

# Selenium UI-Element & UI-Map



+



# Part 1: An Introduction to Selenium

- Introduction
- Selenium tools
- Using selenium
- Quick example
- Selenium benefits
- Example case
- Selenium Drawbacks

# Part 2: Selenium UI-map

- Selenium extentions
- UI-element & UI-map
  - UI-element Setup
  - UI-element summary
  - UI-map summary
  - UI-element terminology
  - UI-map setup
  - add UI-elements
  - UI-arguments modifying the XPATH
  - UI-arguments properties
  - testcases without UI-element
  - testcases with UI-element
  - rollups
  - writing a rollup
  - UI-elements and UI-maps conclusion
  - a list apart ui-map example
  - Our UI-map
  - Links

# Part 1

# Introduction

- Testers and developers can record and or write manual tests in a lightweight language, and play these back. great for regression testing.
- Selenium has four tools;
  - These can be used in isolation or combination to create an full testing suite



# Selenium tools

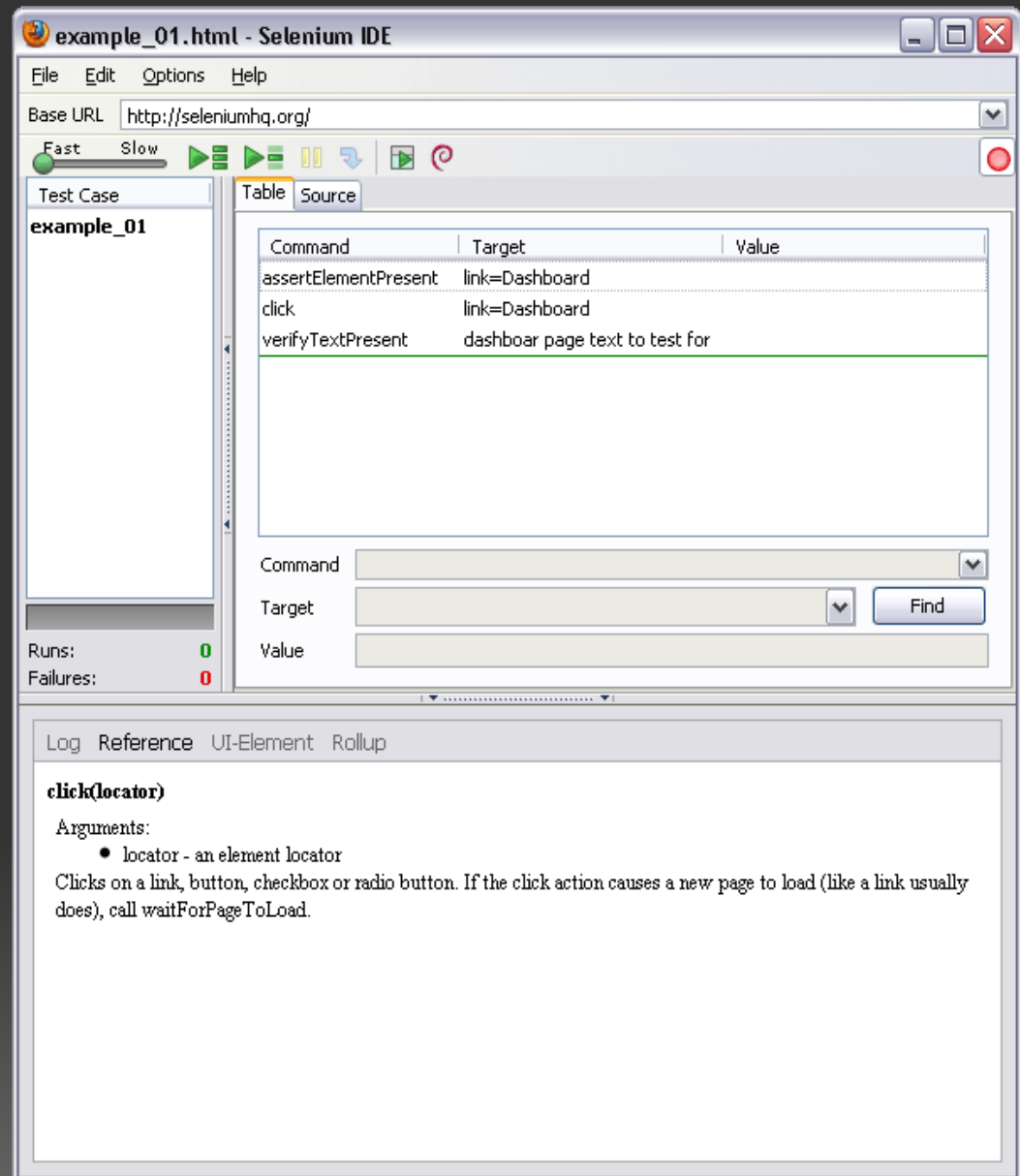
- IDE
- Core/TestRunner
- RC
- Grid

# IDE



## Firefox add-on

- Write or record tests
- Add asserts or verifies to test elements and text
- Controls section
- Reference section
- Tests, and suites of tests
- Tests can also be exported in a range of languages for RC and Grid Tools e.g.
- Ruby, Java, Python, C#, PHP, ect.



The screenshot shows the Selenium IDE interface for a test case named 'example\_01'. The interface includes a menu bar (File, Edit, Options, Help), a Base URL field (http://seleniumhq.org/), and a speed control slider (Fast to Slow). The main area is divided into 'Table' and 'Source' tabs. The 'Table' tab displays a table of test steps:

Command	Target	Value
assertElementPresent	link=Dashboard	
click	link=Dashboard	
verifyTextPresent	dashboar page text to test for	

Below the table, there are input fields for 'Command', 'Target', and 'Value', along with a 'Find' button. The bottom section of the interface shows a 'Log' tab with the following content:

**click(locator)**  
Arguments:

- locator - an element 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.

# CORE / TESTRUNNER

- A set of Selenium core files which can also run TESTRUNNER in the browser.
- TESTRUNNER Original method for running Selenium commands.
- Can be run in other browsers
- Similar to Selenium-IDE, it does not support running other programming languages.



# CORE / TESTRUNNER

## Selenium

[Selenium TestRunner](#) - Select a test suite to run in Selenium

**Acceptance tests:** These test-suites demonstrate/exercise the functionality of Selenium.

- [Selenium TestSuite](#) - functional tests for Selenium. This suite of tests should pass in any supported browser.
- [Error Checking TestSuite](#) - tests for the error verification commands

**Unit-tests:** Use JUnit to test Selenium internals.

- [Selenium BrowserBot unit-tests](#)
- [JsMock unit-tests](#)

## CMS<sup>2</sup>

[auto]	<a href="#">CMS<sup>2</sup> individual suite</a>
[auto]	<a href="#">CMS<sup>2</sup> modular smoke suite</a>
[auto]	<a href="#">CMS<sup>2</sup> modular smoke suite old</a>
[auto]	<a href="#">CMS<sup>2</sup> rte select all text</a>
[auto]	<a href="#">CMS<sup>2</sup> sample suite</a>
[auto]	<a href="#">CMS<sup>2</sup> widget suite</a>

The screenshot displays the Selenium Test Runner interface. At the top, it says "cms<sup>2</sup> TEST RUNNER" with links for "Back to suites" and "Selenium help".

**Test Suite:** A table listing test cases under the heading "CMS<sup>2</sup> - Modular Smoke Tests [back]".

CMS <sup>2</sup> - Modular Smoke Tests [back]	
Main navigation in good to know	
Dashboard update simple	
Dashboard update full	
Dashboard article preview	

**Current Test:** A table showing the details of the selected test case.

main_navigation_site_goodtoknow	
rollup	login
rollup	navigate
rollup	navigate
rollup	navigate
rollup	navigate
rollup	navigate
rollup	navigate
clickAndWait	ui=allPages::loginOut()

**Control Panel:** A panel for executing tests, including a speed slider (Fast to Slow), a "Highlight elements" checkbox, and a status section.

