select a Class the window will show only the Class structure and Testpoints of that Class. If you select a Test Suite, the window will

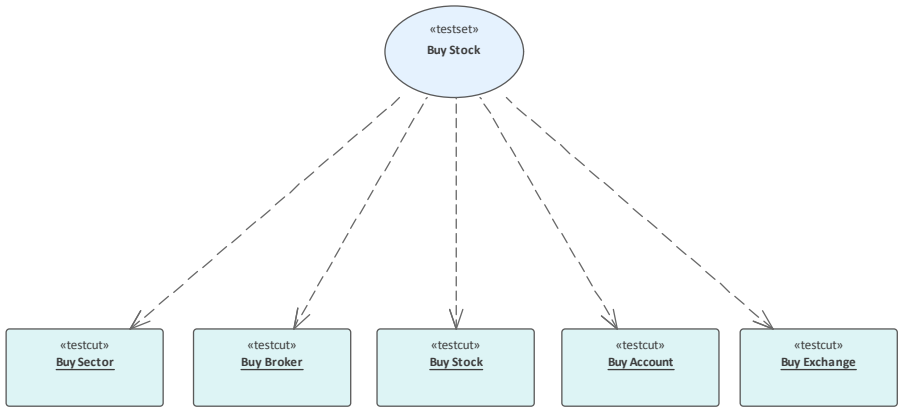
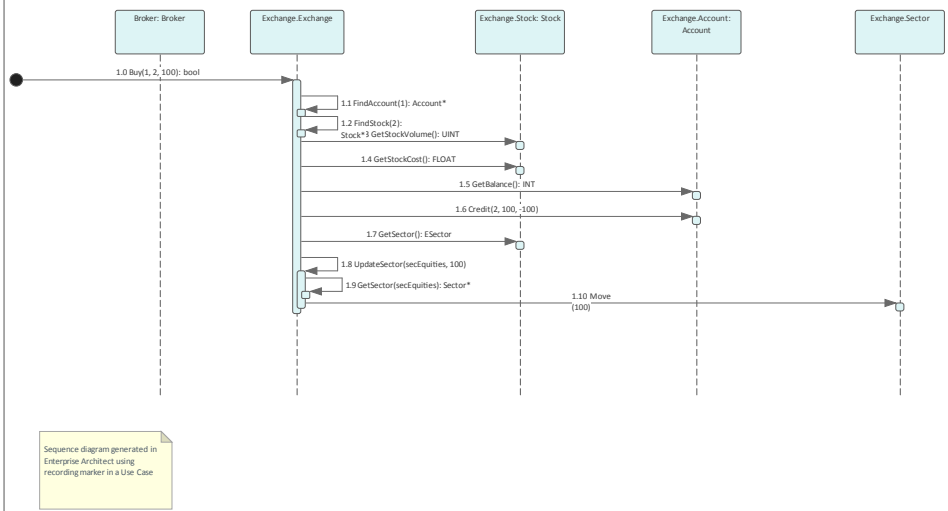
	display the entire domain hierarchy and all the Testpoints included in it. Clicking on the Run button will load the Testpoint domain in the Execution Analyzer, which will then evaluate, collect and update the report window as Use Cases pass or fail each test. The exact details of each constraint type and the when and how of that constraint's capture are: • A Class Invariant is evaluated by the Analyzer whenever any method called on an object of this Class type is completed; the invariant serves to test that the state of a complying object is both known and permitted • Pre-conditions are evaluated immediately before an operation is called • Post-conditions are evaluated (at the same time as a Class invariant) when the method is completed • Line-conditions are evaluated if and when their specific line of code comes into scope during program execution

