

## Space Details

<b>Key:</b>	BOSS
<b>Name:</b>	BOSS
<b>Description:</b>	
<b>Creator (Creation Date):</b>	frode (Nov 30, 2006)
<b>Last Modifier (Mod. Date):</b>	frode (Nov 30, 2006)

## Available Pages

- CubicTest
  - Continuous Integration - Maven 2 Plugin
  - CubicTest and Agile Software Development
  - CubicTest - Articles and Blogs
  - CubicTest - FAQ
  - CubicTest - HTML Prototype Exporter Plugin
  - CubicTest - License
  - CubicTest - Recorder Plugin
  - CubicTest - Roadmap
  - CubicTest - Selenium Exporter Plugin
  - CubicTest - Test Modeling Tips
  - CubicTest - Troubleshooting
  - CubicTest - Tutorial
  - CubicTest - User manual
  - CubicTest - Videos
  - CubicTest - Watir Exporter Plugin
  - Custom Test Steps
  - Custom Test Suites (and flow control)
  - Essential Concepts in CubicTest
  - Internationalization of tests
  - Parameterisation of test data
  - The Graphical Test Editor
  - Writing your own CubicTest-exporter

## CubicTest

---

This page last changed on Mar 06, 2009 by [schwarz](#).

**Welcome to the CubicTest Wiki!**

See [www.cubictest.org](http://www.cubictest.org) for an introduction to CubicTest, features and downloads.

### CubicTest Documentation

[Tutorial](#)

[Essential concepts in CubicTest](#)

[The Graphical Test Editor](#)

[Custom Test Steps](#)

[Custom Test Suites \(and flow control\)](#)

[Continuous Integration - Maven 2 Plugin](#)

[Parameterisation of test data](#)

[Internationalization of tests](#)

[Test Modeling Tips](#)

[CubicTest and Agile Software Development](#)

[FAQ](#)

[Troubleshooting](#)

[License](#)

### CubicTest Plugins

[Selenium Exporter / Runner](#)

[Recorder](#)

[HTML Prototype Exporter](#)

[Watir Exporter](#)

### Miscellaneous

[Writing your own CubicTest-exporter](#)

[Discussion Forum](#) (for questions, comments, etc.)

[Issue Tracker](#)

[Articles and Blogs](#)

## Continuous Integration - Maven 2 Plugin

---

This page last changed on Feb 23, 2009 by [schwarz](#).

### Run tests with Maven

To run tests from the command line:

1. Create at least one [Custom Test Suite](#) (JUnit Test) that specifies which tests to run. This file must be in the `src/test/java` folder.
2. Type **mvn test** and the command line at the project root, and all JUnit tests classes defined in the project (Custom Test Suites) will be run.

Please note: The `JAVA_HOME` environment variable must point to minimum a Java 5 JRE.

The maven runner is independent of Eclipse, and does not require that CubicTest is installed.

### Usage tips

To **pause the test runner** during a run from the command line, press the "pause" key on your keyboard with the command prompt active. Press any key to continue the test runner.

To get and keep test reports, use a Continuous Integration Tool like Hudson to capture the Maven Surefire plugin (JUnit runner) test reports. Then you can use the CI tool's features for statistics on test run results.

## CubicTest and Agile Software Development

---

This page last changed on Nov 29, 2008 by [schwarz](#).

## CubicTest and Agile Software Development

CubicTest lets the "customer" (non-technical / business-side person) as well as developers write automatable tests of a web application.

### Tests written in CubicTest can be used

1. to report bugs in the form of a failing test
2. for regression test existing features during the life cycle of a project
3. as input to the developer when a feature is about to be implemented
4. for specifying changes to existing functionality

Our experience is that web components are best extensively tested at the Controller / Action level with a unit testing framework (and not with CubicTest), to make them faster to run and avoid starting a web server.

### CubicTest is in our experience best used for

- System sanity checks (check all main functions of a web applicatins), i.e. testing web flow and web configuration
- Testing JavaScript (although there are seperate testing frameworks for that too)
- Testing existing/legacy web applications without unit tests at the Controller / Action level

One should try to write the tests as focused as possible, not asserting more than strictly necessary. Manual testing is much better for finding bugs in content.

## CubicTest - Articles and Blogs

---

This page last changed on Mar 06, 2009 by schwarz.

### **Installing CubicTest**

<http://feelingblack.blogspot.com/2009/02/installing-cubictest.html>

### **CubicTest: Creating a custom Test Step to add the missing SeleniumIDE actions**

[http://www.dzone.com/links/rss/cubictest\\_creating\\_a\\_custom\\_test\\_step\\_to\\_add\\_the.html](http://www.dzone.com/links/rss/cubictest_creating_a_custom_test_step_to_add_the.html)

### **CubicTest, with pictures**

<http://www.daveliebreich.com/blog/?p=173>

## CubicTest - FAQ

---

This page last changed on Mar 02, 2009 by [schwarz](#).

### User forum

The most questions and answers are found in the CubicTest user forum:

<http://clearspace.openqa.org/community/selenium/cubictest>

The rest of this page is the official CubicTest FAQ.

### General

#### Can I test any kind of web applications with CubicTest?

- Yes, that should be possible as long as it is based on HTML and not e.g. Flash or Java Applets.

#### What is the relation between CubicTest and Selenium?

- Some of the CubicTest plugins use Selenium to run tests and for recording tests. CubicTest does not depend on Selenium in other ways than this.

#### Can I run a CubicTest test from the command line or from Maven?

- Yes, CubicTest tests can be launched from JUnit, so any tool like Maven can be used to run the tests. Maven is the easiest option, due to automatic classpath setup and JAR file download.

#### What license does CubicTest come with?

- CubicTest is licensed under the [Eclipse Public License v1.0](#) (free, open source, no warranty).

#### How are tests stored on disk?

- Tests are stored in XML, directly mapped from the CubicTest domain model to XML via [XStream](#). Tests can at any time be exported to Selenium Core tables (a popular test format) or Watir test cases.

#### Can I version control tests?

- Yes, CubicTest uses Eclipse as foundation, and all Eclipse team / version control plugins can be used. [Subclipse](#) is bundled with the rich client version of CubicTest.

### Graphical test editor

#### Why is it not possible to create "cycles" in the tests (i.e. a transition back to a previous page/state)?

- This is not possible because it would complicate the exporting and running of the tests, as well as probably making the test harder to understand.

#### How to specify a keypress of a special key?

- You must use char codes. See the following example char code listing:

[http://www.geekpedia.com/KB53\\_A-list-of-keys-and-the-JavaScript-char-codes-they-correspond-to.html](http://www.geekpedia.com/KB53_A-list-of-keys-and-the-JavaScript-char-codes-they-correspond-to.html)

Use a **backslash** in front of the char code, e.g. like this for pressing the "enter" key:

Action type: Key press, text input: \13

If the char code is used in a Custom Test Step, use two backslashes (the Java way).

### Running tests

#### I need to use HTTP Basic Authentication and want to enter username/password automatically

- Firefox supports URL as follows: <http://yourUsername:yourPassword@yourHost/> (e.g. <http://john:pw@localhost/>). Use this as your initial URL start point.

## CubicTest - HTML Prototype Exporter Plugin

---

This page last changed on Jul 30, 2007 by [schwarz](#).

### About

This exporter generates a HTML prototype of the modelled web application. It is written by Erlend S. Halvorsen and is maintained by the CubicTest team. It is mainly a proof of concept exporter for fast HTML prototype creation.

### Features

- All content and site navigation from the tests will be present in the prototype.
- The user interactions modeled in the tests will be available in the prototype as well, e.g. as clickable links and buttons.

### System requirements

- None (Eclipse has an internal browser to view the HTML prototype that is created by this plugin).

### How to use

- Create the tests as normally (preferrably avoiding tree tests, sub tests and extensive use of extension start points).
- Right click in test editor -> choose "Create HTML prototype"  
This generates the HTML pages (placed in the "generated" directory).
- Open the "index" file from the "generated" directory in the package explorer (in folder with test name as name).

### Recommended use

- Use when detailing a user story, when writing tests together with the customer for getting an instant sample of how the modeled application will work.

### Limitations

- Not very usable when test uses extension points extensively.

## CubicTest - License

---

This page last changed on Jan 30, 2008 by [schwarz](#).

CubicTest is open source software, licensed under the terms of the **Eclipse Public License - v 1.0**, which can be found at <http://www.eclipse.org/legal/epl-v10.html>



## CubicTest - Recorder Plugin

---

This page last changed on Mar 01, 2009 by [schwarz](#).

### About

The CubicTest "Recorder" plugin provides the possibility to record from a real browser directly into the CubicTest graphical test editor.  
It is based on Selenium RC and is actively supported and maintained by the CubicTest team.