Execute Tests: [Run] [Pause] [Stop] [Refresh]

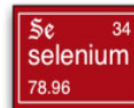
Elapsed: 00.00

Tests	Commands
0 run	0 passed
0 failed	0 failed
	0 incomplete

Tools: [View DOM] [Show Log]

Labels: Test Suite, Current Test, Control Panel

Footer: | AUT: Application Under Test |

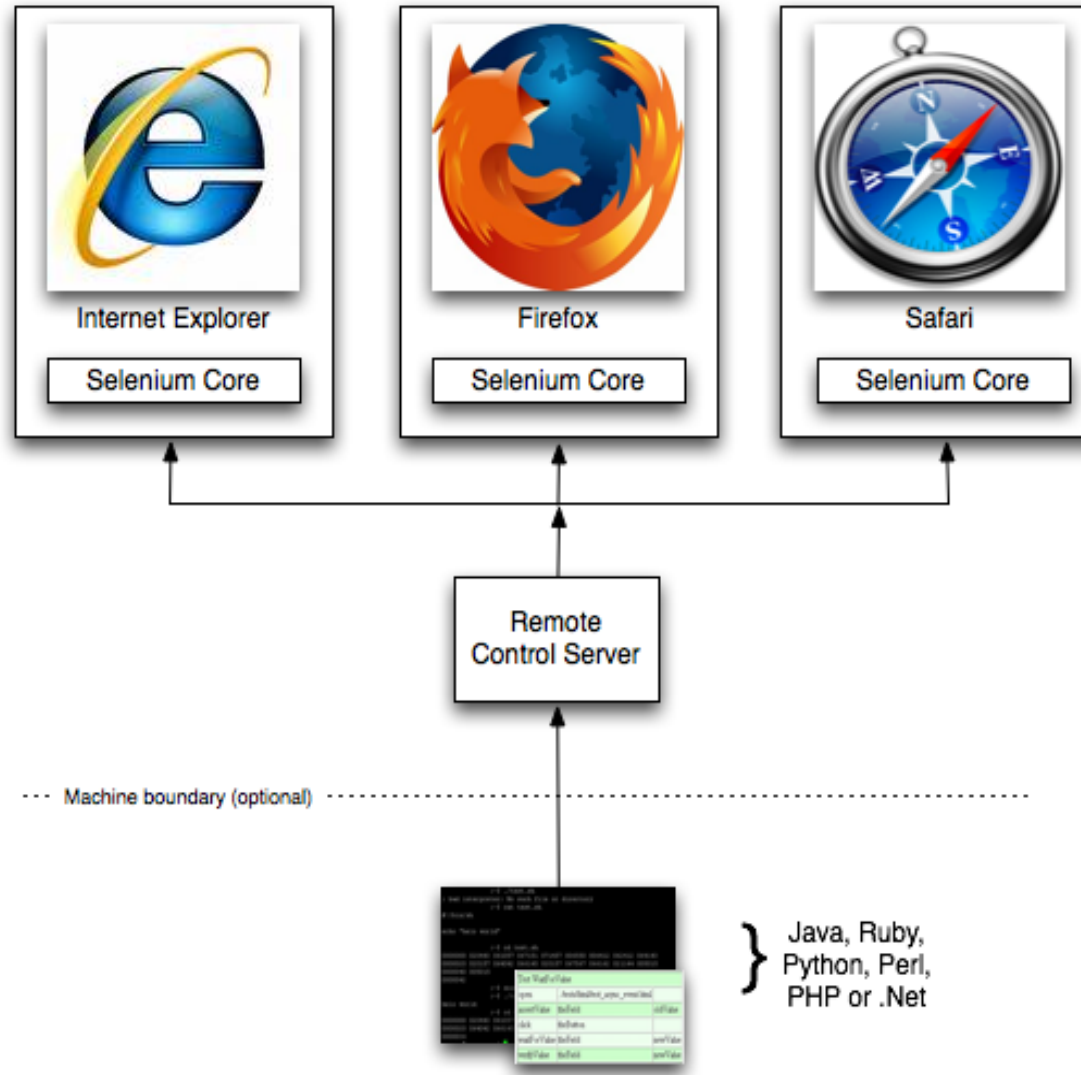


### Selenium Test Runner

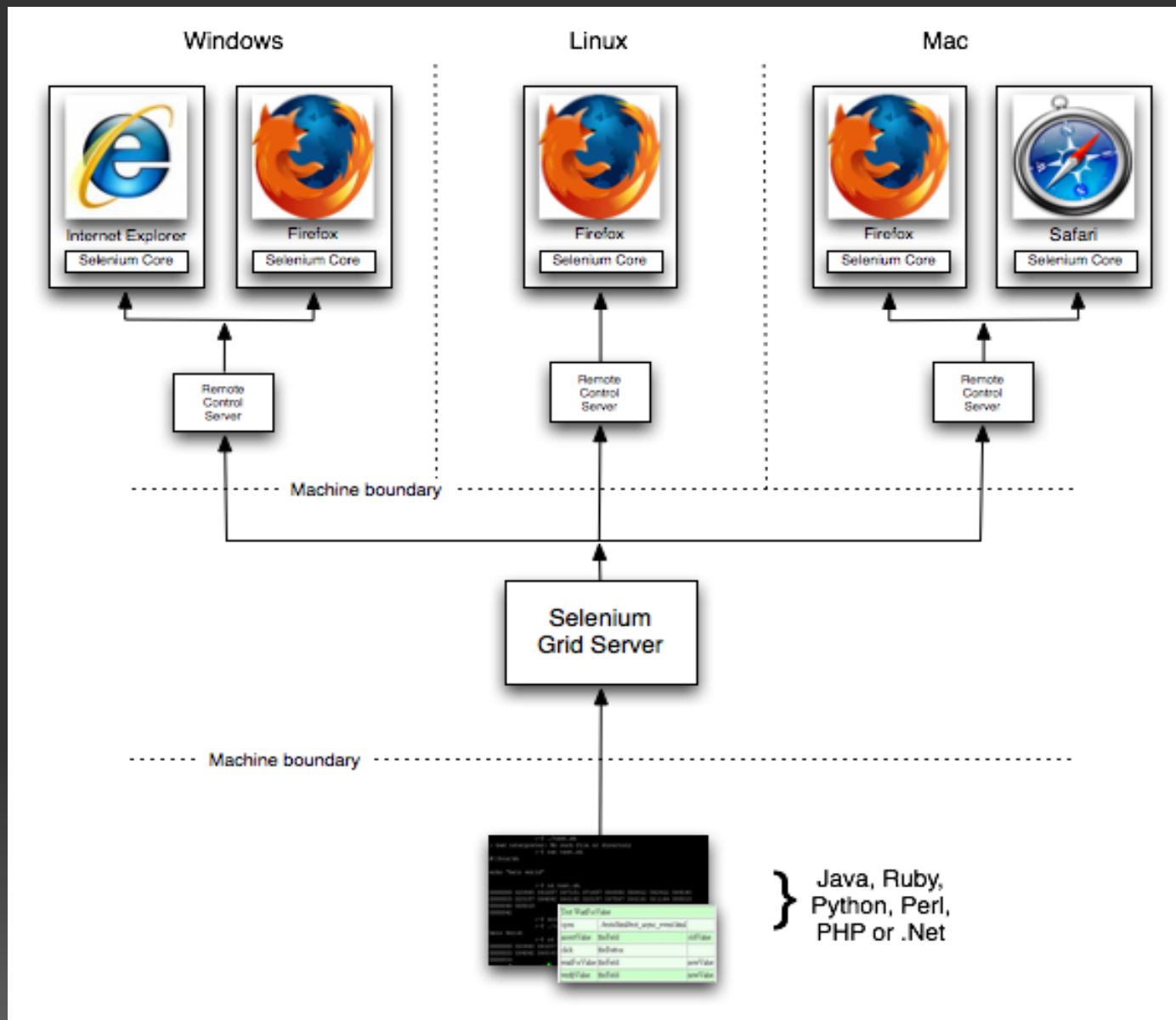
Holding page, would be application under test when test suite/case is loaded by [ThoughtWorks](#) and friends  
For more information on Selenium, visit <http://selenium.openga.org>

# RC

Windows, Linux, or Mac (as appropriate)...



# GRID



# Using Selenium

- Test syntax
- Test structure
- Test suites

# Test syntax

When using selenium to access/use items on the page like a textbox, we need to write out or record in the IDE **commands**.

These commands essentially create a testing language to tell Selenium what to do. these commands are often called ***selenese***.

## Command

Selenium commands are simple, they consist of the **command parameter** and **two optional pa**

Command	Target	Value
assertElementPresent	link=Dashboard	
click	link=Dashboard	
verifyTextPresent	dashboar page text to test for	
pause		5000

Command	<input type="text"/>	▼
Target	<input type="text"/>	▼ <input type="button" value="Find"/>
Value	<input type="text"/>	

## Optional Parameters syntax

these can be one of the following;

- a **locator** for identifying a UI element within a page.
  - **XPATH:** `//img[@id='button']`
  - **DOM:** `dom=document.images[0]`
  - **Link:** `link=click me`
  - **CSS:** `css=a#button`
  - **ID:** `button`
- a **text pattern** for **verifying** or **asserting** expected page content
- a **text pattern** or a selenium variable for **entering text** in an input field or for **selecting an option** from an option list.

