**CA DLP DDS  
iConsole Automated Test Suite**

|  |  |
| --- | --- |
| Release Number | 1.0 |
| Component Name | iConsole Automated Test Suite |
| Author | Richard Dickinson |
| Revision | 1 |

**Responsibility List**

| Action | Responsibility |
| --- | --- |
| Owner | Richard Dickinson – Project Leader |
| Primary Peer Review | Julian Bowles – Automation Team Lead |
| Primary Peer Review | Sharon Robinson – Senior QA Software Engineer |
| Primary Peer Review | Robin Peeler - QA Manager |
| Secondary Peer Review | Mike Beckerleg - QA Software Engineer |

**Note:** The Responsibility List reflects those included in the peer review process.

**Change History:**

| **Revision Date** | **Last Revision By** | **Reason for Change** |
| --- | --- | --- |
| 27 May 2011 | Richard Dickinson | 1.0 – Initial Version |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[PREFACE 5](#_Toc295720433)

[1 INTroduction 6](#_Toc295720434)

[1.1 Scope 6](#_Toc295720435)

[1.2 Definitions, Acronyms and Abbreviations 6](#_Toc295720436)

[1.3 References 6](#_Toc295720437)

[2 Architecture 7](#_Toc295720438)

[2.1 Overview 7](#_Toc295720439)

[2.2 Sample Flow Execution Diagram 9](#_Toc295720440)

[3 Design Limitations, Assumptions, and Issues 10](#_Toc295720441)

[3.1.1 Issues 10](#_Toc295720442)

[4 Internals 11](#_Toc295720443)

[4.1 Programming details 11](#_Toc295720444)

[4.1.1 The Test Controller (TestRunner.java) 11](#_Toc295720445)

[4.1.2 Test Cases (WgnTestCase.java, WgnTestStep.java, ConfigurationTestStep.java, Test scripts) 11](#_Toc295720446)

[4.1.3 Configuration module (ConfigurationFile.java) 12](#_Toc295720447)

[4.1.4 Audit Configuration module (AuditConfigurationManager.java) 12](#_Toc295720448)

[4.1.5 Utilities (Utilities.java) 13](#_Toc295720449)

[4.1.6 Logging module (LogFile.java) 15](#_Toc295720450)

[4.1.7 Timer module (StopWatch.java) 16](#_Toc295720451)

[4.1.8 Mail verification module (ReadMail.java) 16](#_Toc295720452)

[4.1.9 Error Reporting 16](#_Toc295720453)

[4.1.10 Supporting Tools 17](#_Toc295720454)

[5 Internationalization 18](#_Toc295720455)

[5.1 Localization 18](#_Toc295720456)

[6 Configuration files 19](#_Toc295720457)

[6.1 System configuration 19](#_Toc295720458)

[6.2 Global Configuration file 20](#_Toc295720459)

[6.3 Test Configuration file 21](#_Toc295720460)

[7 Running the Test Suite 22](#_Toc295720461)

[8 Packaging and Installation 23](#_Toc295720462)

[8.1 Setup 23](#_Toc295720463)

[8.2 Installing the Test Suite 23](#_Toc295720464)

[9 Appendix - Selenium Javadoc 24](#_Toc295720465)

[9.1 Element Locators 24](#_Toc295720466)

[9.2 Element Filters 25](#_Toc295720467)

[9.3 String-match Patterns 26](#_Toc295720468)

[9.4 Java Documentation: Available Selenium Methods 27](#_Toc295720469)

[9.4.1 Startup, Shutdown and Configuration commands 27](#_Toc295720470)

[9.4.2 Screenshot commands 29](#_Toc295720471)

[9.4.3 Keyboard and Mouse commands 31](#_Toc295720472)

[9.4.4 Checkbox and Select controls 42](#_Toc295720484)

[9.4.5 Misc 44](#_Toc295720485)

[9.4.6 Window, Frame & Pop-up handling commands 49](#_Toc295720486)

[9.4.7 Get commands 57](#_Toc295720487)

[9.4.8 Is Queries 63](#_Toc295720488)

[9.4.9 Element Position and Sizing 65](#_Toc295720489)

[9.4.10 Waits and Timeouts 67](#_Toc295720490)

[9.4.11 Cookie handling 69](#_Toc295720491)

# PREFACE

The Detailed Design Specifications (DDS) describes external functional specifications as well as design specifications for one component or design entity of a project. There can be many DDSs for each project, one for each major feature described in the Product Requirements Document (PRD).

This is intended to be a living document. The product development cycle is a dynamic process in which the project and its criteria for success are refined over time. Therefore, it is expected that the completed DDS will undergo many revisions during the course of a project as requirements, resources, and constraints evolve.

The Development Manager is responsible for the contents of this document. Deliverables that must be completed prior to releasing this document are:

• Product Requirements Document (PRD)

• Top Level Design Specification (TLDS)

All template instructions can be identified by their gray italic type. This information may be removed after completing the necessary project information. If a section is not applicable for a project, identify the section as “N/A.”

# INTroduction

## Scope

Identify the requirement being addressed and the feature being designed.

The scope of this document is the design and usage of the Automated Test Suite for the iConsole using Selenium Server for execution.

## Definitions, Acronyms and Abbreviations

Provide the definitions of terms, acronyms and abbreviations required to interpret this document. For consistency, use the format in the following examples:

Selenium: A **suite of tools** to automate web app testing across many platforms. We are using Selenium Server 1.0.1 for test execution and the Selenium Java library for browser execution commands.

Detailed Design Specifications (DDS): A representation of a software system or component of a system created to facilitate analysis, planning, implementation, and decision-making. The DDS is used as the primary medium for communicating software design information.

Data Loss Prevention (DLP): The CA DLP product enables businesses to protect all types of sensitive information and assets from loss and misuse, remain compliant with regulations and corporate policies and reduce overall business risk.

iConsole: Web-based console which provides search, reporting and review capabilities for captured DLP data.

## References

[Provide a list of documents referenced elsewhere in this document.](http://seleniumhq.org/docs/index.html)

[Identify each document by title, report number (if applicable), date and publishing organization.](http://seleniumhq.org/docs/index.html)

[Specify the sources from which the references can be obtained.](http://seleniumhq.org/docs/index.html)

1. [Selenium website documentation](http://seleniumhq.org/docs/index.html)
2. [Selenium Java Documentation](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html)
3. [AutoIT: Automation and scripting language](http://www.autoitscript.com/site/autoit/)

Use this section to capture functional system requirements. Use case actors are used to define how the system interacts with end users and other systems to achieve the goal. Non technical language should be used so the section is easily understood by all team members.

Each key use case defined in the PRD/TLDS should be specified in detail here.

Additional use cases may be defined for each key use case as required to completely specify the external behavior of the system. Use cases may be structured hierarchically, with parent use cases including multiple detailed use cases as required to specify the external behavior of this component.

Complete the following table for each use case.

In addition to tables, diagrams may be used (e.g. UML Interaction Diagrams, Use Case diagrams, Sequence Diagrams, Activity Diagrams, or other) where these can help communicate the intended behavior of the product.

Refer to [CA Product Use Case Standards](https://km.ca.com/rnd/engineeringexcellence/Shared%20Documents/Product%20Management/Published/Use%20Case%20Best%20Practice.docx) for the definition of each of the use case fields and examples.

# Architecture

## Overview

Describe the architecture of the feature and where it fits in the overall design. Architectural design may be represented in many forms, including text, graphical description, pseudo-code representation, or combination. Where applicable cite areas of the Architecture Design Specification (ADS) or the Top Level Design Specification (TLDS) to reduce duplication.

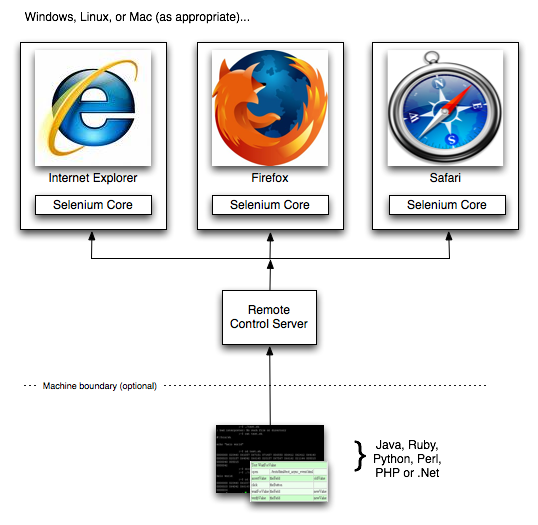
**Selenium Remote Control** (RC) is a test tool that allows you to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.

Selenium RC comes in two parts.

1. A server which automatically launches and kills browsers, and acts as an HTTP proxy for web requests from them.
2. Client libraries for your favorite computer language (including Java).

The RC server also bundles [Selenium Core](http://seleniumhq.org/projects/core/), and automatically loads it into the browser.

Here is a simplified architectural representation....



Selenium Remote Control is great for testing complex AJAX-based web user interfaces under a Continuous Integration system. It is also an ideal solution for users of Selenium Core or Selenium IDE who want to write tests in a more expressive programming language than the Selenese HTML table format customarily used with Selenium Core.

The iConsole Automated Test Suite is a Java project that uses the Selenium client libraries to execute tests against the iConsole on different browsers. The Test Suite sends commands to the Selenium Server (which must already be running) and acts on the return values. It provides comprehensive logging to the user to review test success.