Test Domain Diagram

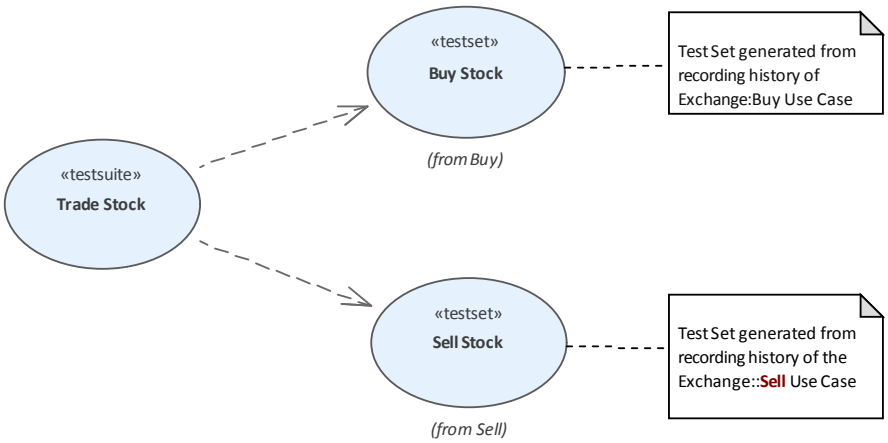
The Test Domain diagram is the medium where you assemble and group test cases for a particular domain. An example of a Test domain might be 'Customer'. The breadth and depth of the domains you assemble is up to you. You might have separate domains for 'Add Customer' and 'Delete Customer', depending entirely on how you consider best to balance the domain hierarchy. The Diagram Toolbox and Shortcut menu provide a number of Artifacts to help model the domain. Because the medium is dynamic, allowing you to revisit and build on relationships between Test domains, the system is a great model for delivering reusable assets to an organization that are low overhead and integrate with both the UML view of the world, and the Software Engineering nuts and bolts of daily life.

Facilities

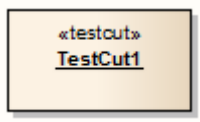
Facility	Details
Test Domain Generation	If you think the process of composing a Test Domain is complex, it can be, but help is at hand! The Execution Analyzer can produce a Test Domain diagram for you. It cannot write the Tests for you, but it can do some of the leg work. It can identify the Classes and pick out only

	those methods that participated in a Use Case. And this is not guesswork. The Analyzer Test Domain is obtained from a running program. This image shows the Test Domain generated by the Execution Analyzer from recording an Example Model program.  And this is the recording itself (as a Sequence diagram) from which the Test Domain was generated. 
Test Domain Composition	The first task on a Test Domain diagram is to create the Use Cases (Test Sets). These define this particular domain's

responsibility. The Diagram Toolbox and shortcut menu provide Artifacts to help you achieve this. The first of these elements is the Test Cut, which is used in the next step; identifying those methods (from the Class model) that you consider to be participants in the Use Case. The Test Cut Artifact is useful because it allows us to partition a Class, selecting only those methods that are relevant. Test Cuts can be run individually or linked to one or more Test Sets. Test Sets in turn can be linked to one or more Test Suites. In any case, any element of the Test Domain tree can be run individually or as a whole.



Test Cut



Description

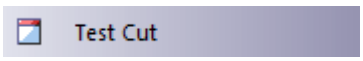
A Test Cut element is a stereotyped Object element, used internal to Enterprise Architect for defining Test Sets using the Testpoint code testing facilities.

A task, such as 'Print', might involve operations on different Classes. In order to create a 'Print' test, you would want to include only the 'Print' operations of these Classes and exclude any other operations.

A Test Cut enables you to capture only the operations that represent the behavior (in this case, 'Print') defined for a single Class. You might then place the Test Cut from each of several Classes into a single task as a Test Set.

When you drag a Test Cut element onto a Test Domain diagram, you create a Dependency relationship with the required Class element. As a result, when you select the Test Cut element on the Testpoints window, the operations of the Class are listed in the window, each with a checkbox. You then select the checkbox against each Class operation to include in the Test Cut.