### Features

- Supports both URL start points and Extension start points (in the latter case, the test browser is "forwarded" by using the Selenium RC Runner plugin to run the test to the start point and start recording from there).
- Starts up a Firefox browser with a custom profile (no setup necessary except for requiring Firefox to be installed).
- Injects a context menu in the HTML of the page to record from.
- Populates the graphical test editor in Eclipse directly.
- Records both page element assertions and user interactions.
- Captures as many identifiers as possible on the recorded elements.

### System requirements

- Requires that Firefox is installed.

### How to use

- To start recording: Right click in the CubicTest graphical test editor -> choose "Record" (or right click on a Page/State and select "Record from selected Page/State")  
This starts up Firefox and forwards the browser to the selected point to start recording.
- To assert elements present: Right click on elements of the page to record from (in Firefox)
- To record user interactions: Do the user interactions as normal (fill in a form, click buttons or links etc.).
- To stop recording: Same as above (toggle the "Record" menu item)

The active page that elements will be put into (the "cursor" of the recorder) is highlighted as "selected" in the CubicTest graphical test editor.

### Recommended use

- Use for writing tests for an existing web application.

### Limitations

- Has limited support for contexts. Workaround: Record first, then edit the test, creating contexts manually, and drag and drop page elements into their belonging contexts. Another possibility is to not use contexts (e.g. when the recorded identifiers suffice).

### If Firefox is installed in a location other than the default one...

You have two options:

- 1) Set the PATH variable of the operating system to include the folder where the firefox binary is located.
- 2) Set the "firefoxDefaultPath" Java System Property. This can be set in eclipse.ini, located in the same folder as eclipse binary.

Example eclipse.ini:

```
-vmargs  
-DfirefoxDefaultPath=c:/path/to/firefox.exe
```

If tests are run from maven, set the Java System property behind the "mvn test" command.

## CubicTest - Roadmap

---

This page last changed on Aug 27, 2007 by [schwarz](#).

### Introduction

This page shows the planned releases of CubicTest with the main new features to add.  
A reminder of the **CubicTest version numbers**: Releases are going to be: 1(for now).lastDigitOfYear.month

### CubicTest 1.7.8 release

- Just bug fixes from 1.7.7 release.
- Intended for release on openqa.org. Should ship with much documentation

### CubicTest 1.7.9 release

- Introduce "**Custom steps**" for exporter-agnostic custom steps (custom code) for test assertions/ user interactions that is not possible to write using the graphical test editor. E.g. assert stuff present based on business rules, or make user interactions based on conditions. The different test exporters must be adapted to this, as the "custom step" support will be extensible and exporter neutral.
- "**Record from here**" context menu of a page/state that forwards the test browser to that state (Recorder plugin).

### CubicTest 1.7.10 release

- Refactoring support: "**Extract subtest**".
- Context menu of extension points: "**Find tests starting from this extension point**".
- Show warning when deleting extension points that have tests that start from it.

## CubicTest - Selenium Exporter Plugin

---

This page last changed on Mar 01, 2009 by [schwarz](#).

### About

The CubicTest Selenium Exporter Plugin runs tests directly in the CubicTest graphical test editor using an embedded version of Selenium RC. No setup is necessary, and the target server does not have to be instrumented. Using this plugin is the preferred way to run tests in CubicTest.

The plugin can also export to Selenium Core HTML tables for running tests with an external Selenium installation.

The plugin is written and maintained by the CubicTest team. It is also has a [Maven 2 plugin](#) based on it.

### Features

- Supports the entire CubicTest test model (all identifiers, actions and contexts).
- Gives you direct feedback by coloring the test model.

### System requirements

- Either IE, Firefox, Opera, Google Chrome or Safari must be installed.

### How to use

#### Running the tests directly in the GUI

1. Create/open a test
2. Right click in test editor -> choose "Run As" -> "Run CubicTest with Selenium".  
This opens a browser and runs the tests. The results are shown in the test model. Green = pass, red = fail, orange = exception
3. Click "Reset test-run" to remove the coloring of the test.

#### Exporting to Selenium Core files

Alternatively, the tests can be exported to separate Selenium Core HTML files for running in an existing Selenium installation:

1. Create/open a test
2. Right click in test editor (or on the file in the package explorer) -> choose "Export to Selenium Core test script"  
The generated files are placed in the "generated" subfolder of the test project (found in the package explorer).

### Limitations

- Options in SelectLists do not support more than one identifier, and no modifiers (e.g. "begins with") when Options are used for user interactions (though it is supported for asserting presence).

### Running tests through a proxy server

If your browser needs to use a proxy server to reach external sites, configure CubicTest as follows.

Set the following vmargs in the Launch Configuration of the test runner:

```
-Dhttp.proxyHost=my.proxy.com  
-Dhttp.proxyPort=8080  
-Dhttp.proxyUser=my_username  
-Dhttp.proxyPassword=my_password
```

(not all need to be set)

If tests are run from maven, set the properties behind the "mvn test" command.

## **If Firefox is installed in a location other than the default one...**

You have two options:

- 1) Set the PATH variable of the operating system to include the folder where the firefox binary is located.
- 2) Set the following Java System Property in the Launch Configuration of the test runner:

-DfirefoxDefaultPath=c:/path/to/firefox.exe

If tests are run from maven, set the Java System property behind the "mvn test" command.

## CubicTest - Test Modeling Tips

---

This page last changed on Feb 23, 2009 by [schwarz](#).

### **Use Extension Points for your front page, logged in state and start of all main functions**

Tests typically need common setup such that they start from a well known condition (e.g. logged in and on front page). Initial tests that for example logs in the user and creates an extension point for the logged in state should be created such that each normal test of an application function can start from such an extension point. Starting from such a "logged in extension point", another test laying out extension points for all main functions of the application should be made, such that tests related to "Function A" can start from a "Function A extension point".

### **Use subtests as building blocks**

For tests involving many pages/states and transitions one should consider using subtests as building blocks to make the test more readable and maintainable. Such sub test building blocks can then be used in variations of the same test and in other tests. Sub tests can have either an Extension start point (uses common setup) or a sub test start point using no test setup.

### **Use Contexts extensively**

Contexts are a good and robust way to identify elements. Use them. Good examples of contexts are tables, rows and sections of the page (HTML "div" elements).

### **Improve and refactor the recorded tests**

The CubicTest Recorder is a useful tool for recording tests for an existing web application. Recorded tests should be improved with e.g. contexts grouping together elements, use of Commons, decomposition into subtests (use copy and paste), and better Label / "Show in editor" identifier values for better readability. CubicTest has refactoring support for moving subtests to other directories (paths are updated automatically), and support for extracting a subtest of selected pages/states (select the nodes, and right click -> Refactor -> Extract subtest).

### **Avoid extensive use of tree-tests**

Tree tests (tests with more than one path) can complicate tests if used for long paths. Tree tests is best used for small variations in a test, and not for writing multiple tests in the same test file.

## CubicTest - Troubleshooting

---

This page last changed on Feb 23, 2009 by [schwarz](#).

### User forum

The most questions and answers are found in the CubicTest user forum:

<http://clearspace.seleniumhq.org/community/selenium/cubictest>

Use the forum for questions. The rest of this page is the official CubicTest Troubleshooting.

### All versions

#### Problem with locating elements on web page

##### **Solution: Use Selenium Runner debug log / XPath Checker**

If you are having problem with locating an element when running a test (no element or wrong element found), remember to check the identifiers of the element. All identifiers with "must" moderator must be satisfied. If more than one element matches the identifier, the first one is picked by the runner. Consider using contexts if you do not have enough identifiers (if you can find a unique parent element).

Also consider using the "Contains", "Starts with" or "Ends with" moderators, as some identifiers (e.g. labels) may have HTML formatting around them, and then these moderators will solve the problem by not requiring the the whole string matches.

If you still are having problems, check the Selenium Runner has a debug log. **Press the "Show log" button** in the Selenium frameset in the browser (e.g. in Firefox)

Here the XPath expressions used for locating elements can be seen. If you are having trouble with an XPath expression locating an element, copy the XPath from the log and paste it into e.g. the **XPath Checker Firefox extension** (must be downloaded seperately as a Firefox extension). When installed, right click in the web page and select "**View XPath**") and paste the XPath there. The XPath Checker shows you the elements that match your XPath expression. Adjust your XPath expression until you see only the element you wish to locate.

PS: The XPath Checker extension will not be loaded with the Selenium Runner in CubicTest, so you will have to use a seperate browser window.

### Update-site version

#### **The CubicTest perspective will not open / show up in list of perspectives**

Make sure you have [minimum requirements](#) installed.