As Selenium can only interact with the web browser running the suite the Java must also perform some other tasks to facilitate effective testing:

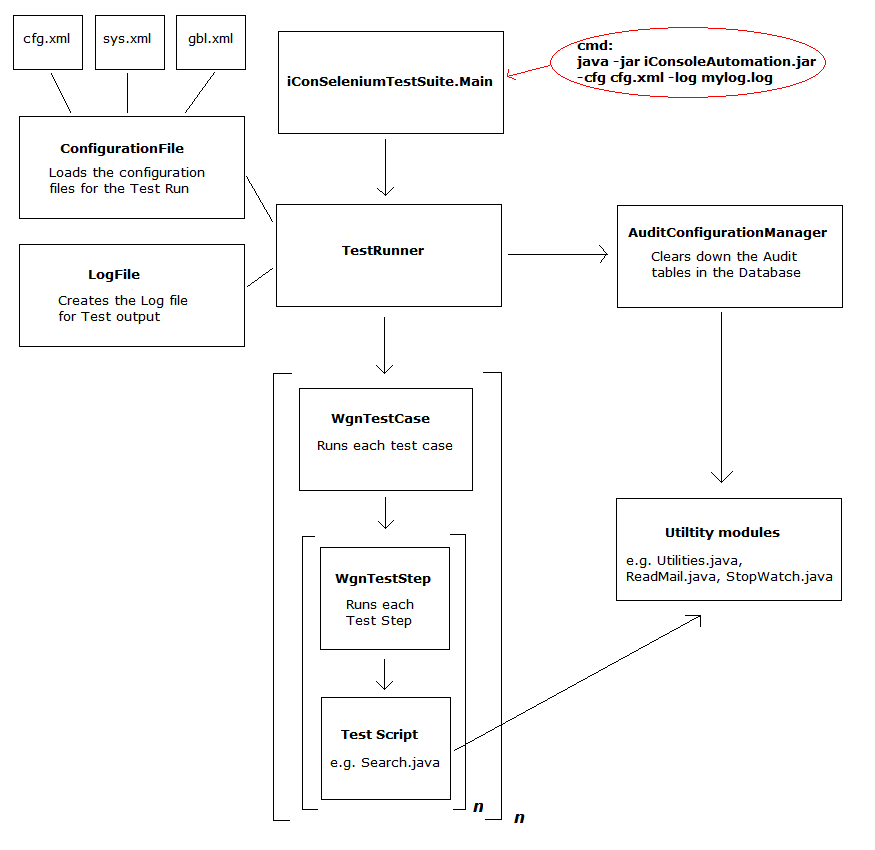
1. Database interaction: to review test outcomes it is necessary to know certain things about the CA DLP CMS configuration. This requires direct querying of the CMS DB and Java libraries for SQL Server and Oracle have been included in the project to provide this possibility. It is also possible to change database values in this way to force certain configurations without needing to automate other CA DLP components.
2. IIS: some iConsole config changes don’t take effect until after a timeout or an IIS reset. Waiting for the timeout would makes tests too lengthy and the browser session could timeout first. Therefore in some cases it can be necessary to force an IIS Reset on the iConsole web server. This can be achieved with the psexec utility, the use of which can be automated in the Java suite.
3. Registry keys: Much of the iConsole configuration is set in the registry. To test the effect of configuration changes it is therefore necessary to change the registry values. Java has existing libraries for this and if remote registry interaction is required psexec is also an option.
4. System pop-ups: Selenium can only interact with the browser. Where a system dialog is encountered e.g. a Browse dialog for file upload, we need a way of passing the necessary information to the dialog and dismissing it. A tool called AutoIt exists for this purpose. It can be used to create executables specific for the dialog you need to automate and these exes can then be called via Java code in the suite.
5. Mail verification: the iConsole can send Audit mails to senders/recipients of captured events on the CMS. To verify these mails have been sent correctly with correct flags and content etc there is a Java library available called Javamail which can log into a Mail server and open a user’s mailbox to retrieve specific mail.

## Sample Flow Execution Diagram

With what other components does this component need to interact and how do these relationships work? Often, you will be able to depict these relationships graphically using structure charts, data flow diagrams or transaction diagrams. Show a sample execution of features including the flow and interaction of different components involved.

The Automated Test Suite is packaged as a Jar file and designed to be run from the cmd line. Ultimately it should be included in the CTC project which will further automate deployment, execution and reporting of the test cases.

The following diagram shows how the java scripts that make up the Test Suite hang together and are called during a test run:



# Design Limitations, Assumptions, and Issues

This section should list current limitations and assumptions made in the design. These may include unique characteristics or testing requirements.

1. It was decided to code the test suite in Java as this is Selenium’s native language and so support and integration was likely to be smoothest. Java also provides the necessary functionality for DBMS, registry and system interaction that using Selenium IDE alone would not provide. JUnit was not used for test case control however as likely integration with the existing QA CTC Controller would provide overlapping functionality.
2. To ensure results of tests can be predicted at the beginning of every test suite run the CMS DB Audit data tables will be truncated to ensure that NO audit issues pre-exist for the test. This means you should not run the Test Suite on a production or shared Test server as you will destroy all existing Audit data on the system.

### Issues

Since the DDS is an evolving representation of the design, this section is used to keep track of issues and items that need special attention.

\* If PRIME Clarity is used to manage the issues or risks, include the Issue Name and ID, and the remaining fields can be blank.

| **Issue Name** | **Description** | **Priority** | **Resolution** |
| --- | --- | --- | --- |
| Firefox 4 | Not supported by Selenium Server 1.0.3 | 2 | Requires Selenium Server upgrade to 2.0 – when available |
|  |  |  |  |

# Internals

## Programming details

List procedures, modules and programs to be written, changed or deleted to meet the module specifications. Create as many subsections as needed for each module/program that is affected. All external UI elements and messages (strings) must be externalized to a runtime loadable module to be an i18n enabled product. For a list of i18n enabled products, go to: <http://intranet.ca.com/develop/local/mlsp.htm>.

The heart of the implementation is the iConsole\_Automation.jar file which packages together all the java scripts that enable running of the test suite. Each script and its function plus example usage is detailed in this section.

### The Test Controller (TestRunner.java)

Provide a description of the module, including the purpose, functionality, and interdependency. Describe major ideas and concepts that pertain to the design of this module. If this is an add-on feature, describe the design changes required in existing support modules.

This script is called by the iConSeleniumTestSuite.Main function and manages the test run. It drives loading the configuration parameters, opening the log file, flushing the Audit data from the CMS DB, launching the Test cases and finally closing everything down again. It passes errors back up to the Main function on completion (or abortion) of the run.

Public functions for use in Test scripts:

NewLogFileGroup(String msg, int loglevel) – adds an opening line for a logging group in the log file and indents all following lines until CloseLogFileGroup() is called

CloseLogFileGroup(String msg, int loglevel) – removes the indent and writes a closing line for the group

ReportError(String s) – report fatal or major error conditions, forces the exit code to 1 for the test suite.

### Test Cases (WgnTestCase.java, WgnTestStep.java, ConfigurationTestStep.java, Test scripts)

Multiple Test Cases can be defined in a single cfg.xml file and the TestRunner will load and call each of them in turn. Likewise a Test Case can be made up of multiple Test Steps e.g. a step that configures and run a standard search and a step that performs tests on the results returned and finally a step that cleans up afterwards (deletes derived searches, resets prefs etc). The Test Steps are the test scripts that define the actual steps and tests to be performed during the running of the test case.

The WgnTestCase, WgnTestStep and ConfigurationTestStep scripts handle loading and running the test cases and steps and shouldn’t generally need any changes for new Test scripts added. New test scripts can be created to cover as yet untouched areas of the iConsole or existing test scripts can be expanded to include new tests related to their existing tests.

Existing test scripts can also be called as steps for new test cases added. The Search.java script is particularly useful for this as it runs a standard search with a variety of parameter options and presents a results page. New test scripts can then be added to perform tests on the results page.

Currently the Test scripts in the suite are:

**Search.java** – runs the standard search with a variety of parameters

**Audit.java** – runs all the manual and bulk audit operations

**AuditOptions.java** – checks iConsole display and behavior against the many Audit configuration options available in the Acon (config is updated directly in the DB)

**QuickViewPanes.java** – builds on Search.java to exercise the various options in the QV panes

**Admin\_tab.java** – tests the functions available on the Administer Searches tab

**AdminDerivedSearches.java** – creates and uses a derived search

**MyPrefs.java** – tests local login user preferences

**Gprefs.java** – tests the Global user preferences

**ColumnSelector.java** – builds on Search.java to exercise the column selector dialog

**SearchTabView.java** – verifies default search and report availability and parameter defaults

**SendAuditMail.java** - builds on Search.java to verify the Audit Mail options on the QV pane

### Configuration module (ConfigurationFile.java)

This module handles loading and parsing of the configuration parameters for the test run. These parameters are provided by the user in 3 XML files – sys.xml, cfg.xml & gbl.xml.

Sys.xml – contains System parameters for the run such as the location of the iConsole and CMS DB and the desired log level.

Cfg.xml – defines the Test cases and their Test steps for the run

Gbl.xml – contains a set of global parameters that are common to several test cases

### Audit Configuration module (AuditConfigurationManager.java)

This module provides interaction with the CA DLP CMS DB, both Oracle and SQL Server. It can truncate delete all the audit data in the database to provide a clean starting point for audit tests and also set and retrieve specific audit configuration values from WgnWellKnownString.

The Audit Configuration module uses several supporting modules:

#### AuditActionValue.java

Retrieves the current Audit Action (Audit Field 2) values from the DB.

#### AuditStatusValue.java

Retrieves the current Audit Status (Audit Field 1) values from the DB.

#### AuditResolutionValue.java

Retrieves the current Audit Resolution (Audit Field 3) values from the DB.

#### AuditComment.java

Retrieves the current Audit comments from the DB.

#### AuditButton.java

Retrieves the current Audit Button configuration from the DB.

#### SelectElement.java

Contains functions for validating the contents of the Audit Field select elements paying particular attention to the dependencies between the 3 Audit Fields.

#### UserPreferences.java

Returns the values of the ‘Move to Next Event After Auditing’ and ‘Remove Event after Auditing’ preferences for the logged in user.

### Utilities (Utilities.java)

This script contains a set of functions that are common to many or all test cases e.g. the Login(), Logoff() and DoClick() functions. The functions and their example usage are listed below:

#### Login(WgnTestCase wtc, boolean defLogin)

Performs a non-SSO login with the user name and password passed for the current Test Case in the cfg.xml file. For SSO don’t use this function, just set ‘SSO=true’ for the Test Case in the cfg.xml file.

#### Logoff(WgnTestCase wtc)

Performs a logout.

#### DoClick (WgnTestCase wtc, String clickElement, String waitElement, String timeout)

This function wraps the Selenium browser Click() function which sends a left mouseclick command to the defined element in the browser window. The wrapper allows us to check for and catch any js errors that occur onclick or on the post-click action called. It also takes an optional wait condition that makes the suite wait until the condition is true.