Toolbox icon



Test Set

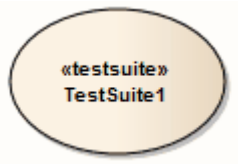
Description

A Test Set element is a stereotyped Use Case element used to aggregate one or more groups of methods (Test Cuts), which perhaps span multiple Classes, into a single task. Test Sets can also be aggregated into Test Suites.

You link the Test Cut elements to the Test Set using Dependency connectors.

Toolbox icon

Test Suite

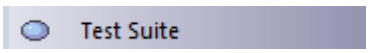


Description

A Test Suite element is a stereotyped Use Case element, used to aggregate one or more groups of tasks (Test Sets).

You link the Test Set elements to the Test Suite using Dependency connectors.

Toolbox icon



The Testpoints Window

The Testpoints Window is the hub where Test Domain constraints are composed. It is also the control that lets you verify a particular Test Domain on a program. The program might be already running or it can be launched using the control's Toolbar. Here you will also be able to see the results of your tests, as they happen. This control is context-sensitive, responding to the selection of elements in the Browser window or on a diagram. Depending on the selection, tests can be carried out on a single Class, a Use Case (Test Set) or a collection of Use Cases (A Test Suite).

Access

Ribbon	Execute > Tools > Tester > Show Testpoint Window

Testpoints Window Columns

Column	Usage
Tests	Displays the name of the selected

	Testpoint object and the hierarchy of objects beneath it. The selected object can be a: • Class • Operation • Test Cut • Test Set or • Test Suite
Id	For an Operation, this column shows a Testpoint marker icon () when the Analyzer has successfully bound this operation in the target application. If no icon appears in this column during a run, it indicates that the model and code base might not be synchronized; perhaps the signature of the function has changed, or the operation is a new method you are working on that exists in the source code but not yet in your model. For a Testpoint, this column shows a generated id number. This id number is used in trace output to indicate which constraint is being referenced.
Constraints	A pencil icon () in this column indicates that one or more constraints are defined for this Class or Operation.

Status	During a test run, indicates these possible statuses: • (×) Failed - Constraint has evaluated as false one or more times. • (!) Invalid Statement - Constraint failed to parse due to invalid syntax. • (?) Variable not found - A referenced variable name was not found at the location where the constraint was evaluated. No icon is shown if a constraint has Passed.
Evals	During a test run, indicates the number of times the Execution Analyzer has evaluated this constraint.
Passes	During a test run, indicates the number of times the test passed.
Fails	During a test run, indicates the number of times the test failed.
Last Run By	Displays the username of the last person to run this test. (Values are derived from the Project Author definitions in the 'People' dialog - 'Configure > Reference Data > Model Types > People > Project

	Authors'.)
Last Run Date	Displays the date and time this test was last evaluated.
Last Run Result	Displays the result of the last test run.
Parent Collections Pane	Lists any parent collections that include the selected object as part of their design. Double-click this collection to make it the selected object in the left pane. The Parent Collections pane can be hidden by clicking the Show / Hide Parent Collections pane button on the Testpoints Window Toolbar.

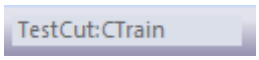
Testpoints Toolbar

The Testpoints Window Toolbar provides options to execute configured tests on the currently selected Testpoint object, stop a test run currently in progress, filter the displayed items, and save the results of a completed test run.

Access

Ribbon	Execute > Tools > Tester > Show Testpoint Window

Testpoints toolbar options

Toolbar Button	Action
	Field showing the name of the currently selected Testpoint object.
	Execute the test run.
	Stop the test run currently in progress.