# Tests: structure

## View in IDE

Command	Target	Value
open	../tests/html/Frames.html	
selectFrame	bottomFrame	
click	changeBlank	
waitForPopUp	_blank	10000
selectWindow	_blank	
click	changeSpan	
close		
selectWindow	null	
click	changeBlank	
waitForPopUp	_blank	10000
selectWindow	_blank	
click	changeSpan	
close		
selectWindow	null	
selectFrame	bottomFrame	
submit	formBlank	
waitForPopUp	_blank	10000
selectWindow	_blank	
click	changeSpan	
close		
selectWindow	null	
open	../tests/html/test_select_window...	
click	popupBlank	
waitForPopUp	_blank	10000
selectWindow	_blank	
verifyTitle	Select Window Popup	

## HTML code

```
<head>
  <meta content="text/html; charset=ISO-8859-1"
  http-equiv="content-type">
  <title>TestClickBlankTarget</title>
</head>
<body>
  <p>This test reproduces bug SEL-272</p>

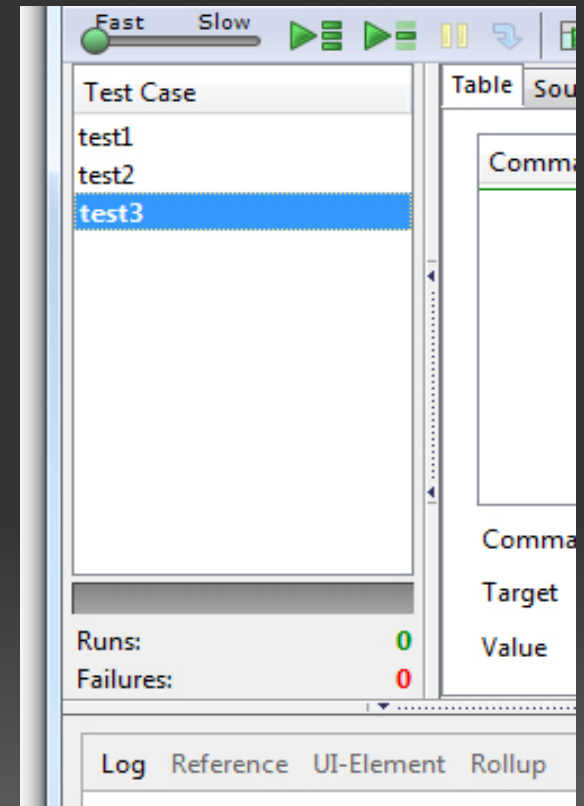
  <table cellpadding="1" cellspacing="1" border="1">
    <tbody>
      <tr>
        <td rowspan="1" colspan="3">TestClickBlankTarget<br>
        </td>
      </tr>
      <tr>
        <td>open</td>
        <td>../tests/html/Frames.html</td>
        <td></td>
      </tr>
      <tr>
        <td>selectFrame</td>
        <td>bottomFrame</td>
        <td></td>
      </tr>
      <tr>
        <td>click</td>
        <td>changeBlank</td>
        <td></td>
      </tr>
      <tr>
        <td>waitForPopUp</td>
        <td>_blank</td>
        <td>10000</td>
      </tr>
    </tbody>
  </table>
```

# Test suites

A Test suite is a collection of tests. This allows us to run all the tests in a test suite as one continuous batch-job:

Allowing us to run all the tests for a section of the UI for instance.

```
<html>
  <head>
    <title>Suite Function Tests</title>
  </head>
  <body>
    <table>
      <tr><td><b>Suite Of Tests</b></td></tr>
      <tr><td><a href="./test1.html">Test 1</a></td></tr>
      <tr><td><a href="./test2.html">Test 2</a></td></tr>
      <tr><td><a href="./test3.html">Test 3</a></td></tr>
    </table>
  </body>
</html>
```



# Quick example

## A Selenium Test

I'll just open a Firefox browser with selenium IDE add-on installed....

- Basic buttons
- Writing and recording tests
- Playing back a test
- Test cases & suites

## A Selenium Test suite

....and quickly I'll show a test suite in selenium CORE TESTRUNNER.....