Example usage: Utilities.DoClick(mTestCase, "id=tab\_global", "selenium.isElementPresent('id=MoveAfterAuditEvent\_Enforced')", "10000");

There’s also a newer version of the function which also allows you to optionally suppress any logging in the function using the DoLog parameter – useful where the function is called in a loop that produces a distracting and unhelpful number of log lines:

DoClick(WgnTestCase wtc, String clickElement, String waitElement, String timeout, Boolean doLog)

#### replaceChars(String oldStr) returns String

The following characters are escaped to '\_' in event subjects before using in window titles, filenames etc:

\* '?', '[', ']', '/', '\\', '=', '+', '<', '>', ':', ';', '\',' ',,' '\*', '#', '&', '\"'

So we need to do the same before comparing the subject line to any retrieved window titles, filenames etc

#### isElementPresent(WgnTestCase wtc, String strId, String strDesc, Boolean b)

Checks whether an element is present on the page or not and logs the outcome, ***without*** throwing an exception and ending the test.

#### isCheckboxChecked(WgnTestCase wtc, String strId, String strDesc, String strChecked)

Checks whether a checkbox is checked or unchecked and logs the outcome. Pass ‘true’ or ‘false’ for StrChecked as the expected value.

#### compareSelectLabel(WgnTestCase wtc, String strId, String strDesc, String strValue)

Check that actual selected value in a Select dropdown matches the expected value passed in strValue

#### compareTextBoxValue(WgnTestCase wtc, String strId, String strDesc, String strValue)

Check that actual text box default value matches the expected value passed in strValue.

#### SetSearchAttributes(WgnTestCase wtc, Selenium browser)

Retrieves the numSearchResults & pageSize values from the DOM for a search results page and sets test case attributes to the returned values, for use in further tests later.

#### GetUserPreferences(WgnTestCase wtc, Selenium browser)

Retrieves the current values of the “Remove Event after Audit” and “Move to Next Event after Audit” parameters for the logged in user and sets them as the values of global parameters (removeAuditedEvents, moveNextEvent) for use in further tests.

#### GetParamAttribute(WgnTestCase wtc, String name, String attrib, String expected)

Retrieves the list of search parameters used to generate a search results page from the DOM for the results page and check s that the actual value for the parameter named in ‘name’ matches the expected value passed in ‘expected’. The outcome is written as PASS or FAIL to the log file.

Example usage:

Utilities.GetParamAttribute(mTestCase, "dateRange", "attrs", "1;2;1;1");

Utilities.GetParamAttribute(mTestCase, "E-mail Events only", "value", "true");

Utilities.GetParamAttribute(mTestCase, "txtEEAddress1", "value",

mParameters.get("emailaddress"));

### Logging module (LogFile.java)

This module creates and manages the log file for test run output. One log file is created per test suite run and the default file name is ‘Selenium\_DD-MM-YYYY-hhmmss.txt’. This name can be overridden on the cmd line using the –log parameter when iConsole\_Automation.jar is first executed. There are 6 log levels: 0-none, 1-errors, 2-Known errors, 3-Tests, 4-Info, 5-debug. By default the log level is 0. Fatal errors will always be written to the log file (i.e. exceptions).

The following functions are publically available in the LogFile script:

Log(String str, int indent, int loglevel) – the standard

Write(String line) – bypasses the log level and ALWAYS writes to the log file

Example usage: mTestRunner.NewLogFileGroup("Test that changes to My Profile are Saved", LogFile.iINFO);

mTestRunner.WriteLogFileEntry("End of My Profile tests as " + mTestCase.GetAttribute("username"), LogFile.iINFO);

### Timer module (StopWatch.java)

Contains functions for timing the execution length of tests. E.g. you can start the timer running then run a standard search and stop the timer again as soon as the progress dialog has disappeared.

Example:

StopWatch itimer = new StopWatch().start();

Utilities.DoClick(mTestCase, "id=btn\_Customize.Run", "!selenium.browserbot.getCurrentWindow().document.getElementById('progressContent')", "20000");

itimer.stop();

Then to get the value of time elapsed use e.g.

String myTime = String.valueOf(itimer.getElapsedTime());

### Mail verification module (ReadMail.java)

Opens a user’s mailbox and retrieves the most recent mail to verify the correct delivery and configuration of iConsole Audit mail. Uses the mail-1.4.3.jar java library to open the mailbox via the exchange’s pop3 server (this won’t be running by default so you need to configure it on the exchange server box).

### Error Reporting

The majority of error conditions returned by the Test Suite will be caused by configuration settings. When a configuration error or Java exception is encountered the test suite will exit with error code=1.

Where genuine iConsole errors are encountered they will be handled as follows:

Javascript errors: These are always assumed to be fatal conditions as we can’t predict the behavior of the iConsole after one has occurred and so the test suite exits with exit code=1 and the js error is recorded in the log file.

Other errors: These may be incorrect defaults for parameters or unexpected return values for actions performed. They generally will be reported in the log file as a FAIL and the test suite will continue to run and when it completes exit with exit code=2 to indicate a problem was encountered. If the unexpected occurrence stops the script running (because it provokes a Selenium or Java exception) then the suite will exit immediately with exit code=1.

Known errors: Where a known (i.e. bugged) iConsole deficiency is encountered the code will be changed to report the FAIL as a KNOWN error and the test suite will exit with exit code=3 when complete.

### Supporting Tools

**AutoIT:** To automate population and dismissal of System pop-ups (domain login dialogs, File Upload, Browse dialogs) a 3rd Party Tool is required. Auto IT is designed for this purpose and allows you to build standalone executables that are scripted to deal with different pop-ups as they occur. These exes can then be launched by the Java Test Suite and they sit in the background waiting for a matching pop-up to appear, do their scripted stuff and exit.

Currently the Test Suite utilizes two of these exes:

#1 test\_Searches.exe: to handle the File Upload dialog that appears when installing new searches

#2 ff\_login.exe: to handle the domain security login dialog that Firefox launches when the browser opens.

**PSExec:** Some iConsole configuration changes don’t take effect until the web cache times out, which can be up to 1 hour. This is no good for automated testing as the browser session will timeout first and it would take too long to run anyway. Therefore it is necessary to reload the cache which can be actioned by performing an IISReset on the web server. PSExec can be used to automate remote tasks like this and can be called using the Java RunExec() function.

# Internationalization

All CA products and common components must comply with the [Product Internationalization Policy](http://qms.ca.com/documents/default.asp?srchID=9332). Provide a description of how the component or design entity supports the internationalization requirements defined in the PRD. Additional information about these requirements can be found in the related documents: [Internationalization Requirements](http://cawiki.ca.com/download/attachments/17797398/Internationalization+Requirements_v1.0.docx)  and [Internationalization Standards](http://cawiki.ca.com/download/attachments/17797398/Internationalization+Standards_v1.1.docx).

All strings used in text match comparisons tests or as identifiers for iConsole elements are passed into the test suite from the cfg.xml and gbl.xml configuration files. This provides a central resource that can be localized as required and ensures that the test scripts themselves are language-agnostic.

## Localization

If the product is localized, files required by the UI and messages must be externalized from the code. The identification of particular files needs to be considered but does not need to be included in the document as this information will be captured as part of the localization process. Refer to the [Localization Procedure](http://canet.ca.com/develop/local/style_guide/) (<http://canet.ca.com/develop/local/style_guide/>) for execution details.

No localization is currently planned.

# Configuration files

## System configuration

This file contains the system parameters necessary to open the browser, login to its domain (if necessary), login into the CMS DBMS and load the iConsole.

The default file name is sys.xml.

The available parameters are (all required unless otherwise stated):

**seleniumhost** = the location where the selenium server is running, relative to the Java

Test suite

**targetbrowser** = the browser type e.g. “\*iexplore”, “\*firefox”, “\*chrome”

**firefoxtitle, firefoxuser, firefoxpassword** [required for Firefox] = login details for the

Browser’s domain to dismiss the security popup that Firefox always displays. The title is the security pop-up’s title, usually “Authentication Required” for Firefox. The IE popup could be dismissed the same way but it has a “remember me” checkbox so it’s easier to manually complete this once and then it doesn’t appear again.

**cms** = the location of the CMS that the iConsole under test accesses.

**database** = the name of the CMS DBMS database e.g. WGN\_myDB

**dbtype** = the type of the CMS DBMS, ‘oracle’ or ‘sqlserver’

**databaseserver** = the location of the DBMS server

**dbusername, dbpassword** = login details for the DBMS. Use the ‘sa’ system login for

sqlserver and ‘wgnowner’ or ‘wgnuser’ for ‘oracle’.

**protocol** = web server protocol, “http://” or “https://”

**webserver** = the iConsole host server

**virtualdir** = the web server directory for the iConsole, usually ‘CADLP’

**loglevel** [optional] = default: none. The output level for the Test suite:

0=none, 1=errors only, 2=+Known errors, 3=+tests, 4=Informational, 5=debug

Example system configuration file:

<?xml version="1.0" encoding="UTF-8"?>

<root>

<config

seleniumhost="localhost" targetbrowser="\*firefox"

firefoxtitle="Authentication Required"

fierfoxuser="tb\administrator" firefoxpassword="yeo"

cms="593-TB-FAST"

database="WGN\_593-TB-FAST" dbtype="sqlserver"

databaseserver="593-TB-FAST.tb.com"

dbusername="sa" dbpassword="avon"

protocol="http://" webserver="593-tb-fast.tb.com" virtualdir="CADLP"

loglevel="4"

/>

</root>

## Global Configuration file

This file contains a list of parameters that are global to several or all tests. It is mostly a list of string parameters that will need to be translated if the test suite is run against a localized iConsole. It shouldn’t be changed except in the case of a translation of the parameter values.

The default file name is gbl.xml.

## Test Configuration file

This file defines which tests will be run in the next test suite pass. Each Test case is specified with a <test> tag and can contain multiple <step> tags to define the the test steps.

The default file name is cfg.xml.

The <test> tag takes these parameters:

**name** = the test case name (for logging)

**SSO** = are we using single sign-on for the iConsole, true or false

**username, password** = primary iConsole login for the test (if SSO=’false’)

**rlsusername, rlspassword** = secondary login info for tests that use multiple logins

The <step> tag takes these parameters:

**classname** = the name of the test script being called by the test step (an example xml is included in the test suite handoff for each test case available which will pre-populate this value).