If you have multiple JREs (Java Runtime Environments) on your computer, you may have to tell Eclipse to use the Java 5 JRE:

- Create a shortcut to your eclipse.exe file,
- In the shortcut: Add a -vm parameter pointing to the javaw.exe in the Java 5 JRE directory that Eclipse should use.
- E.g.: "C:\Program Files\eclipse-SDK-3.2-win32\eclipse.exe" **-vm** "C:\Program Files\Java\jdk1.5.0\_08\jre\bin\javaw.exe"
- Start Eclipse with this shortcut.

If this does not help, you might want to try the rich client version of CubicTest.

## Rich Client version

### Problem with sharing a project with subclipse

Subclipse is bundled in the CubicTest rich client version, but sharing a project might show an error message. Try configuring the project with Tortoise SVN first, then share the project with subclipse in CubicTest.

## Maven version

### Error message "Unsupported major.minor version 49.0"

This happens when the command prompt is not configured with Java 5 or higher as JAVA\_HOME

**Solution:** Set JAVA\_HOME to a Java 5 or higher JRE. If you use the rich client version, invoke the following command in the command prompt: set JAVA\_HOME=<cubictest install dir>\jre. Closing the command prompt window sets JAVA\_HOME back to the original value.



## CubicTest - Tutorial

This page last changed on Feb 24, 2009 by [schwarz](#).

### Download CubicTest

First, go to the [download page](#) and download CubicTest. We recommend the update site version for Eclipse 3.4 Ganymede users, and the "rich client" version for users that do not have minimum Eclipse 3.4 Ganymede or Java 5 installed.

If the rich client version is used, unzip the zip file to your hard drive.

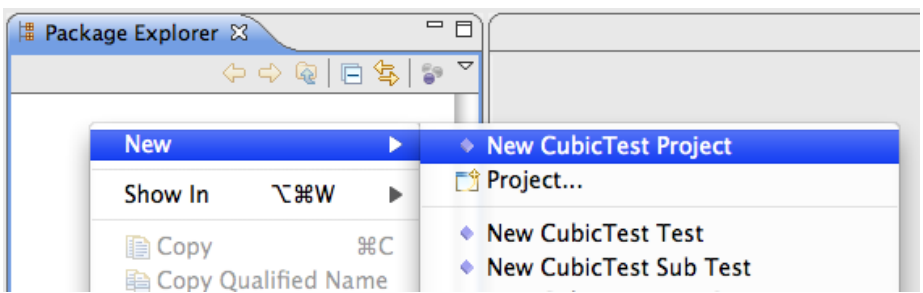
### Start CubicTest

**Update site version:** Start Eclipse and **open the CubicTest perspective** (Window -> Open Perspective -> Other... -> CubicTest)

**Rich client:** Double click on the "**eclipse.exe**" file in the folder where you unzipped CubicTest to start CubicTest.

### Writing your first CubicTest test

#### Create a New CubicTest Project

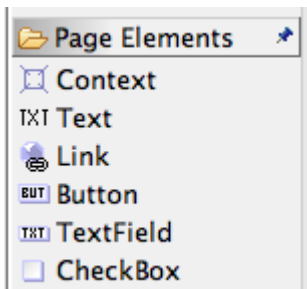
-  Right-click in the "package explorer" view (top left frame) in Eclipse, and select:
  - New -> **New CubicTest Project**
  - Follow the instructions on screen.

An empty test will appear in the Editor window with an initial, empty first page/state.

#### Short introduction to the CubicTest test model

- **Tests** in CubicTest consists of a series of pages/states and transitions between the page/states.
- Each **page/state** can have several **page elements** in them, which means that they should be asserted present on the page. Examples of page elements include text, links, text fields, buttons etc. These page elements can also be the subject of user interactions.
- **Transitions** take the "test user" from one page/state to the next. A transition can consist of several **user interactions** on page elements from the source page. Together these user interactions form the transition.  
Examples of user interactions include entering text into a text field, clicking a link and "mouse over" an image. When a transition has multiple user interactions (e.g. filling out a form) only the last one (e.g. "click Submit") is assumed to trigger the transition to the next page/state.

For more info about the domain model, see the wiki item [Essential Concepts in CubicTest](#).



## Create assertions

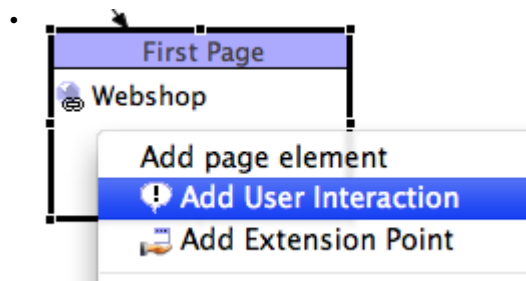
### for the initial page/state

Add some page elements (e.g. links) to the first page that appears in the new test.

Page elements to assert is found in the **palette** on the left of the graphical test editor.

## Create a user interaction

After some page elements are added to the page, create a User Interaction transition that leads from one page to another:



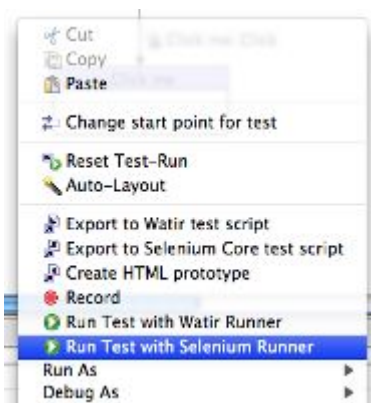
\*Right click on the page/

state\* -> choose **"Add user interaction"**

- Select the page element to apply user action to (in the first column, e.g. a link).
- Then select the action type (second column, e.g. "Click").
- For page elements that accept text input, the input goes into the third column.
- You can create several user inputs in the same transition by clicking the **"Add new user input"-button**.
- Press OK to add the transition

A new target Page/State is automatically created. Here you can add page elements (e.g. Text) to assert the presence of some result of the user interaction.

## Run the test



browser,

To run this test in a web

- Right click in the test editor -> **Run As --> "Run CubicTest with Selenium"**.  
This will open up a browser and run the tests. The test model in the CubicTest editor will be colored with the results.
- Alternatively, right click in the test editor -> choose "Run Test with Watir Runner"

This will run the tests in Internet Explorer using Watir. Ruby and Watir must be installed, see <http://wtr.rubyforge.org/install.html>

## Recording tests

Alternatively to creating a test manually (as described above), tests can be recorded by interacting with an existing web application.

The recorder requires that Firefox or Opera is installed.

To record a test,

1. **Right click** in the background of a Test and choose **"Record"** (or right click on an arbitrary Page/State and choose "Record from this Page/State")  
Firefox/Opera will open and load the start point URL.
2. To record page element assertions, **right click on the page in Firefox** on "page elements" (e.g. a link) and choose e.g. "Assert link present".  
To record text-present-assertions, first highlight/select the text and then right click and choose "assert text present".
3. To record user interactions, **interact with the page as normal**. The interactions will be recorded, and "click" events will trigger the transition to a new state. Text input will not automatically trigger a transition to a new state. Observe that user interactions are added to the Cubic test model in the background.

To stop recording,

1. Right click in the graphical test editor -> toggle "Record" off.

## Custom Test Steps

**A Custom Test Step** is a way to write test code for test steps that you **cannot model** in CubicTest with the Graphical Test Editor and elements from the palette.

Custom Test Steps let you **write any code you want**, and to use any libraries you want on the classpath. The code can be run and debugged using standard Eclipse mechanisms.

Initially, CubicTest Custom Test Steps only support Selenium RC but the other frameworks (HTML export, Selenium export and Watir) will follow.

### To create a Custom Step and add it to a test:

1. Right click on a folder in the package explorer view then "New -> **New CubicTest Custom Test Step**".
2. Complete the wizard that opens, and the custom step editor will open.
3. Drag and drop the created .custom file into a test's Graphical Test Editor, and the custom step will be added to the test.

All installed CubicTest exporters (e.g. Selenium, Watir, HTML Prototype) can provide an implementation of the Custom Test Step, such that the test and custom step can work with different frameworks.

### To create a Selenium RC implementation of the custom test step:

1. Open the custom test step (either open the .custom file, or double click on a Custom Step in the Graphical Test Editor).
2. Press the **Selenium RC link** to create a custom test step class using Selenium RC. Provide a suitable name.
3. After the class is created, click on the Selenium RC link again for a shortcut to open the class.
4. Implement the class using Selenium RC. The Selenium RC Javadoc is attached. Check out the ICustomTestStep interface for javadoc for CustomTestStep itself.

To launch a test with a custom test step in it, you need to right click on the CubicTest and select **"Run As -> Run CubicTest with Selenium"**. This starts up with firefox as the default browser. To change the browser please edit the launch configuration by opening the "run dialog" found in the toolbar.

## Creating a test suite and running tests from the command line

When a new CubicTest Project is created, a default JUnit test named "CustomTestSuite.java" is created. Here you can launch tests using a SeleniumRunner object provided by CubicTest. The default implementation runs all tests within the "tests" folder.