# Selenium benefits

- Free to use
- Range of tools for automated testing
- Lightweight selenese language, easy to pick up
- Can record testers actions while manually testing the interface
- Can use test suites to create batches of tests
- Can use different programming languages to run tests in RC, and GRID, to utilise programmatic logic i.e. loops.

# Example case

## locators

Before we move on to drawbacks, first to highlight an example case of selenium locators;

As we saw with locators they have specific XPATH, or DOM, ect to reach a page element.  
For the current DOM within a web page;

```
<div id="example">  
  <a href="about.php">click me</a>  
</div>
```

we might wright/record a selenium locator in XPATH;

XPATH:        //div[@id='example']/a[contains(@href, 'about.php')]

Command	Target	Value
click	//div[@id='example']/a[contains(@href, 'about.php')]	

For many tests we might in each test write out this locator to access the <a> tag...

Test 1

click

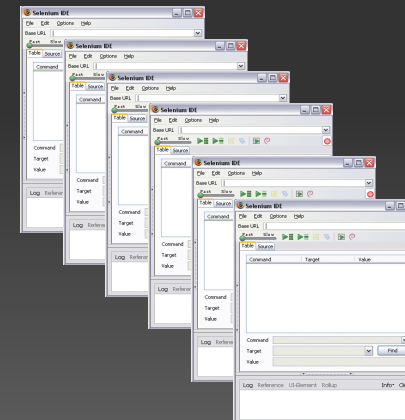
```
//div[@id='example']/a[contains(@href, 'about.php')]
```

Test 2

waitForElementPresent

```
//div[@id='example']/a[contains(@href, 'about.php')]
```

Ect, different uses, but each XPATH written out individually.



Although if the UI changes to

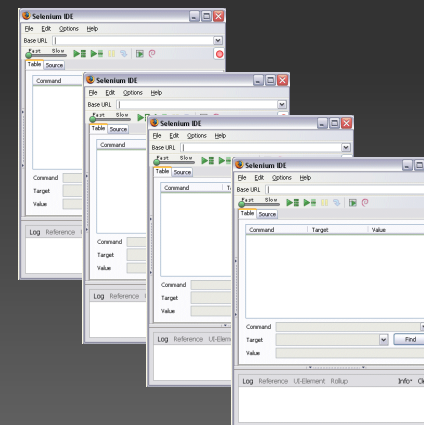
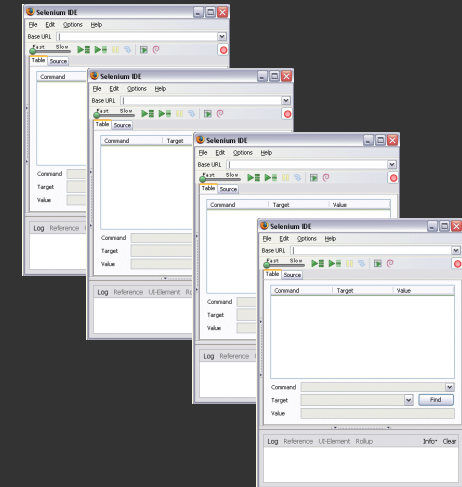
```
<div id="updated_example">  
  <a href="about.php">click me</a>  
</div>
```

in each test we need to change the locator to

```
//div[@index='updated_example']/a[contains(@href, 'about.php')]
```

This can become very laborious if you have many saved test i.e. a suite of 10 test and maybe 10 locators per test using this id with different commands.

That's 100 changes, to keep a 1 test suite of 10 test up to date.



# Selenium Draw backs

- No conditional statements in selenium IDE, although developers can use a programming languages like PHP and the selenium tools RC/GRID to allow this.
- Developers/testers having to write out long Locators to elements.
- Locators have to be written each time, little Locator code re-use
- Can mean it's very slow to write tests
- Tests can often get out of date when the UI changes.

# Part 2: Selenium UI-map

- Selenium extentions
- UI-element & UI-map
  - UI-element Setup
  - UI-element summary
  - UI-map summary
  - UI-element terminology
  - UI-map setup
  - add UI-elements
  - UI-arguments modifying the XPATH
  - UI-arguments properties
  - testcases without UI-element
  - testcases with UI-element
  - rollups
  - writing a rollup
  - UI-elements and UI-maps conclusion
  - a list apart ui-map example
  - Our UI-map
  - Links

# Selenium extensions

To help us with some of these drawbacks there are a wide range of extensions that we can use in Selenium.

These are available to download ( see links page ) and, or developers can write new extensions.