**name** = the test step name (for logging)

Example Test configuration file:

<?xml version="1.0" encoding="UTF-8"?>

<root>

<test name="Column Selector" sso="false" username="Admin" password="axe" >

<step classname="testcases.Search" name="Run Search">

<parameter name="searchtimeout" value="60"/>

<parameter name="back" value="false" />

</step>

<step classname="testcases.ColumnSelector" name="Column Selector">

</step>

</test>

</root>

# Running the Test Suite

The Test Suite is run by calling the compiled iConsole\_automation.jar from the Command line. All parameters are optional as defaults are used if omitted.

The default configuration xml files are sys.xml, cfg.xml and gbl.xml.

The default log file format is: Selenium\_DD-MM-YYYY\_hhmmss.txt

**Usage:** iConSeleniumTestSuite [-vb] [-sys <filename>] [-gbl <filename>] [-cfg <filename>] [-log <log filename>]

Where: -vb gives verbose command line output

-sys specifies an xml file containing system configuration parameters

-cfg specifies an xml file containing test case-specific configuration parameters

-gbl specifies an xml file containing global test case configuration parameters

-log specifies an output file for test results (created in the logs subdirectory)

**Start the Selenium Server running**:

a. Open a Command Prompt and navigate to the location of your java.exe file

e.g. C:\Program Files\Java\jdk1.6.0\_21\bin

b. Run the Selenium Server:

java -jar "C:\Program Files\Java\selenium-server-1.0.3\selenium-server.jar" -userExtensions user-extensions.js

**Example of running from the Command line:**

Open a Command Prompt and go to the directory containing the Test Suite jar file.

Set your Java classpath e.g.

path=C:\Program Files\Java\jdk1.6.0\_21\bin;./lib;

Run the Test suite:

Java –jar iConsole\_automation.jar –vb –cfg cfg\_audit.xml –log mylog.log

# Packaging and Installation

This section should be used when the feature has an impact on the packaging or installation. If so, indicate and detail any special packaging or installation requirements. Include conversion utilities, compatibility, and migration issues for existing customers. Detail any new files that are required, as well as any new licensing requirements.

* Dependencies
* Configuration
* Install Paths
* Multi-language information
* Upgrade Information
* Registry Information
* Default Installation
* Licensing Information (also see [Licensing Requirements Checklist](http://qms.ca.com/documents/default.asp?srchID=1158))
* Common Component levels to be upgraded

The package handoff can be found at <//130.119.47.199/Selenium>

## Setup

Before configuring the Test Suite you will first need to have installed a JVM and the Selenium Server to your test server.

Java can be downloaded from: <http://www.java.com/en/download/index.jsp>

Selenium Server can be downloaded from: <http://seleniumhq.org/download>

## Installing the Test Suite

1. Unzip the Test Suite from icon\_auto-x.x.x.zip to your hard drive.

2. Two search definition files, test\_Searches.xml and test\_Reports.xml, are included in the package at <dist>\xml. Set their location in the ‘searchesPath’ and ‘reportsPath’ parameters in cfg\_admintab.xml.

3. Copy user-extensions.js from <dist>\user-extensions to the bin folder of your JVM

e.g. C:\Program Files\Java\jdk1.6.0\_21\bin

4. Configuration files:

a. Copy config\sys.xml file to the same location as iconsole\_automation.jar.

b. Configure the parameters in sys.xml for your system.

c. Use the cfg\_xxx.xml files to define which tests to run (a default cfg file containing example scripts for all test cases is packaged in the zip and installed to the same location as the jar).

d. Do NOT edit gbl.xml

# Appendix - Selenium Javadoc

URL for latest updates: <http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html>

**Package**: com.thoughtworks.selenium   
**Interface**: public interface Selenium

All Known Implementing Classes: [DefaultSelenium](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/DefaultSelenium.html)

**public interface Selenium**: Defines an object that runs Selenium commands

## Element Locators

Element Locators tell Selenium which HTML element a command refers to. The format of a locator is:

locatorType**=**argument

Supports the following strategies for locating elements:

* **identifier**=id: Select the element with the specified @id attribute. If no match is found, select the first element whose @name attribute is id. (This is normally the default; see below.)
* **id**=id: Select the element with the specified @id attribute.
* **name**=name: Select the first element with the specified @name attribute.
  + username
  + name=username

The name may optionally be followed by one or more element-filters, separated from the name by whitespace. If the filterType is not specified, **value** is assumed.

* + name=flavour value=chocolate
* **dom**=javascriptExpression: Find an element by evaluating the specified string. This allows you to traverse the HTML Document Object Model using JavaScript. Note that you must not return a value in this string; simply make it the last expression in the block.
  + dom=document.forms['myForm'].myDropdown
  + dom=document.images[56]
  + dom=function foo() { return document.links[1]; }; foo();
* **xpath**=xpathExpression: Locate an element using an XPath expression.
  + xpath=//img[@alt='The image alt text']
  + xpath=//table[@id='table1']//tr[4]/td[2]
  + xpath=//a[contains(@href,'#id1')]
  + xpath=//a[contains(@href,'#id1')]/@class
  + xpath=(//table[@class='stylee'])//th[text()='theHeaderText']/../td
  + xpath=//input[@name='name2' and @value='yes']
  + xpath=//\*[text()="right"]
* **link**=textPattern: Select the link (anchor) element which contains text matching the specified pattern.
  + link=The link text
* **css**=cssSelectorSyntax: Select the element using css selectors. Please refer to [CSS2 selectors](http://www.w3.org/TR/REC-CSS2/selector.html), [CSS3 selectors](http://www.w3.org/TR/2001/CR-css3-selectors-20011113/) for more information. You can also check the TestCssLocators test in the selenium test suite for an example of usage, which is included in the downloaded selenium core package.
  + css=a[href="#id3"]
  + css=span#firstChild + span

Currently the css selector locator supports all css1, css2 and css3 selectors except namespace in css3, some pseudo classes(:nth-of-type, :nth-last-of-type, :first-of-type, :last-of-type, :only-of-type, :visited, :hover, :active, :focus, :indeterminate) and pseudo elements(::first-line, ::first-letter, ::selection, ::before, ::after).

* **ui**=uiSpecifierString: Locate an element by resolving the UI specifier string to another locator, and evaluating it. See the [Selenium UI-Element Reference](http://svn.openqa.org/fisheye/browse/~raw,r=trunk/selenium/trunk/src/main/resources/core/scripts/ui-doc.html) for more details.
  + ui=loginPages::loginButton()
  + ui=settingsPages::toggle(label=Hide Email)
  + ui=forumPages::postBody(index=2)//a[2]

Without an explicit locator prefix, Selenium uses the following default strategies:

* **dom**, for locators starting with "document."
* **xpath**, for locators starting with "//"
* **identifier**, otherwise

## Element Filters

Element filters can be used with a locator to refine a list of candidate elements. They are currently used only in the 'name' element-locator.

Filters look much like locators, i.e.

filterType**=**argument

Supported element-filters are:

**value=**valuePattern

Matches elements based on their values. This is particularly useful for refining a list of similarly-named toggle-buttons.

**index=**index

Selects a single element based on its position in the list (offset from zero).

## String-match Patterns

Various Pattern syntaxes are available for matching string values:

* **glob:**pattern: Match a string against a "glob" (aka "wildmat") pattern. "Glob" is a kind of limited regular-expression syntax typically used in command-line shells. In a glob pattern, "\*" represents any sequence of characters, and "?" represents any single character. Glob patterns match against the entire string.
* **regexp:**regexp: Match a string using a regular-expression. The full power of JavaScript regular-expressions is available.
* **regexpi:**regexpi: Match a string using a case-insensitive regular-expression.
* **exact:**string: Match a string exactly, verbatim, without any of that fancy wildcard stuff.

If no pattern prefix is specified, Selenium assumes that it's a "glob" pattern.

For commands that return multiple values (such as verifySelectOptions), the string being matched is a comma-separated list of the return values, where both commas and backslashes in the values are backslash-escaped. When providing a pattern, the optional matching syntax (i.e. glob, regexp, etc.) is specified once, as usual, at the beginning of the pattern.

## Available Selenium Methods

### Startup, Shutdown and Configuration commands

setExtensionJs

void **setExtensionJs**(java.lang.String extensionJs)

Sets the per-session extension JavaScript

start

void **start**()

Launches the browser with a new Selenium session

start

void **start**(java.lang.String optionsString)

Starts a new Selenium testing session with a String, representing a configuration

start

void **start**(java.lang.Object optionsObject)

Starts a new Selenium testing session with a configuration options object

stop

void **stop**()

Ends the test session, killing the browser

shutDownSeleniumServer

void **shutDownSeleniumServer**()

Kills the running Selenium Server and all browser sessions. After you run this command, you will no longer be able to send commands to the server; you can't remotely start the server once it has been stopped. Normally you should prefer to run the "stop" command, which terminates the current browser session, rather than shutting down the entire server.

setSpeed

void **setSpeed**(java.lang.String value)

Set execution speed (i.e., set the millisecond length of a delay which will follow each selenium operation). By default, there is no such delay, i.e., the delay is 0 milliseconds.

**Parameters:** value - the number of milliseconds to pause after operation

getSpeed

java.lang.String **getSpeed**()

Get execution speed (i.e., get the millisecond length of the delay following each selenium operation). By default, there is no such delay, i.e., the delay is 0 milliseconds. See also setSpeed.

**Returns:** The execution speed in milliseconds.

attachFile

void **attachFile**(java.lang.String fieldLocator,java.lang.String fileLocator)

Sets a file input (upload) field to the file listed in fileLocator

**Parameters:**

fieldLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

fileLocator - a URL pointing to the specified file. Before the file can be set in the input field (fieldLocator), Selenium RC may need to transfer the file to the local machine before attaching the file in a web page form. This is common in selenium grid configurations where the RC server driving the browser is not the same machine that started the test. Supported Browsers: Firefox ("\*chrome") only.

### Screenshot commands

captureScreenshot

void **captureScreenshot**(java.lang.String filename)

Captures a PNG screenshot to the specified file.

**Parameter:** filename - the absolute path to the file to be written, e.g. "c:\blah\screenshot.png"

captureScreenshotToString

java.lang.String **captureScreenshotToString**()

Capture a PNG screenshot. It then returns the file as a base 64 encoded string.

**Returns**: The base 64 encoded string of the screen shot (PNG file)

captureEntirePageScreenshotToString

java.lang.String **captureEntirePageScreenshotToString**(java.lang.String kwargs)

Downloads a screenshot of the browser current window canvas to a base 64 encoded PNG file. The entire windows canvas is captured, including parts rendered outside of the current view port. Currently this only works in Mozilla and when running in chrome mode.