The feature is called Custom Test Suite since you can use all the Java code and all libraries you want, e.g. for custom setup and teardown of tests.

The SeleniumRunner object can be configured the same way as the test runner launched from the GUI. See its JavaDoc and setter methods. It also has options for whether to keep the test browser open between tests or not.

To run the Custom Test Suite, right click on the class and select **Run As --> JUnit Test**.

To run it from the command line:

- Ensure that [Maven 2](#) is installed and has minimum Java 5 as its JRE.
- Change directory to the CubicTest test project folder and type **mvn test**
- This will open up the test browser and run the test suite.

For users that want to model which files to run (and not add them with code), there is a test suite editor for specifying which tests to run. Right click on the "test suites" folder and select "New CubicTest Test Suite".

To add tests, make sure the .ats suite file is open and drag and drop test files it into the editor. The advantage of this approach is that renaming/moving of tests automatically is updated in the suite.

This .ats suite file can be launched from the GUI or from a Custom Test Suite.

## Viewing samples

The Rich Client version of CubicTest comes with sample tests. To view them in CubicTest, press File -> Import... -> Existing Projects into Workspace -> Select root directory: <CubicTest install dir>\samples\CubicShopTest -> Press OK

This page last changed on Jun 20, 2007 by [schwarz](#).

## Creating a test project

Before you can create any tests, you first have to create a CubicTest project.

To create such a project in Eclipse:

1. Open the CubicTest perspective by choosing: Window -> Open perspective -> Other... -> CubicTest
2. Press File -> New -> CubicTest Project
3. Complete the wizard that opens.

## Creating a test

A new test is created automatically when you create a new CubicTest project. To create more tests, right click on the "tests" folder in your CubicTest Project. (You might want to see own chapter on how to create a CubicTest project).

## Test contents

### Start points

### URL Start Point

### Extension Start Point

### Page/State

### Page Elements

### Page Element Identifiers

### User Interactions

### Connections

### Contexts

### Commons

### Extension points

### SubTests

## Extending other tests

## Test orchestration

## Tree tests

## Recording a test

## Running a test

## Exporting a test

## CubicTest - Videos

---

This page last changed on Jun 18, 2008 by [schwarz](#).

We do not have any videos that are up to date at this time.

## CubicTest - Watir Exporter Plugin

---

This page last changed on Feb 03, 2008 by [schwarz](#).

### About

The Watir Exporter Plugin runs tests in the Graphical Test Editor using Ruby and Watir. It also exports tests to external Ruby/Watir code.

Watir provides a fast and robust way to run the tests.

Before the runner will work, Ruby and Watir must be installed. See <http://wtr.rubyforge.org/install.html> for installation instructions.

The plugin is written and maintained by the CubicTest team.

### Features

- Supports the entire CubicTest test model (all identifiers, actions and contexts), though with a small limitation on Options i Select lists.
- Gives you direct feedback by coloring the test model.

### System requirements

- This plugin requires that the following is installed (see <http://wtr.rubyforge.org/install.html>)
  - Ruby
  - Watir (version 1.5 or above)

### How to use

#### Running the tests directly in the GUI

1. Create/open a tests
2. Right click in test editor -> choose "Run Test with Watir"  
This opens Internet Explorer and runs the tests. The results are shown in the test model. Green = pass, red = fail, orange = exception
3. Click "Reset test-run" to remove the coloring of the test.

#### Exporting to Watir files without running them

Alternatively, the tests can be exported to separate Ruby/Watir files for running in an existing runner/build environment:

1. Create the tests as normally.
2. Right click in test editor (or on the file in the package explorer) -> choose "Export to Watir tests script"  
The generated files are placed in the "generated" subfolder of the test project (found in the package explorer).

### Limitations

- Options cannot have more than one identifier
- If contexts or multiple identifiers are in use, the HTML page must be syntactically valid and will be parsed by Watir and REXML. This will take a bit longer than when only one identifier is used.

## Custom Test Steps

This page last changed on Feb 07, 2009 by [schwarz](#).

## Custom Test Steps

**Custom Test Steps** provide you the possibility to write **custom code** to be executed during a test run. E.g. for doing assertions or interactions that are not possible to model with the Graphical Test Editor in CubicTest.

A Custom Test Step in CubicTest consists of two things:

1. A generic Custom Test Step definition (element in the Graphical Test Editor) which defines the **name** and **description** of the custom test step and which can define a set of **input parameters** to the step.
2. **Implementation(s)** of the step (chunks of code), max one implementation per exporter plugin.

The **Custom Test Step** feature is fully pluggable, i.e. you can in theory write an implementation of a Custom Test Step in all available CubicTest exporter plugins -- if the plugin provides an editor for it.

**Example:** A Custom Step for typing today's date in an input field can take the input field ID as parameter and populate the field. To make sure that the test runs with both Selenium and Watir, an implementation can be created for both.

As of version 1.8.11 only the Selenium Remote Control plugin for CubicTest has a Custom Test Step Editor.

### To create a Custom Test Step:

1. **Right Click on a folder** in the Package Explorer and select "**New CubicTest Custom Test Step**" (you might want to create a separate folder for all your custom test steps)
2. Provide a **name** and file name for the custom test step. Press Finish.
3. In the Custom Test Step definition page that opens, you can **create implementations** of the step and define parameters. The blue **links** are for creating implementations.

### To add a Custom Test Step to a Test:

1. **Drag and drop** the Custom Test Step from the Package Explorer into the Graphical Test Editor
2. **Connect** the step with a Page/State so that it can be reached.

## The Selenium RC Java Custom Test Step Editor

This is an advanced implementation, with the following features:

1. Add and use all the Java libraries you want
2. Autocomplete and inline javadoc for the Selenium RC API
3. Full Java debug support

To create an implementation:

1. Click on the **Cubic Test Selenium Extension** link in the Custom Test Step definition page
2. Provide a name for the Java Class to be created
3. Write Java code in the class that opens. See the Selenium RC documentation for details of the API.

You will be writing an implementation of the `ICustomTestStep` class.

This class has an **"execute"** method which you should implement. We expect you to use JUnit type assertions like `assertEquals`, etc. In this way CubicTest can handle the CustomTestStep assertion failures correctly. You can throw other kinds of exceptions as well.

### Parameters to the execute method:

- **"arguments"** (key-value pairs) Key names are from Custom Step definition page and values from properties page in the Graphical Test Editor.
- **"context"** Shared Custom Step Context. Makes it possible to send messages from one custom step to another instead of using static variables. Is basically an empty `HashMap` until you decide to fill it up with something.



- **"selenium"** The Selenium Remote Control object.

The method can throw any type of exception or error. Handled by CubicTest. `java.lang.AssertionError` = step failed, others = test exception.

**Example:** Custom Test Step for **selecting a browser window** (e.g. a popup) based on a `windowName` parameter:

```
public class SelectWindow implements ICustomTestStep {  
  
    public void execute(Map<String, String> arguments, IElementContext context,  
        Selenium selenium) throws Exception {  
        selenium.selectWindow(arguments.get("windowName"));  
    }  
  
}
```

## Important notes on browser timing

When entering the Custom Test Step, the Web Browser may not have loaded the page fully.

There are two strategies for handling this, depending on the type of application/page being tested (traditional or Ajax/JavaScript).

### Custom Test Step after page reload (traditional web app style)

When testing traditional web applications, you should add `selenium.waitForPageToLoad(..)` to ensure that all page elements are ready before asserting presence and interacting with the page.

Example: To wait max 10 seconds for page load and then click on an element:

```
selenium.waitForPageToLoad("10000");  
selenium.click(yourLocator);
```

### Custom Test Step responding to Ajax or JavaScript

When testing Ajax/JavaScript pages, you cannot use `selenium.waitForPageToLoad(..)` since there is no page reload to wait for.

Instead, use the Wait class of Selenium RC to assert element presence with a timeout. Then the elements will be present and ready for interaction when the Wait returns.

Example: To wait max 10 seconds for page element and then click on it:

```
new Wait() {  
    public boolean until() {  
        return selenium.isElementPresent(yourLocator);  
    }  
}.wait("Page element not found: " + yourLocator, 10000);  
selenium.click(yourLocator);
```

## Custom Test Suites (and flow control)

---

This page last changed on Feb 22, 2009 by [schwarz](#).

### Custom Test Suites

**Custom Test Suites** is a feature in CubicTest that lets you launch CubicTest tests from **Java and JUnit**. You can launch specific test or all tests in a directory and provide set up and tear down logic for the tests.

#### Usages:

1. Required for launching tests from the command line with maven 2 and for [continuous Integration](#).
2. Setup logic is required for initializing the system before running a test and tear down logic for cleaning up the system afterwards (or asserting system state after test).
3. Deciding which tests to run based on some flow control logic.