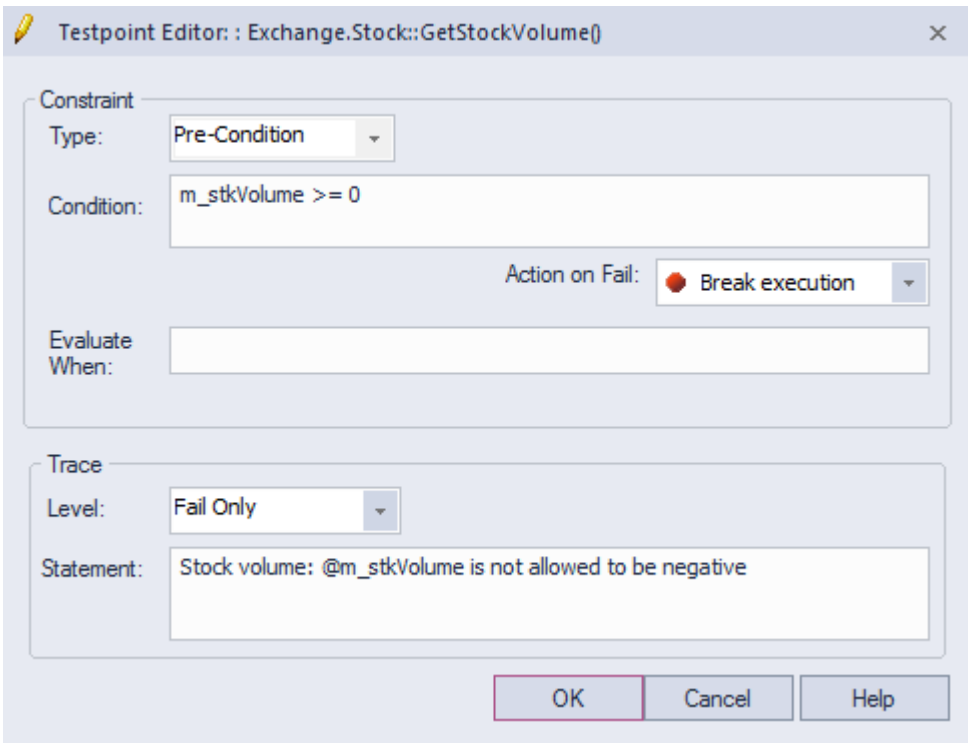
	Toggle between showing all items and showing only those items that have constraints defined.
	Toggle between showing all items and showing only operations that have been marked for inclusion in this Test Cut; this button is only enabled when a Test Cut object is selected. When a Test Cut is selected, each of the operations of its associated Class are displayed with a checkbox; you use this checkbox to mark the operations that apply to this Test Cut.
	Click on the drop arrow next to this icon to display the 'Test Run Options' menu, providing these options: • 'Prefix Trace output With Function Call' - Prefix all trace output lines with the executing function name • 'Enable Standard Breakpoints during Testing' - When not checked, the test run ignores any breakpoints in the current breakpoint set, and any attempts to set breakpoints during the run are ignored • 'View Trace output' - Display the

	'Testpoints' tab of the System Output window
	Click on this icon after completion of a test run to save the results to Test item on the current object. Saved tests can be viewed using the Testing Workspace. A prompt displays to select the Test Class - Unit, Integration, System, Inspection, Acceptance or Scenario. Select the appropriate Test Class and click on the OK button.
	Display the Testpoint Management Help topic.
	Show or hide the Parent Collections pane.

Testpoint Editor

The Testpoint Editor is used to compose constraints for Classes and Operations. The types of constraint permitted are dependent on the selected object. For Classes, the type will always be Invariant. For operations, the type can be either Pre-Condition, Post-Condition or Line-Condition.

Invariants are evaluated by the Analyzer when any method called on an object of the selected Class type completes. Pre-conditions are evaluated at the beginning of each call to the specified operation. Post-conditions are evaluated upon completion of each call to the specified operation. Line-conditions are evaluated each time the specified line of code is executed.



Access

Ribbon	1. Execute > Tools > Tester > Show Testpoint Window. 2. In the Testpoints window, double-click on a Class or Operation to display the 'Testpoint Editor' dialog.