**Parameter:** kwargs - A kwargs string that modifies the way the screenshot is captured. Example: "background=#CCFFDD". This may be useful to set for capturing screenshots of less-than-ideal layouts, for example where absolute positioning causes the calculation of the canvas dimension to fail and a black background is exposed (possibly obscuring black text).

Returns: The base 64 encoded string of the page screenshot (PNG file)

captureEntirePageScreenshot

void **captureEntirePageScreenshot**

(java.lang.String filename, java.lang.String kwargs)

Saves the entire contents of the current window canvas to a PNG file. Contrast this with the captureScreenshot command, which captures the contents of the OS viewport (i.e. whatever is currently being displayed on the monitor), and is implemented in the RC only. Currently this only works in Firefox when running in chrome mode, and in IE non-HTA using the EXPERIMENTAL "Snapsie" utility. The Firefox implementation is mostly borrowed from the Screengrab! Firefox extension. Please see http://www.screengrab.org and http://snapsie.sourceforge.net/ for details.

**Parameters:**

filename - the path to the file to persist the screenshot as. No filename extension will be appended by default. Directories will not be created if they do not exist, and an exception will be thrown, possibly by native code.

kwargs - a kwargs string that modifies the way the screenshot is captured. Example: "background=#CCFFDD". Currently valid options:

background: the background CSS for the HTML document. This may be useful to set for capturing screenshots of less-than-ideal layouts, for example where absolute positioning causes the calculation of the canvas dimension to fail and a black background is exposed (possibly obscuring black text).

### Keyboard and Mouse commands

click

void **click**(java.lang.String locator)

Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad or waitForCondition.

**Parameter:** locator - an element locator

clickAt

void **clickAt**(java.lang.String locator, java.lang.String coordString)

Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.

**Parameters:**

locator - an element locator

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

contextMenu

void **contextMenu**(java.lang.String locator)

Simulates opening the context menu for the specified element (as might happen if the user "right-clicked" on the element).

**Parameter:** locator - an element locator

contextMenuAt

void **contextMenuAt**(java.lang.String locator, java.lang.String coordString)

Simulates opening the context menu for the specified element (as might happen if the user "right-clicked" on the element).

**Parameters:**

locator - an element locator

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

doubleClick

void **doubleClick**(java.lang.String locator)

Double clicks on a link, button, checkbox or radio button. If the double click action causes a new page to load (like a link usually does), call waitForPageToLoad.

**Parameter:** locator - an element locator

doubleClickAt

void **doubleClickAt**(java.lang.String locator, java.lang.String coordString)

Doubleclicks on a link, button, checkbox or radio button. If the action causes a new page to load (like a link usually does), call waitForPageToLoad.

**Parameters:**

locator - an element locator

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

fireEvent

void **fireEvent**(java.lang.String locator, java.lang.String eventName)

Explicitly simulate an event, to trigger the corresponding "onevent" handler.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

eventName - the event name, e.g. "focus" or "blur"

focus

void **focus**(java.lang.String locator)

Move the focus to the specified element; for example, if the element is an input field, move the cursor to that field.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

keyPress

void **keyPress**(java.lang.String locator, java.lang.String keySequence)

Simulates a user pressing and releasing a key.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

keySequence - Either be a string("\" followed by the numeric keycode of the key to be pressed, normally the ASCII value of that key), or a single character. For example: "w", "\119".

shiftKeyDown

void **shiftKeyDown**()

Press the shift key and hold it down until doShiftUp() is called or a new page is loaded.

shiftKeyUp

void **shiftKeyUp**()

Release the shift key.

metaKeyDown

void **metaKeyDown**()

Press the meta key and hold it down until doMetaUp() is called or a new page is loaded.

metaKeyUp

void **metaKeyUp**()

Release the meta key.

altKeyDown

void **altKeyDown**()

Press the alt key and hold it down until doAltUp() is called or a new page is loaded.

altKeyUp

void altKeyUp()

Release the alt key.

controlKeyDown

void **controlKeyDown**()

Press the control key and hold it down until doControlUp() is called or a new page is loaded.

controlKeyUp

void **controlKeyUp**()

Release the control key.

keyDown

void **keyDown**(java.lang.String locator, java.lang.String keySequence)

Simulates a user pressing a key (without releasing it yet).

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

keySequence - Either be a string("\" followed by the numeric keycode of the key to be pressed, normally the ASCII value of that key), or a single character. For example: "w", "\119".

keyUp

void **keyUp**(java.lang.String locator, java.lang.String keySequence)

Simulates a user releasing a key.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

keySequence - Either be a string("\" followed by the numeric keycode of the key to be pressed, normally the ASCII value of that key), or a single character. For example: "w", "\119".

keyDownNative

void **keyDownNative**(java.lang.String keycode)

Simulates a user pressing a key (without releasing it yet) by sending a native operating system keystroke. This function uses the java.awt.Robot class to send a keystroke; this more accurately simulates typing a key on the keyboard. It does not honor settings from the shiftKeyDown, controlKeyDown, altKeyDown and metaKeyDown commands, and does not target any particular HTML element. To send a keystroke to a particular element, focus on the element first before running this command.

**Parameter:** keycode - an integer keycode number corresponding to a java.awt.event.KeyEvent; note that Java keycodes are NOT the same thing as JavaScript keycodes!

keyUpNative

void **keyUpNative**(java.lang.String keycode)

Simulates a user releasing a key, by sending a native operating system key stroke. This function uses the java.awt.Robot class to send a keystroke; this more accurately simulates typing a key on the keyboard. It does not honor settings from the shiftKeyDown, controlKeyDown, altKeyDown and metaKeyDown commands, and does not target any particular HTML element. To send a keystroke to a particular element, focus on the element first before running this command.

**Parameter:** keycode - an integer keycode number corresponding to a java.awt.event.KeyEvent; note that Java keycodes are NOT the same thing as JavaScript keycodes!

keyPressNative

void **keyPressNative**(java.lang.String keycode)

Simulates a user pressing and releasing a key by sending a native operating system keystroke. This function uses the java.awt.Robot class to send a keystroke; this more accurately simulates typing a key on the keyboard. It does not honor settings from the shiftKeyDown, controlKeyDown, altKeyDown and metaKeyDown commands, and does not target any particular HTML element. To send a keystroke to a particular element, focus on the element first before running this command.

**Parameter:** keycode - an integer keycode number corresponding to a java.awt.event.KeyEvent; note that Java keycodes are NOT the same thing as JavaScript keycodes!

mouseOver

void **mouseOver**(java.lang.String locator)

Simulates a user hovering a mouse over the specified element.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

mouseOut

void **mouseOut**(java.lang.String locator)

Simulates a user moving the mouse pointer away from the specified element.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

mouseDown

void **mouseDown**(java.lang.String locator)

Simulates a user pressing the left mouse button (without releasing it yet) on the specified element.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

mouseDownRight

void **mouseDownRight**(java.lang.String locator)

Simulates a user pressing the right mouse button (without releasing it yet) on the specified element.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

mouseDownAt

void **mouseDownAt**(java.lang.String locator, java.lang.String coordString)

Simulates a user pressing the left mouse button (without releasing it yet) at the specified location.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

mouseDownRightAt

void **mouseDownRightAt**

(java.lang.String locator, java.lang.String coordString)

Simulates a user pressing the right mouse button (without releasing it yet) at the specified location.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

mouseUp

void **mouseUp**(java.lang.String locator)

Simulates the event that occurs when the user releases the mouse button (i.e., stops holding the button down) on the specified element.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

mouseUpRight

void **mouseUpRight**(java.lang.String locator)

Simulates the event that occurs when the user releases the right mouse button (i.e., stops holding the button down) on the specified element.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

mouseUpAt

void **mouseUpAt**(java.lang.String locator, java.lang.String coordString)

Simulates the event that occurs when the user releases the mouse button (i.e., stops holding the button down) at the specified location.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

mouseUpRightAt

void **mouseUpRightAt**(java.lang.String locator, java.lang.String coordString)

Simulates the event that occurs when the user releases the right mouse button (i.e., stops holding the button down) at the specified location.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

mouseMove

void **mouseMove**(java.lang.String locator)

Simulates a user pressing the mouse button (without releasing it yet) on the specified element.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

mouseMoveAt

void **mouseMoveAt**(java.lang.String locator, java.lang.String coordString)

Simulates a user pressing the mouse button (without releasing it yet) on the specified element.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

type

void **type**(java.lang.String locator, java.lang.String value)

Sets the value of an input field, as though you typed it in.

Can also be used to set the value of combo boxes, check boxes, etc. In these cases, value should be the value of the option selected, not the visible text.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

value - the value to type

typeKeys

void **typeKeys**(java.lang.String locator, java.lang.String value)

Simulates keystroke events on the specified element, as though you typed the value key-by-key.

This is a convenience method for calling keyDown, keyUp, keyPress for every character in the specified string; this is useful for dynamic UI widgets (like auto-completing combo boxes) that require explicit key events.

Unlike the simple "type" command, which forces the specified value into the page directly, this command may or may not have any visible effect, even in cases where typing keys would normally have a visible effect. For example, if you use "typeKeys" on a form element, you may or may not see the results of what you typed in the field.

In some cases, you may need to use the simple "type" command to set the value of the field and then the "typeKeys" command to send the keystroke events corresponding to what you just typed.

**Parameters**:

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

value - the value to type

dragdrop

void **dragdrop**(java.lang.String locator, java.lang.String movementsString)

deprecated - use dragAndDrop instead

**Parameters:**

locator - an element locator

movementsString - offset in pixels from the current location to which the element should be moved, e.g., "+70,-300"

setMouseSpeed

void **setMouseSpeed**(java.lang.String pixels)

Configure the number of pixels between "mousemove" events during dragAndDrop commands (default=10).

Setting this value to 0 means that a "mousemove" event is sent to every single pixel in between the start location and the end location. That can be very slow and may cause some browsers to force the JavaScript to timeout.

If the mouse speed is greater than the distance between the two dragged objects, we'll just send one "mousemove" at the start location and then one final one at the end location.