### How to create and run a Custom Test Suite

A sample Custom Test Suite is created when you create a new CubicTest Project.

You can create additional test suites, either by copying the sample class or creating a new JUnit test class. The test class must reside in the **src/test/java** folder of the CubicTest Project.

To run it, right click and select **Run As --> JUnit Test**

It can also be run from maven by typing **mvn test** from the command line.

## Essential Concepts in CubicTest

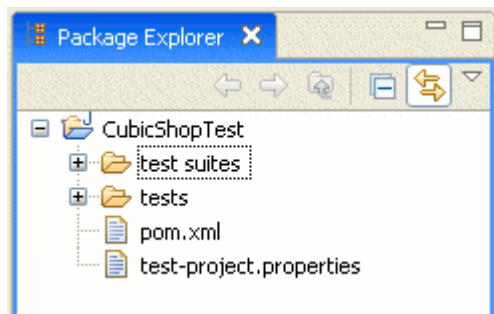
---

This page last changed on Nov 29, 2008 by [schwarz](#).

### Test Project

All tests in CubicTest must reside in a CubicTest test project. The project is created in the CubicTest perspective in Eclipse.

Example test project:



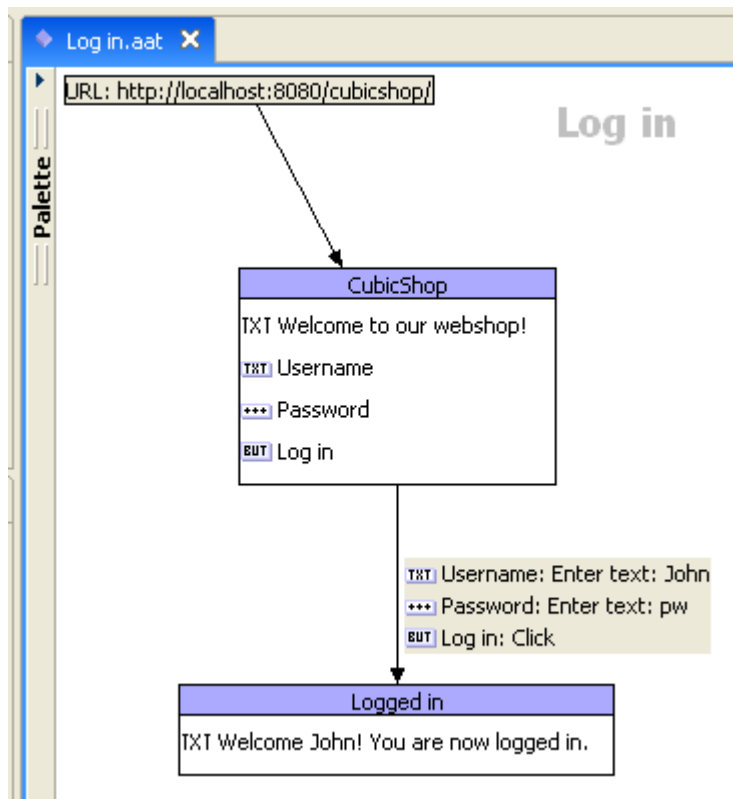
### Test

A test in CubicTest tests a single or multiple requirements of a web application. It is modeled as a series of pages/states with transitions (user interactions) between the pages/states.

A test can be used for:

1. Requirements testing
2. Requirements specification (test driven development)
3. Bug reports
4. Change request (test driven development)

Example test that tests the log-in functionality of a web shop:



The blue squares are pages/states and the arrow is a transition (user interaction) between the two pages/states.

The "Username", "Password" and "Log in" elements are form elements (asserted present and basis for user interaction, as explained below).

## Page/State

A web application typically has several pages/states. In CubicTest, a page/state is a stable view of the application that the user can interact with.

The user can advance to a next page/state by interacting with the application.

When JavaScript or Ajax is used, the state of an application can change frequently, such as during mouse position and text typing. The resulting changes is modeled as a new page/state.

Example page/state (asserts the presence of four "page elements"):



The title of the page/state (top part in blue) serves as a logical name to increase readability. The four elements within the page/state are called "page elements", more info on that below.

## Transitions and user interactions

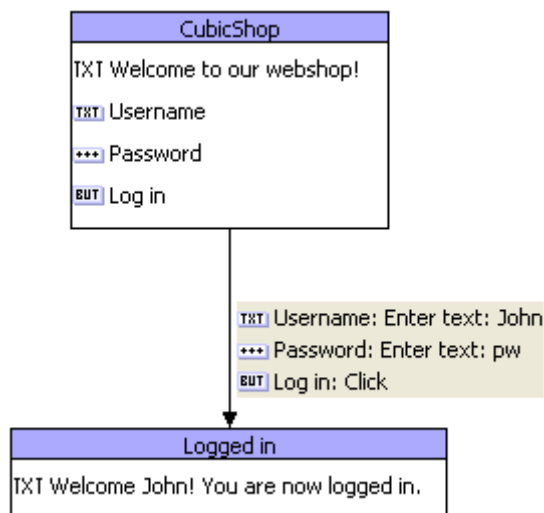
While a page/state represents a stable view of the application, transitions represents how this page/state is changed to get to the next page/state in the test.

A transition can contain one or more "user interactions", which are applied in sequence to trigger the next page/state.

Example user interactions that can be part of a transition:

- Fill out a text field
- Click a link
- Click a button
- Mouse over a page element
- Select an option in a select list

Example transition with three user interactions (fill in username, fill in password and click the "Log in" button. The latter of which should trigger the transition to the new state):



The arrow symbolizes the transition and contains the three user interactions. The transition is not considered applied until all user interactions are applied to the page/state.

### Transitions and page reloads

CubicTest tries to handle transitions and page reloads automatically. Some frameworks like Watir and Selenium Core makes this easy. When e.g. Selenium RC is used (the Selenium Runner), the user might in some cases need to specify some properties on transitions to control whether to wait for a page reload or not.

Default for the Selenium plugin is that after a "click" event in the last user interaction, it will wait for a page reload before continuing the test.

This might not be the desired behavior, so it is possible to specify whether the user interaction should trigger a wait for page reload. This option is in the properties view of the user interaction.

Examples where page reload should be set to false:

- JavaScript click events and page manipulation without a reload
- Opening of popup windows

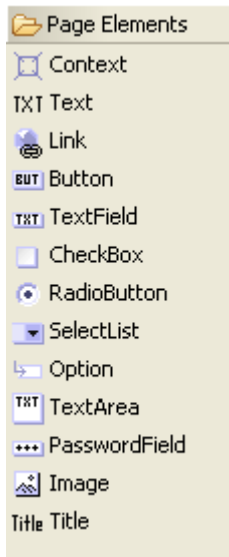
## Page Elements

These are used to assert the presence of things on the page/state, and as basis for user interactions.

Each page element has a set of identifiers that identify the element on the page (see own chapter below). Each element also has a set of user interactions that can be applied to it.

To test that an element is *not* present, there is possible to mark an element as "should not be present".

The following is a list of the available page elements in CubicTest (available in the "palette" of the test editor):



When a page element is added to a page/state, it looks like the following (asserting the presence of four elements):



## Contexts

Contexts make it easy to identify elements by specifying neighboring elements, and are used when elements are hard to identify uniquely by themselves.

Example:

There are many "Buy" buttons on a page, and it is hard to identify which one to press to buy a particular product.

We want to press the "Buy"-button that is in the same row as the text "Foo".

Contexts in CubicTest makes this possible. Contexts are similar to other page elements, but identify a part of the page (e.g. a table, DIV, table row etc).

Specify a context and put some elements in it that together forms a unique combination, and they will be identified on the page.

Contexts have a set of identifiers in the same way as page elements have, but they do not need to be unique as **the child elements of the context help identify the context itself**. But it is a good idea to provide some identifiers to the context as well (especially element name, e.g. "tr" for table row).

Contexts do not need to be the direct parent of its elements, it only has to be an ancestor. Thus detailed knowledge of the page structure is not needed.