Constraint Group fields

Field	Usage
Type	The type of constraint for the selected Class or Operation: • Invariant - Evaluated after any method called on the specified Class has completed • Pre-Condition - Evaluated at the beginning of each call to a specific Operation • Post-Condition - Evaluated after completion of each call to a specific Operation • Line-Condition - Evaluated upon execution of a specific line of code

	within an Operation
Offset	For Line-Conditions only, the Line number within the specified operation upon which to evaluate the constraint. An offset value is automatically set if the Testpoint was created using the Code Editor context menu.
Condition	The constraint to be evaluated when this Testpoint is triggered. A status of pass or fail will be recorded depending upon whether this constraint condition evaluates as true or false.
Action on Fail	Click on the drop-down arrow and select from the three options: • 'Continue' - ignore failure of this constraint and continue execution • 'Break execution' - halt execution and display the Stack trace • 'Disable on fail' - do not execute the constraint again after failing once
Evaluate When	(Optional) An additional constraint which must be met before the main Testpoint Condition is evaluated, providing greater control over test coverage.

Trace Group fields

Option	Action
Level	Specifies when the trace statement (if defined) will be output. Available options are: • 'Fail Only' - Output trace statement only when this Testpoint condition fails • 'Always' - Output trace statement every time this Testpoint is evaluated
Statement	(Optional) A message to be output when this Testpoint is evaluated. Variables currently in scope can be included in a trace statement output by prefixing the variable name with a $ token for string variables, or a @ token for primitive types such as int or long. Output from a Trace Statement can be directed either to the 'Testpoints' tab of the System Output Window, or to an external file, as configured by the Analyzer Script for the parent Package.

Testpoint Constraints

A Constraint is typically composed using local and member variables in expressions, separated by operators to define one or more specific criteria that must be met. A constraint must evaluate as true to be considered as Passed. If a constraint evaluates as false, it is considered as Failed.

Any variables referenced within the constraint must be in scope at the position where the Testpoint or Breakpoint is evaluated.

General/Arithmetic Operators

Operator	Description
+	Add Example: a + b > 0
-	Subtract Example: a - b > 0
/	Divide Example: a / b == 2
*	Multiply Example: a * b == c

%	Modulus Example: a % 2 == 1
()	Parentheses - Used to define precedence in complex expressions. Example: ((a / b) * c) <= 100
[ ]	Square Brackets - Used for accessing Arrays. Example: Names[0].Surname == "Smith"
.	Dot operator - Used to access member variables of a Class. Example: Station.Name == "Flinders"
->	Alternative notation for the Dot operator. Example: Station->Name == "Flinders"

Comparison Operators

Operator	Description
=	Equal To Example: a = b

==	Equal To Example: a == b
!=	Not Equal To Example: a != b
<>	Not Equal To Example: a <> b
>	Greater Than Example: a > b
>=	Greater Than or Equal To Example: a >= b
<	Less Than Example: a < b
<=	Less Than or Equal To Example: a <= b

Logical Operators

Operator	Description
AND	Logical AND Example: (a >= 1) AND (a <= 10)
OR	Logical OR Example: (a == 1) OR (b == 1)

Bitwise Operators

Operator	Description
&	Bitwise AND Example: (1 & 1) = 1 (1 & 0) = 0
|	Bitwise OR Example: (1 | 1) = 1 (1 | 0) = 1
^	Bitwise XOR (exclusive OR) Example: (1 ^ 1) = 0 (1 ^ 0) = 1

Additional Examples

Example	Description
((m_nValue & 0xFFFF0000) == 0)	Use a Bitwise AND operator (&) with a hexadecimal value as the right operand to test that no bits are set in high order bytes of the variable.
((m_nValue & 0x0000FFFF) == 0)	Use a Bitwise AND operator (&) with a hexadecimal value as the right operand to test that no bits are set in low order bytes of the variable.
m_value[0][1] = 2	Accessing a multi-dimensional array
a AND (b OR c)	Combining AND and OR operators, using parentheses to ensure precedence. In this example, variable 'a' must be true, and either 'b' or 'c' must be true.

Notes

- String comparisons are case-sensitive