**Parameters:** pixels - the number of pixels between "mousemove" events

getMouseSpeed

java.lang.Number **getMouseSpeed**()

**Returns:** Number of pixels between "mousemove" events during dragAndDrop commands (default=10)

dragAndDrop

void **dragAndDrop**(java.lang.String locator, java.lang.String movementsString)

Drags an element a certain distance and then drops it

**Parameters:**

locator - an element locator

movementsString - offset in pixels from the current location to which the element should be moved, e.g., "+70,-300"

dragAndDropToObject

void **dragAndDropToObject**(java.lang.String locatorOfObjectToBeDragged,

java.lang.String locatorOfDragDestinationObject)

Drags an element and drops it on another element

**Parameters:**

locatorOfObjectToBeDragged - an element to be dragged

locatorOfDragDestinationObject - an element whose location (i.e., whose center-most pixel) will be the point where locatorOfObjectToBeDragged is dropped

### Checkbox and Select controls

check

void **check**(java.lang.String locator)

Check a toggle-button (checkbox/radio)

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

uncheck

void **uncheck**(java.lang.String locator)

Uncheck a toggle-button (checkbox/radio)

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

select

void **select**(java.lang.String selectLocator, java.lang.String optionLocator)

Select an option from a drop-down using an option locator.

Option locators provide different ways of specifying options of an HTML Select element (e.g. for selecting a specific option, or for asserting that the selected option satisfies a specification). There are several forms of Select Option Locator.

* label=labelPattern: matches options based on their labels, i.e. the visible text. (This is the default.)
  + label=regexp:^[Oo]ther
* value=valuePattern: matches options based on their values.
  + value=other
* id=id: matches options based on their ids.
  + id=option1
* index=index: matches an option based on its index (offset from zero).
  + index=2

If no option locator prefix is provided, the default behaviour is to match on **label**.

**Parameters:**

selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

optionLocator - an option locator (a label by default)

addSelection

void **addSelection**(java.lang.String locator, java.lang.String optionLocator)

Add a selection to the set of selected options in a multi-select element using an option locator.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a multi-select box

optionLocator - an option locator (a label by default)

removeSelection

void **removeSelection**

(java.lang.String locator, java.lang.String optionLocator)

Remove a selection from the set of selected options in a multi-select element using an option locator.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a multi-select box

optionLocator - an option locator (a label by default)

removeAllSelections

void **removeAllSelections**(java.lang.String locator)

Unselects all of the selected options in a multi-select element.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a multi-select box

### Misc

submit

void **submit**(java.lang.String formLocator)

Submit the specified form. This is particularly useful for forms without submit buttons, e.g. single-input "Search" forms.

**Parameters:** formLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) for the form you want to submit

highlight

void **highlight**(java.lang.String locator)

Briefly changes the backgroundColor of the specified element yellow. Useful for debugging.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

setBrowserLogLevel

void **setBrowserLogLevel**(java.lang.String logLevel)

Sets the threshold for browser-side logging messages; log messages beneath this threshold will be discarded. Valid logLevel strings are: "debug", "info", "warn", "error" or "off". To see the browser logs, you need to either show the log window in GUI mode, or enable browser-side logging in Selenium RC.

**Parameter:** logLevel - one of the following: "debug", "info", "warn", "error" or "off"

runScript

void **runScript**(java.lang.String script)

Creates a new "script" tag in the body of the current test window, and adds the specified text into the body of the command. Scripts run in this way can often be debugged more easily than scripts executed using Selenium's "getEval" command. Beware that JS exceptions thrown in these script tags aren't managed by Selenium, so you should probably wrap your script in try/catch blocks if there is any chance that the script will throw an exception.

**Parameter:** script - the JavaScript snippet to run

addLocationStrategy

void **addLocationStrategy**(java.lang.String strategyName,

java.lang.String functionDefinition)

Defines a new function for Selenium to locate elements on the page. For example, if you define the strategy "foo", and someone runs click("foo=blah"), we'll run your function, passing you the string "blah", and click on the element that your function returns, or throw an "Element not found" error if your function returns null. We'll pass three arguments to your function:

locator: the string the user passed in

inWindow: the currently selected window

inDocument: the currently selected document

The function must return null if the element can't be found.

**Parameters:**

strategyName - the name of the strategy to define; this should use only letters [a-zA-Z] with no spaces or other punctuation.

functionDefinition - a string defining the body of a function in JavaScript. For example: return inDocument.getElementById(locator);

rollup

void **rollup**(java.lang.String rollupName, java.lang.String kwargs)

Executes a command rollup, which is a series of commands with a unique name and optionally arguments that control the generation of the set of commands. If any one of the rolled-up commands fails, the rollup is considered to have failed. Rollups may also contain nested rollups.

**Parameters:**

rollupName - the name of the rollup command

kwargs - keyword arguments string that influences how the rollup expands into commands

addScript

void **addScript**(java.lang.String scriptContent, java.lang.String scriptTagId)

Loads script content into a new script tag in the Selenium document. This differs from the runScript command in that runScript adds the script tag to the document of the AUT, not the Selenium document. The following entities in the script content are replaced by the characters they represent: < > & The corresponding remove command is removeScript.

**Parameters:**

scriptContent - the Javascript content of the script to add

scriptTagId - (optional) the id of the new script tag. If specified, and an element with this id already exists, this operation will fail.

removeScript

void **removeScript**(java.lang.String scriptTagId)

Removes a script tag from the Selenium document identified by the given id. Does nothing if the referenced tag doesn't exist.

**Parameter:** scriptTagId - the id of the script element to remove.

useXpathLibrary

void **useXpathLibrary**(java.lang.String libraryName)

Allows choice of one of the available libraries.

**Parameter:** libraryName - name of the desired library Only the following three can be chosen: ajaxslt - Google's library javascript - Cybozu Labs' faster library default - The default library. Currently the default library is ajaxslt. If libraryName isn't one of these three, then no change will be made.

setContext

void **setContext**(java.lang.String context)

Writes a message to the status bar and adds a note to the browser-side log.

**Parameters:** context - the message to be sent to the browser

showContextualBanner

void **showContextualBanner**()

Shows in the RemoteRunner a banner for the current test The banner is 'classname : methodname' where those two are derived from the caller The method name will be unCamelCased with the insertion of spaces at word boundaries

showContextualBanner

void **showContextualBanner**(java.lang.String className,

java.lang.String methodName)

Shows in the RemoteRunner a banner for the current test. The banner is 'classname : methodname' The method name will be unCamelCased with the insertion of spaces at word boundaries

retrieveLastRemoteControlLogs

java.lang.String **retrieveLastRemoteControlLogs**()

Retrieve the last messages logged on a specific remote control. Useful for error reports, especially when running multiple remote controls in a distributed environment. The maximum number of log messages that can be retrieve is configured on remote control startup.

**Returns:** The last N log messages as a multi-line string.

assignId

void **assignId**(java.lang.String locator, java.lang.String identifier)

Temporarily sets the "id" attribute of the specified element, so you can locate it in the future using its ID rather than a slow/complicated XPath. This ID will disappear once the page is reloaded.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an element

identifier - a string to be used as the ID of the specified element

allowNativeXpath

void **allowNativeXpath**(java.lang.String allow)

Specifies whether Selenium should use the native in-browser implementation of XPath (if any native version is available); if you pass "false" to this function, we will always use our pure-JavaScript xpath library. Using the pure-JS xpath library can improve the consistency of xpath element locators between different browser vendors, but the pure-JS version is much slower than the native implementations.

**Parameter:** Allow - Boolean, true means prefer native XPath; false means only use JS XPath

ignoreAttributesWithoutValue

void **ignoreAttributesWithoutValue**(java.lang.String ignore)

Specifies whether Selenium will ignore xpath attributes that have no value, i.e. are the empty string, when using the non-native xpath evaluation engine. You'd want to do this for performance reasons in IE. However, this could break certain xpaths, for example an xpath that looks for an attribute whose value is NOT the empty string. The hope is that such xpaths are relatively rare, but the user should have the option of using them. Note that this only influences xpath evaluation when using the ajaxslt engine (i.e. not "JavaScript-xpath").

**Parameters:** Ignore - Boolean, true means we'll ignore attributes without value at the expense of xpath "correctness"; false means we'll sacrifice speed for correctness.

### Window, Frame & Pop-up handling commands

open

void **open**(java.lang.String url)

Opens a URL in the test frame; this accepts both relative and absolute URLs. The "open" command waits for the page to load before proceeding, i.e. the "AndWait" suffix is implicit. Note: The URL must be on the same domain as the runner HTML due to security restrictions in the browser (Same Origin Policy). If you need to open an URL on another domain, use the Selenium Server to start a new browser session on that domain.

**Parameters:** url - the URL to open; may be relative or absolute

openWindow

void **openWindow**(java.lang.String url, java.lang.String windowID)

Opens a popup window (if a window with that ID isn't already open). After opening the window, you need to select it using the selectWindow command.

This command can also be a useful workaround for bug SEL-339. In some cases, Selenium will be unable to intercept a call to window.open (if the call occurs during or before the "onLoad" event, for example). In those cases, you can force Selenium to notice the open window's name by using the Selenium openWindow command, using an empty (blank) url, like this: openWindow("", "myFunnyWindow").

**Parameters:**

url - the URL to open, which can be blank

windowID - the JavaScript window ID of the window to select

selectWindow

void **selectWindow**(java.lang.String windowID)

Selects a popup window using a window locator; once a popup window has been selected, all commands go to that window. To select the main window again, use null as the target.

Window locators provide different ways of specifying the window object: by title, by internal JavaScript "name," or by JavaScript variable.

**title**=My Special Window: Finds the window using the text that appears in the title bar. Be careful; two windows can share the same title. If that happens, this locator will just pick one.

**name**=myWindow: Finds the window using its internal JavaScript "name" property. This is the second parameter "windowName" passed to the JavaScript method window.open(url, windowName, windowFeatures, replaceFlag) (which Selenium intercepts).

**var**=variableName: Some pop-up windows are unnamed (anonymous), but are associated with a JavaScript variable name in the current application window, e.g. "window.foo = window.open(url);". In those cases, you can open the window using "var=foo".

If no window locator prefix is provided, we'll try to guess what you mean like this:

1.) if windowID is null, (or the string "null") then it is assumed the user is referring to the original window instantiated by the browser).

2.) if the value of the "windowID" parameter is a JavaScript variable name in the current application window, then it is assumed that this variable contains the return value from a call to the JavaScript window.open() method.