To sum up, a Context has the following meanings:

1. It should be present on the page
2. It should contain all child elements that are specified to be in it

Good candidates for contexts are (any parent element is possible to use):

1. Rows

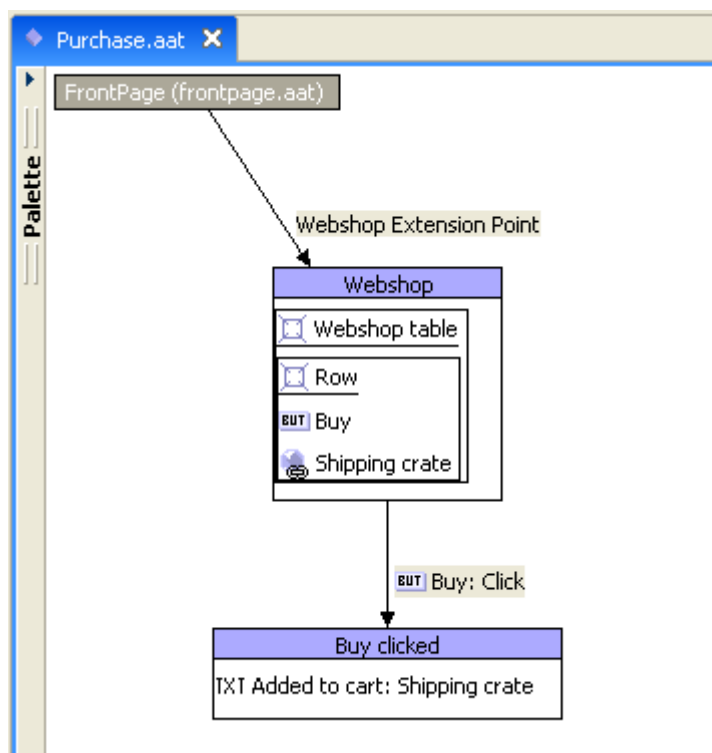
2. DIVs
3. Tables

**Example:** A table with a row in it, containing a link and a button. The hierarchical structure and sibling elements gives unique identification. All elements can have additional identifiers, but it may not be needed. Only "element type" - table and tr - are used in this case (not shown).



**Example shown in the graphical test editor:**

Context used to identify which "Buy" button to press (here the "Buy" button does not need any identifiers, as the "Shipping crate" link identifies which row is meant):



## Identifiers

A page element or context can be identified by a set of identifiers found in the HTML source.

Examples of identifiers (not all are applicable to all elements):

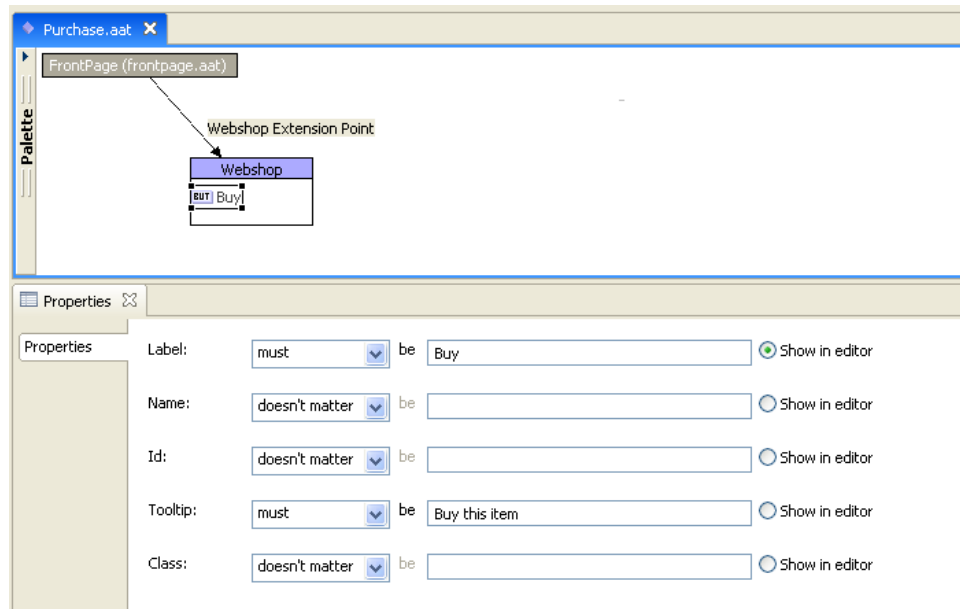
- ID
- Name
- Href
- Src
- CSS class
- Tooltip

CubicTest supports providing multiple identifiers at the same time, and a **logical "AND" is applied between** them, meaning that all must be satisfied.

An element can also be placed in a context to identify it (or it can be both in a context and have some identifiers itself).

In addition, each identifier can have one of the following moderators on its value: "be equal to", "contains", "starts with" or "ends with". This is handy when only part of the identifier value is known or possible to assert.

Example of a set of identifiers for the selected "Buy"-button (moderators not shown):



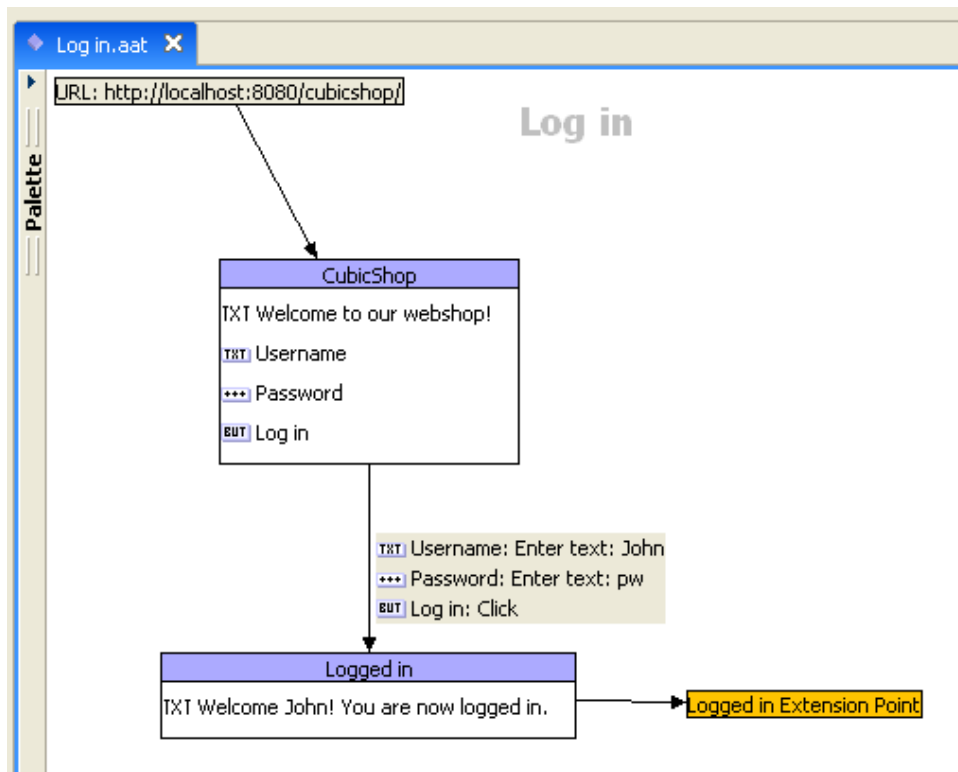
## Start Points and Extension Points

A test can start by either invoking a URL or by starting from an "Extension point" in another test.

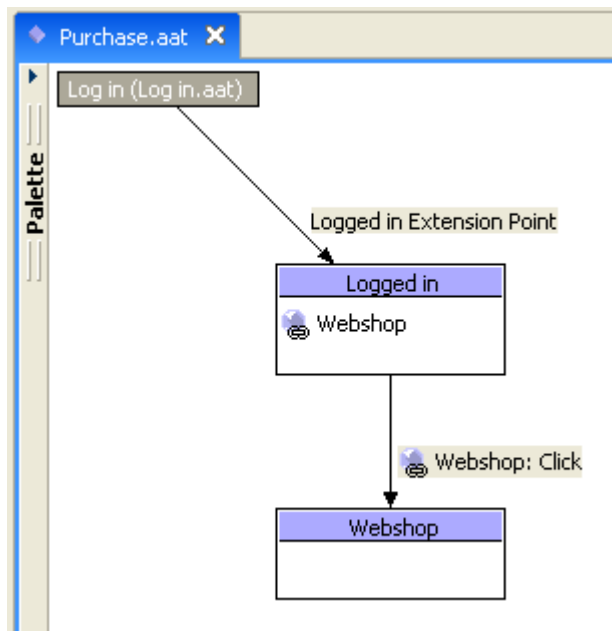
In general, it is possible to define an extension point at any page/state in a test, and other tests can continue (start from) this state. In this way, interactions and assertions common to more than one test do not have to be duplicated such that a bootstrapping does not have to be duplicated in all tests.

Example of a "URL start point" that starts from "http://localhost:8080/cubicshop/":





In this test we also define an "Logged in" extension point that other tests can start from.  
 Example of how this extension point can be used as the start point of another test:



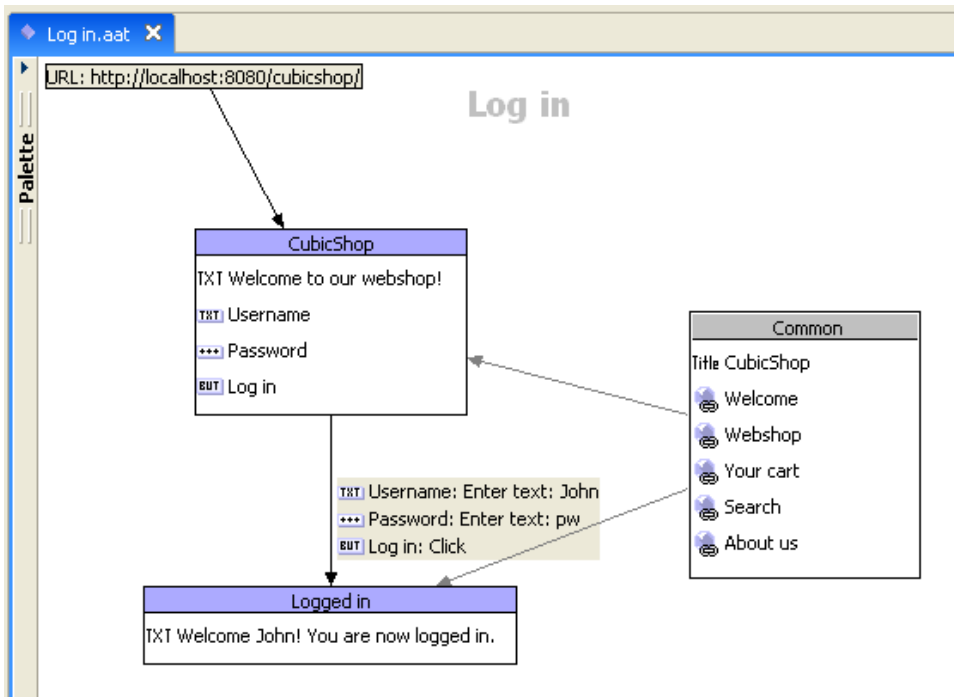
Here we start from the "Logged in" state defined earlier, and continue with a test that asserts and clicks a webshop link (rest of test omitted).

**To change the start point of an existing test,** right click on the start point and click "Change start point for test".

## Commons

A Common is a virtual page/state used for testing the same assertions on multiple pages.

Example using a Common for checking the presence of the main navigation links:



Commons are found in the Palette just as pages/states. To connect a Common to a page/state, drag and drop a Connection from the Palette from the Common to the page/state.

Elements from a Common can also be the subject of user interactions.

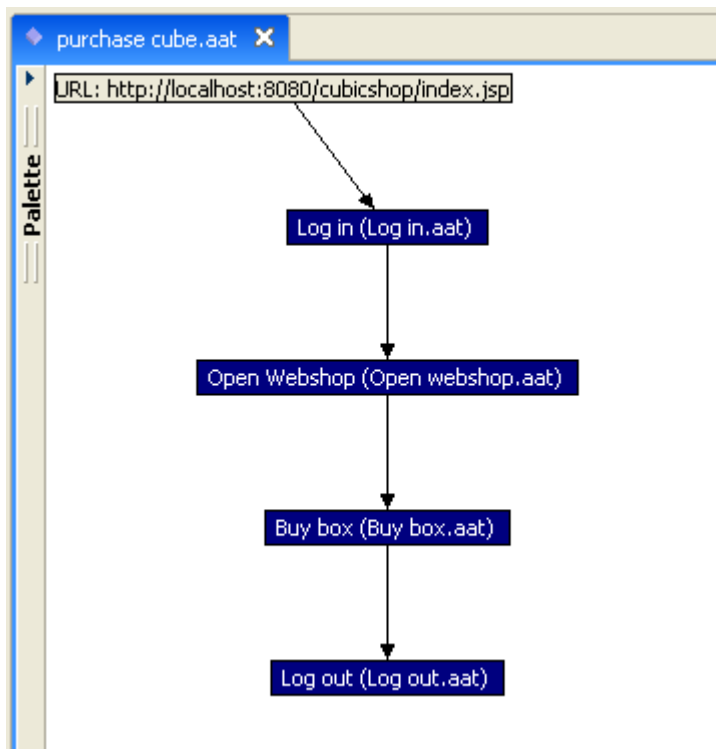
## Sub Tests

A test can be included in another test. This is using the test as a sub test.

Subtests are connected just as a page/state, and will be executed fully or to a defined extension point.

**To add a subtest to a test**, drag and drop a test from the "package explorer" into the Graphical Test Editor of another test.

Example test using four subtests:



### Dedicated sub tests using Sub Test Start Points:

Any test can be used as a sub test, but the most suitable sub tests are tests without their own setup-logic (i.e. without URL or Extension start points) such that they can be used in different settings and test scenarios.

To create a dedicated sub test,

1. Right click in the CubicTest package explorer,
2. New -> New CubicTest Test
3. Choose "Sub test start point" as start point type in step 2 of the wizard.

The new test will have a **Sub Test Start Point**, and cannot run stand alone (has no URL or Extension start point), it must be used as a sub test in a higher level test that contains a proper start point.

The start point can be changed to another type of start point by right clicking in the Test and choosing "Change start point of test".

## Test suites

There are two types of test suites. Custom Test Suites (JUnit java classes) and visual test suites (.ats files).

See page [Custom Test Suites \(and flow control\)](#). The rest of this section is a description of visual test suites:

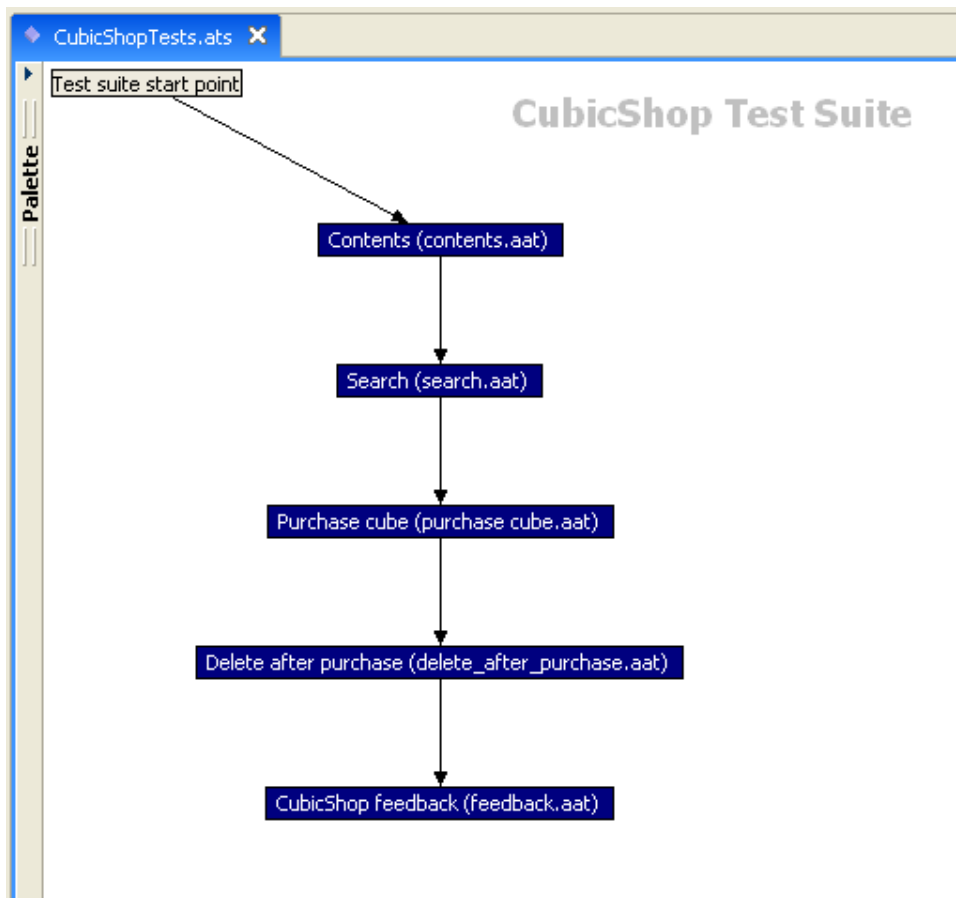
Visual Test suites are special tests that start with a "Test suite start point" and they typically reside in the provided "test suites" folder.

Visual Test suite files have the extension \*.ats. The main difference between a test suite and a normal test is the start point, which does not invoke any URL or previous test.

**To create a test suite**, right click on a folder (typically the "test suites" folder) and select "New -> New CubicTest test suite".

**To add tests to the test suite**, drag and drop tests from the Package Explorer into the Graphical Test Editor of the test suite and connect them with connections, starting from the start point. Test suites can be run / exported in the same way as normal tests.

Example test suite:



This page last changed on Nov 29, 2008 by [schwarz](#).

### Testing a web page in different languages

**Internationalization**, location or i18n is a way for you to be able to test the same page in different languages. The sole purpose of this feature is to prevent DRY (don't repeat yourself) tests for every different language you got.