3.) Otherwise, selenium looks in a hash it maintains that maps string names to window "names".

4.) If that fails, we'll try looping over all of the known windows to try to find the appropriate "title". Since "title" is not necessarily unique, this may have unexpected behavior.

If you're having trouble figuring out the name of a window that you want to manipulate, look at the Selenium log messages which identify the names of windows created via window.open (and therefore intercepted by Selenium). You will see messages like the following for each window as it is opened:

debug: window.open call intercepted; window ID (which you can use with selectWindow()) is "myNewWindow"

In some cases, Selenium will be unable to intercept a call to window.open (if the call occurs during or before the "onLoad" event, for example). (This is bug SEL-339.) In those cases, you can force Selenium to notice the open window's name by using the Selenium openWindow command, using an empty (blank) url, like this: openWindow("", "myFunnyWindow").

**Parameters:** windowID - the JavaScript window ID of the window to select

windowFocus

void **windowFocus**()

Gives focus to the currently selected window

windowMaximize

void **windowMaximize**()

Resize currently selected window to take up the entire screen

getAllWindowIds

java.lang.String[] **getAllWindowIds**()

**Returns:** The IDs of all windows that the browser knows about.

getAllWindowNames

java.lang.String[] **getAllWindowNames**()

**Returns:** The names of all windows that the browser knows about.

getAllWindowTitles

java.lang.String[] **getAllWindowTitles**()

**Returns:** The titles of all windows that the browser knows about.

selectFrame

void **selectFrame**(java.lang.String locator)

Selects a frame within the current window. (You may invoke this command multiple times to select nested frames.) To select the parent frame, use "relative=parent" as a locator; to select the top frame, use "relative=top". You can also select a frame by its 0-based index number; select the first frame with "index=0", or the third frame with "index=2".

You may also use a DOM expression to identify the frame you want directly, like this: dom=frames["main"].frames["subframe"]

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a frame or iframe

getWhetherThisFrameMatchFrameExpression

boolean **getWhetherThisFrameMatchFrameExpression**

(java.lang.String currentFrameString, java.lang.String target)

Determine whether current/locator identify the frame containing this running code.

This is useful in proxy injection mode, where this code runs in every browser frame and window, and sometimes the selenium server needs to identify the "current" frame. In this case, when the test calls selectFrame, this routine is called for each frame to figure out which one has been selected. The selected frame will return true, while all others will return false.

**Parameters:**

currentFrameString - starting frame

target - new frame (which might be relative to the current one)

**Returns:** True if the new frame is this code's window

getWhetherThisWindowMatchWindowExpression

boolean **getWhetherThisWindowMatchWindowExpression**

(java.lang.String currentWindowString,java.lang.String target)

Determine whether currentWindowString plus target identify the window containing this running code.

This is useful in proxy injection mode, where this code runs in every browser frame and window, and sometimes the selenium server needs to identify the "current" window. In this case, when the test calls selectWindow, this routine is called for each window to figure out which one has been selected. The selected window will return true, while all others will return false.

**Parameters:**

currentWindowString - starting window

target - new window (which might be relative to the current one, e.g., "\_parent")

**Returns:** True if the new window is this code's window

waitForPopUp

void **waitForPopUp**(java.lang.String windowID, java.lang.String timeout)

Waits for a popup window to appear and load up.

**Parameters:**

windowID - the JavaScript window "name" of the window that will appear (not the text of the title bar)

timeout - a timeout in milliseconds, after which the action will return with an error

chooseCancelOnNextConfirmation

void chooseCancelOnNextConfirmation()

By default, Selenium's overridden window.confirm() function will return true, as if the user had manually clicked OK; after running this command, the next call to confirm() will return false, as if the user had clicked Cancel. Selenium will then resume using the default behavior for future confirmations, automatically returning true (OK) unless/until you explicitly call this command for each confirmation.

Take note - every time a confirmation comes up, you must consume it with a corresponding getConfirmation, or else the next selenium operation will fail.

chooseOkOnNextConfirmation

void **chooseOkOnNextConfirmation**()

Undo the effect of calling chooseCancelOnNextConfirmation. Note that Selenium's overridden window.confirm() function will normally automatically return true, as if the user had manually clicked OK, so you shouldn't need to use this command unless for some reason you need to change your mind prior to the next confirmation. After any confirmation, Selenium will resume using the default behavior for future confirmations, automatically returning true (OK) unless/until you explicitly call chooseCancelOnNextConfirmation for each confirmation.

Take note - every time a confirmation comes up, you must consume it with a corresponding getConfirmation, or else the next selenium operation will fail.

answerOnNextPrompt

void **answerOnNextPrompt**(java.lang.String answer)

Instructs Selenium to return the specified answer string in response to the next JavaScript prompt [window.prompt()].

**Parameters:** answer - the answer to give in response to the prompt pop-up

goBack

void **goBack**()

Simulate a user clicking the "back" button on their browser.

refresh

void **refresh**()

Simulate a user click of the "Refresh" button on their browser.

close

void **close**()

Simulates a user clicking the "close" button in the title bar of a popup window or tab.

isAlertPresent

boolean **isAlertPresent**()

Has an alert occurred? This function never throws an exception

**Returns:** True if there is an alert

isPromptPresent

boolean **isPromptPresent**()

Has a prompt occurred? This function never throws an exception

**Returns:** True if there is a pending prompt

isConfirmationPresent

boolean **isConfirmationPresent**()

Has JavaScript confirm() been called? This function never throws an exception

**Returns:** True if there is a pending confirmation

getAlert

java.lang.String **getAlert**()

Retrieves the message of a JavaScript alert generated during the previous action, or fail if there were no alerts.

Getting an alert has the same effect as manually clicking OK. If an alert is generated but you do not consume it with getAlert, the next Selenium action will fail.

Under Selenium, JavaScript alerts will NOT pop up a visible alert dialog.

Selenium does NOT support JavaScript alerts that are generated in a page's onload() event handler. In this case a visible dialog WILL be generated and Selenium will hang until someone manually clicks OK.

**Returns:** The message of the most recent JavaScript alert

getConfirmation

java.lang.String **getConfirmation**()

Retrieves the message of a JavaScript confirmation dialog generated during the previous action.

By default, the confirm function will return true, having the same effect as manually clicking OK. This can be changed by prior execution of the chooseCancelOnNextConfirmation command.

If an confirmation is generated but you do not consume it with getConfirmation, the next Selenium action will fail.

NOTE: under Selenium, JavaScript confirmations will NOT pop up a visible dialog.

NOTE: Selenium does NOT support JavaScript confirmations that are generated in a page's onload() event handler. In this case a visible dialog WILL be generated and Selenium will hang until you manually click OK.

**Returns:** The message of the most recent JavaScript confirmation dialog

getPrompt

java.lang.String **getPrompt**()

Retrieves the message of a JavaScript question prompt dialog generated during the previous action.

Successful handling of the prompt requires prior execution of the answerOnNextPrompt command. If a prompt is generated but you do not get/verify it, the next Selenium action will fail.

NOTE: under Selenium, JavaScript prompts will NOT pop up a visible dialog.

NOTE: Selenium does NOT support JavaScript prompts that are generated in a page's onload() event handler. In this case a visible dialog WILL be generated and Selenium will hang until someone manually clicks OK.

**Returns:** The message of the most recent JavaScript question prompt

### Get commands

getLocation

java.lang.String **getLocation**()

**Returns:** The absolute URL of the current page

getHtmlSource

java.lang.String **getHtmlSource**()

**Returns:** The entire HTML source between the opening and closing "html" tags.

getTitle

java.lang.String **getTitle**()

**Returns:** The title of the current page

getBodyText

java.lang.String **getBodyText**()

**Returns:** The entire text of the page

getValue

java.lang.String **getValue**(java.lang.String locator)

Gets the (whitespace-trimmed) value of an input field (or anything else with a value parameter). For checkbox/radio elements, the value will be "on" or "off" depending on whether the element is checked or not.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

**Returns:** The element value, or "on/off" for checkbox/radio elements

getText

java.lang.String **getText**(java.lang.String locator)

Gets the text of an element. This works for any element that contains text. This command uses either the textContent (Mozilla-like browsers) or the innerText (IE-like browsers) of the element, which is the rendered text shown to the user.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

**Returns:** The text of the element

getEval

java.lang.String **getEval**(java.lang.String script)

Gets the result of evaluating the specified JavaScript snippet. The snippet may have multiple lines, but only the result of the last line will be returned.

Note that, by default, the snippet will run in the context of the "selenium" object itself, so this will refer to the Selenium object. Use window to refer to the window of your application, e.g. window.document.getElementById('foo')

If you need to use a locator to refer to a single element in your application page, you can use this.browserbot.findElement("id=foo") where "id=foo" is your locator.

**Parameters:** script - the JavaScript snippet to run

**Returns:** The results of evaluating the snippet

getTable

java.lang.String **getTable**(java.lang.String tableCellAddress)

Gets the text from a cell of a table. The cellAddress syntax tableLocator.row.column, where row and column start at 0.

**Parameters:** tableCellAddress - a cell address, e.g. "foo.1.4"

**Returns:** The text from the specified cell

getSelectedLabels

java.lang.String[] **getSelectedLabels**(java.lang.String selectLocator)

Gets all option labels (visible text) for selected options in the specified select or multi-select element.

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** An array of all selected option labels in the specified select drop-down

getSelectedLabel

java.lang.String **getSelectedLabel**(java.lang.String selectLocator)

Gets option label (visible text) for selected option in the specified select element.

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** The selected option label in the specified select drop-down

getSelectedValues

java.lang.String[] **getSelectedValues**(java.lang.String selectLocator)

Gets all option values (value attributes) for selected options in the specified select or multi-select element.

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** An array of all selected option values in the specified select drop-down

getSelectedValue

java.lang.String **getSelectedValue**(java.lang.String selectLocator)

Gets option value (value attribute) for selected option in the specified select element.

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** The selected option value in the specified select drop-down

getSelectedIndexes

java.lang.String[] **getSelectedIndexes**(java.lang.String selectLocator)

Gets all option indexes (option number, starting at 0) for selected options in the specified select or multi-select element.