#### How to add internationalization to a test

To add internationalization support in CubicTest:

1. Click on the background of a graphical CubicTest editor and select the internationalization tab in the properties view.
2. Click the "Add Language" button to add a language (a properties file) to the Test.
3. When the language is added, select a page element in the Test and enable internationalization. Set the element's key corresponding to the language .properties file.
4. To update the page element with the language value, set the language in the Test Internationalization properties page.

## Parameterisation of test data

---

This page last changed on Apr 08, 2008 by [schwarz](#).

### Parameterisation

Parameterisation enables a test to be reused for different test data sets.

Parameters can be controlled both on the **test level** and on the **sub test level**.

Both page element identifiers (e.g. label text, name and ID) and user input can be parameterised with data from a spreadsheet file (semicolon separated), or from the built in parameterisation editor.

Parameterization is controlled in the Graphical Test Editor, on the **Parameterisation property sheet** of a Test.

The data from a parameter row is inserted into the test, and stored there. This is controlled by the parameter index.

To use parameters:

1. **Create a parameter file** by right clicking on the "Create new parameter file" button on the parameterisation property sheet of a Test.
2. **Add data** to the parameter file (Click Add key / Add Row)
3. **Save** the parameter file and (on the parameterisation property sheet of the Test) press "**Refresh parameters**".
4. **Assign parameters to page element identifiers** in the property tab of page elements.  
To parameterise user input, choose "Enter parameter text" in the User Interactions dialog (in the dropdown of the **Action Type** column), and select the appropriate parameter name.  
This "Enter parameter text" option is only available on action types that accept text input.

If the test is used as a sub test, the parameter index can be controlled independently on the sub test level.

### Running a test with different data sets

Add the test as a sub test multiple times (one for each parameter index), and set the index on the subtest accordingly.

In the future, there will probably be an option in the runner to loop over all the indexes automatically.

### Comments

---

The user interactions dialog doesn't have a "enter parameter text" option. The only fields are Action Element, Action Type and Text Input. The possible values of Action type does not include "enter parameter text" either.

Posted by [clee](#) at Apr 07, 2008.

---

"Enter parameter text" in the User Interactions dialog is an option in the **Action Type** column. Select it and then select the appropriate parameter name.

This "Enter parameter text" option is only available on action types that accept text input.

Documentation updated.

Posted by [schwarz](#) at Apr 08, 2008.

---

I need to have a parameter with a DATETIME value which I can append to a string. Example helloworld2220041900@hotmail.com. Is there any functionality available other than reading from a table of parameters? My reason is that if I use an existing account I'm following a different path from creating a new account.

Posted by  at Apr 10, 2008.

---

I tried to edit the ruby script to be as follows:

```
t = Time.now  
puts "Could not set(\"test\" + t.strftime(\"%d%H%S%M\") + \"@hotmail.com\") [TextField: 'email']"
```

the script still executes as it did when it was recorded.

Posted by at Apr 10, 2008.

---

## The Graphical Test Editor

---

This page last changed on Apr 03, 2008 by [schwarz](#).

The Graphical Test Editor is where tests are modeled. It provides rich editing support for web tests.

The editor works like other windows applications with features like:

- Undo and redo
- Drag and drop
- Copy, cut, paste

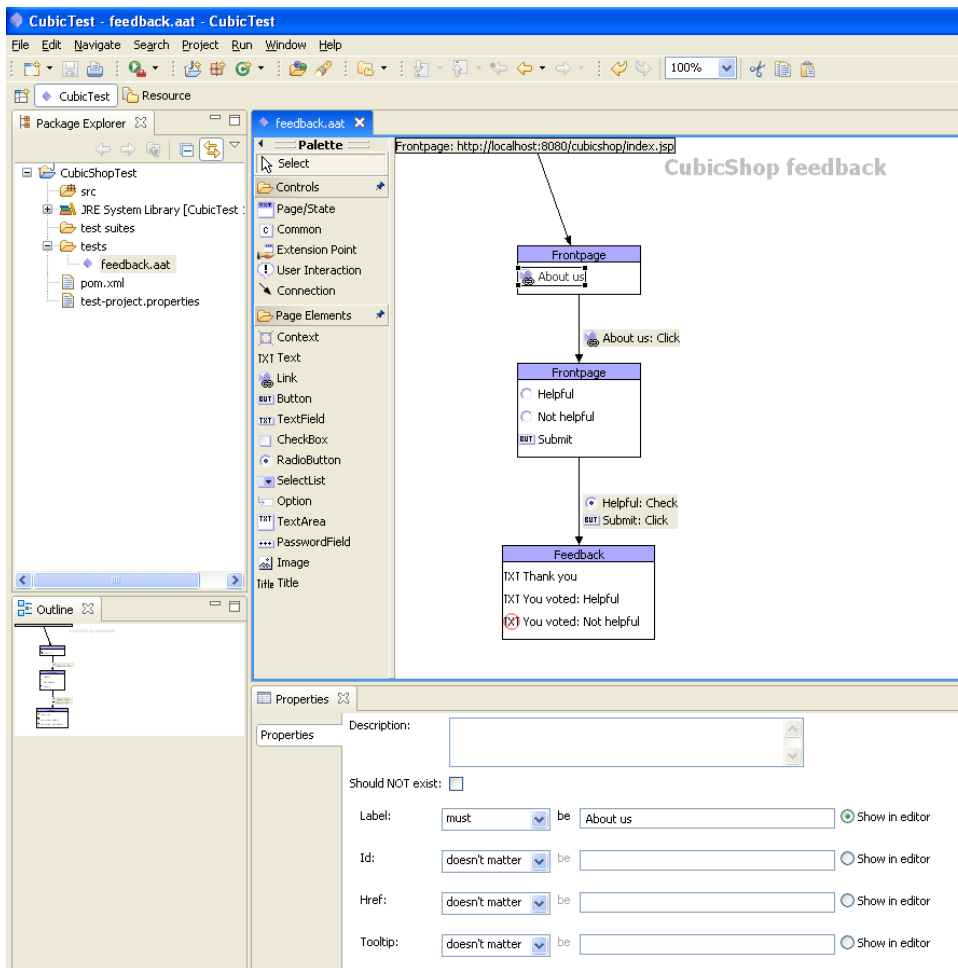
There is a **context sensitive menu** when the user **right clicks** on an element in the test editor.

Features that may not be obvious at first sight:

- Most elements has a **properties view** associated with it. Click on the element to see its properties. There may be more than one "tab".
- Click on the test background (canvas) to see **test properties**.
- Page elements can be **dragged and dropped** from one page/state to another (or to a Common). If the moved element participates in a user interaction, it will be deleted from the user interaction.
- To **change the start point of a test**, right click on it and select "Change start point"
- The **order of user interactions** in a transition can be edited by selecting e.g. "Move up" in the select list of the first column in the user interaction editor. Here actions can also be deleted.
- Copy, paste and undo works as expected. The undo support is extensive.
- **Sub tests are added** by dragging and dropping a .aat test file from the package explorer into the graphical test editor.
- Double click on sub tests to **open the sub test**.
- If sub tests are **moved or renamed**, the path is automatically updated in all tests that refer to the subtest.
- **Tree tests** are supported. To create a tree (i.e. more than one path), create multiple user interactions from a source page. It is recommended to keep tree tests to a minimum though, and instead create separate tests to improve readability and reusability.
- Eclipse's **Restore from Local History** (right click on file) is supported.
- User interaction elements can be moved up and down by selecting "Move up" or "Move down" in the dropdown of the User Interactions table editor.
- There is **refactoring** support:
  - When moving tests, the paths to subtests are automatically updated.
  - To extract a set of pages/states to a subtest, select them, right click and select Refactor -> "Extract subtest".

Example screenshot (the Graphical Test Editor is the tab with "CubicShop feedback" which contains the actual test):





## Writing your own CubicTest-exporter

---

This page last changed on Feb 25, 2008 by [schwarz](#).

### Introduction

CubicTest is created to be easily extensible with new exporters.

An exporter only has to provide some callback classes for converting e.g. a page element assertion to an appropriate string/action.

The exporters do not have to know e.g. how to traverse a CubicTest test (and can ignore Extension points, sub tests etc.)

Take a look at the provided exporters, create your own copy and modify it. Plugging it into CubicTest is as easy as just including the JAR file in the plugins/ folder of the CubicTest installation.

CubicTest uses the Eclipse "Play fair" rule for integration with the test editor. This means that the provided recorder and all runners and exporters are plug-ins and not tightly integrated with the rest of CubicTest. Additional runner/exporter or recorder plug-ins will have equal opportunity for integration as the provided plug-ins.

### SVN-repository

To learn from the existing plugins, take a look at the CubicTest source code located here: <http://svn.openqa.org/fisheye/browse/cubictest/trunk/>

### Comments

---

If you want to download/checkout cubic test using subversion then the URL is <http://svn.openqa.org/svn/cubictest>

Posted by at May 06, 2008.  
.....