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** An array of all selected option indexes in the specified select drop-down

getSelectedIndex

java.lang.String **getSelectedIndex**(java.lang.String selectLocator)

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** The selected option index (option number, starting at 0) in the specified select drop-down

getSelectedIds

java.lang.String[] **getSelectedIds**(java.lang.String selectLocator)

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** An array of all selected option IDs in the specified select drop-down or multi-select

getSelectedId

java.lang.String **getSelectedId**(java.lang.String selectLocator)

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** The selected option ID in the specified select drop-down

getSelectOptions

java.lang.String[] **getSelectOptions**(java.lang.String selectLocator)

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** An array of all option labels in the specified select drop-down

getAttribute

java.lang.String **getAttribute**(java.lang.String attributeLocator)

**Parameter:** attributeLocator - an element locator followed by an @ sign and then the name of the attribute, e.g. "foo@bar"

**Returns:** The value of an element attribute. The value of the attribute may differ across browsers (this is the case for the "style" attribute, for example).

getAllButtons

java.lang.String[] **getAllButtons**()

**Returns:** A string array of IDs for all buttons on the page. If a given button has no ID, it will appear as "" in this array.

getAllLinks

java.lang.String[] **getAllLinks**()

**Returns:** The IDs of all links on the page. If a given link has no ID, it will appear as "" in this array.

getAllFields

java.lang.String[] **getAllFields**()

**Returns:** The IDs of all input fields on the page . If a given field has no ID, it will appear as "" in this array

getAttributeFromAllWindows

java.lang.String[] **getAttributeFromAllWindows**(java.lang.String attributeName)

**Parameters:** attributeName - name of an attribute on the windows

**Returns:** The set of values of this attribute from all known windows.

getElementIndex

java.lang.Number **getElementIndex**(java.lang.String locator)

Get the relative index of an element to its parent (starting from 0). The comment node and empty text node will be ignored.

**Parameter:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an element

**Returns:** Number of relative index of the element to its parent (starting from 0)

getExpression

java.lang.String **getExpression**(java.lang.String expression)

Returns the specified expression. This is useful because of JavaScript preprocessing. It is used to generate commands like assertExpression and waitForExpression.

**Parameters:** expression - the value to return

**Returns:** The value passed in

getXpathCount

java.lang.Number **getXpathCount**(java.lang.String xpath)

Returns the number of nodes that match the specified xpath, e.g. "//table" would give the number of tables.

**Parameter:** xpath - the xpath expression to evaluate. Do NOT wrap this expression in a 'count()' function; we will do that for you.

**Returns:** The number of nodes that match the specified xpath

### Is Queries

isChecked

boolean **isChecked**(java.lang.String locator)

Gets whether a toggle-button (checkbox/radio) is checked. Fails if the specified element doesn't exist or isn't a toggle-button.

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to a checkbox or radio button

**Returns:** True if the checkbox is checked, false otherwise

isTextPresent

boolean **isTextPresent**(java.lang.String pattern)

Verifies that the specified text pattern appears somewhere on the rendered page shown to the user.

**Parameter:** pattern - a [pattern](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#patterns) to match with the text of the page

**Returns:** True if the pattern matches the text, false otherwise

isElementPresent

boolean **isElementPresent**(java.lang.String locator)

Verifies that the specified element is somewhere on the page.

**Parameter:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

**Returns:** True if the element is present, false otherwise

isVisible

boolean **isVisible**(java.lang.String locator)

Determines if the specified element is visible. An element can be rendered invisible by setting the CSS "visibility" property to "hidden", or the "display" property to "none", either for the element itself or one if its ancestors. This method will fail if the element is not present.

**Parameter:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

**Returns:** True if the specified element is visible, false otherwise

isEditable

boolean **isEditable**(java.lang.String locator)

Determines whether the specified input element is editable, ie hasn't been disabled. This method will fail if the specified element isn't an input element.

**Parameter:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators)

**Returns:** True if the input element is editable, false otherwise

isSomethingSelected

boolean **isSomethingSelected**(java.lang.String selectLocator)

Determines whether some option in a drop-down menu is selected.

**Parameter:** selectLocator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) identifying a drop-down menu

**Returns:** True if some option has been selected, false otherwise

isOrdered

boolean **isOrdered**(java.lang.String locator1, java.lang.String locator2)

Check if these two elements have same parent and are ordered siblings in the DOM. Two same elements will not be considered ordered.

**Parameters:**

locator1 - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to the first element

locator2 - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to the second element

**Returns:** True if element1 is the previous sibling of element2, false otherwise

### Element Position and Sizing

setCursorPosition

void **setCursorPosition**(java.lang.String locator, java.lang.String position)

Moves the text cursor to the specified position in the given input element or textarea. This method will fail if the specified element isn't an input element or textarea.

**Parameters:**

locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an input element or textarea

position - the numerical position of the cursor in the field; position should be 0 to move the position to the beginning of the field. You can also set the cursor to -1 to move it to the end of the field.

getElementPositionLeft

java.lang.Number **getElementPositionLeft**(java.lang.String locator)

Retrieves the horizontal position of an element

**Parameter:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an element OR an element itself

**Returns:** Number of pixels from the edge of the frame.

getElementPositionTop

java.lang.Number **getElementPositionTop**(java.lang.String locator)

Retrieves the vertical position of an element

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an element OR an element itself

**Returns:** Number of pixels from the edge of the frame.

getElementWidth

java.lang.Number **getElementWidth**(java.lang.String locator)

Retrieves the width of an element

**Parameters:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an element

**Returns:** Width of an element in pixels

getElementHeight

java.lang.Number **getElementHeight**(java.lang.String locator)

Retrieves the height of an element

**Parameter:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an element

**Returns:** Height of an element in pixels

getCursorPosition

java.lang.Number **getCursorPosition**(java.lang.String locator)

Retrieves the text cursor position in the given input element or textarea; beware, this may not work perfectly on all browsers.

Specifically, if the cursor/selection has been cleared by JavaScript, this command will tend to return the position of the last location of the cursor, even though the cursor is now gone from the page. This is filed as [SEL-243](http://jira.openqa.org/browse/SEL-243).

This method will fail if the specified element isn't an input element or textarea, or there is no cursor in the element.

**Parameter:** locator - an [element locator](http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html#locators) pointing to an input element or textarea

**Returns:** The numerical position of the cursor in the field

### Waits and Timeouts

waitForCondition

void **waitForCondition**(java.lang.String script, java.lang.String timeout)

Runs the specified JavaScript snippet repeatedly until it evaluates to "true". The snippet may have multiple lines, but only the result of the last line will be considered.

Note that, by default, the snippet will be run in the runner's test window, not in the window of your application. To get the window of your application, you can use the JavaScript snippet selenium.browserbot.getCurrentWindow(), and then run your JavaScript in there

**Parameters:**

script - the JavaScript snippet to run

timeout - a timeout in milliseconds, after which this command will return with an error

setTimeout

void **setTimeout**(java.lang.String timeout)

Specifies the amount of time that Selenium will wait for actions to complete. Actions that require waiting include "open" and the "waitFor\*" actions. The default timeout is 30 seconds.

**Parameter:** timeout - a timeout in milliseconds, after which the action will return with an error

waitForPageToLoad

void **waitForPageToLoad**(java.lang.String timeout)

Waits for a new page to load. You can use this command instead of the "AndWait" suffixes, "clickAndWait", "selectAndWait", "typeAndWait" etc. (which are only available in the JS API).

Selenium constantly keeps track of new pages loading, and sets a "newPageLoaded" flag when it first notices a page load. Running any other Selenium command after turns the flag to false. Hence, if you want to wait for a page to load, you must wait immediately after a Selenium command that caused a page-load.

**Parameter:** timeout - a timeout in milliseconds, after which this command will return with an error

waitForFrameToLoad

void **waitForFrameToLoad**

(java.lang.String frameAddress, java.lang.String timeout)

Waits for a new frame to load.

Selenium constantly keeps track of new pages and frames loading, and sets a "newPageLoaded" flag when it first notices a page load. See waitForPageToLoad for more information.

**Parameters:**

frameAddress - FrameAddress from the server side

timeout - a timeout in milliseconds, after which this command will return with an error

### Cookie handling

getCookie

java.lang.String **getCookie**()

Return all cookies of the current page under test.

**Returns:** All cookies of the current page under test

getCookieByName

java.lang.String **getCookieByName**(java.lang.String name)

Returns the value of the cookie with the specified name, or throws an error if the cookie is not present.

**Parameter:** name - the name of the cookie

**Returns:** The value of the cookie

isCookiePresent

boolean **isCookiePresent**(java.lang.String name)

Returns true if a cookie with the specified name is present, or false otherwise.

**Parameter:** name - the name of the cookie

**Returns:** True if a cookie with the specified name is present, or false otherwise.

createCookie

void **createCookie**

(java.lang.String nameValuePair, java.lang.String optionsString)

Create a new cookie whose path and domain are the same as those of the current page under test, unless you specified a path for this cookie explicitly.

**Parameters:**

nameValuePair - name and value of the cookie in a format "name=value"

optionsString - options for the cookie. Currently supported options include 'path', 'max\_age' and 'domain'. the optionsString's format is "path=/path/, max\_age=60, domain=.foo.com". The order of options are irrelevant, the unit of the value of 'max\_age' is second. Note that specifying a domain that isn't a subset of the current domain will usually fail.

deleteCookie

void **deleteCookie**(java.lang.String name, java.lang.String optionsString)

Delete a named cookie with specified path and domain. Be careful; to delete a cookie, you need to delete it using the exact same path and domain that were used to create the cookie. If the path is wrong, or the domain is wrong, the cookie simply won't be deleted. Also note that specifying a domain that isn't a subset of the current domain will usually fail. Since there's no way to discover at runtime the original path and domain of a given cookie, we've added an option called 'recurse' to try all sub-domains of the current domain with all paths that are a subset of the current path. Beware; this option can be slow. In big-O notation, it operates in O(n\*m) time, where n is the number of dots in the domain name and m is the number of slashes in the path.

**Parameters:**

name - the name of the cookie to be deleted

optionsString - options for the cookie. Currently supported options include 'path', 'domain' and 'recurse.' The optionsString's format is "path=/path/, domain=.foo.com, recurse=true". The order of options are irrelevant. Note that specifying a domain that isn't a subset of the current domain will usually fail.

deleteAllVisibleCookies

void **deleteAllVisibleCookies**()

Calls deleteCookie with recurse=true on all cookies visible to the current page. As noted on the documentation for deleteCookie, recurse=true can be much slower than simply deleting the cookies using a known domain/path.