For instance, we can use the main selenium extensions file, called user-extensions.js for adding new selenese commands.

for example

```
Selenium.prototype.doEval = function(script) {  
  try {  
    return eval(script);  
  } catch (e) {  
    throw new SeleniumError("Threw an exception: " + e.message);  
  }  
};
```

Recently there's been another extension file we can use called **UI-element**, with which you can use a map of UI-elements called a **UI-map**.

UI-element has been widely used and is now been built into Selenium IDE tool, and is bundled with Selenium CORE.

The rest of this presentation aims to discuss this new extensions...

# UI-element & UI-map

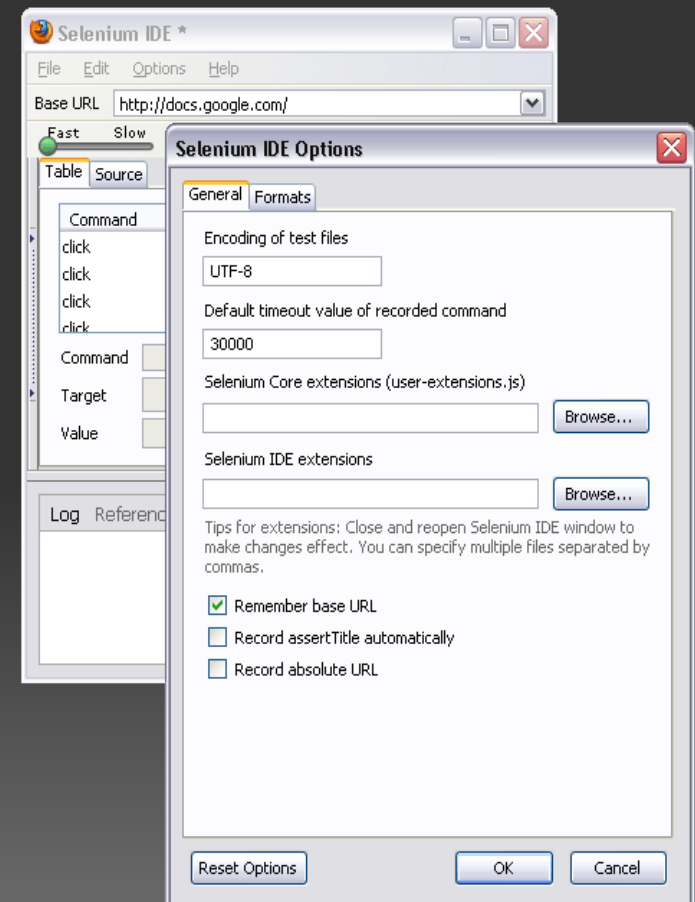
# Setup

**UI-maps** can be used by selenium **IDE** by adding them as user extensions in the options menu..

in Selenium core extentions feild browse for and add;  
**Z:\cms-trunk\web\testrunner\core\scripts\user-extensions.js**

then repeat the following for;

**Z:\cms-trunk\web\testrunner\core\scripts\ui-map\ui-map.js**  
**Z:\cms-trunk\web\testrunner\core\scripts\ui-map\ui-map-articles.js**  
**Z:\cms-trunk\web\testrunner\core\scripts\ui-map\ui-map-dashboard.js**  
**Z:\cms-trunk\web\testrunner\core\scripts\ui-map\ui-map-rte.js**



# UI-element Setup

....And by selenium CORE/ TESTRUNNER by adding the same files in the top includes of the testrunner.html.....

....And for the future could be added to selenium RC by incorporating all the files into the user-extensions.js file and running;

```
cd selenium-remote-control-folder
```

```
cd server
```

```
java -jar selenium-server.jar -userExtensions user-extensions.js
```

Although there are more methods to achieve and this I've added some links at the end for developers.

# UI-Element summary

A UI-Element is a;

user defined meaningful name representing web page elements.

or shortcuts to elements,  
a mapping of a name to a xpath locator;

- Easier for non-developers to write tests.
- Saves time when writing tests,
- Removes the need to write very much Code to access each element,
- Promotes code re-use,
- Adds the ability to 'roll-up' shortcuts into bunches of commands, called with one line.

Developers can write UI-element mappings using JavaScript Object Notation (**JSON**).

# UI-map summary

Is a collection of UI-Element shortcuts.

Allowing developers to create a map for the interface and application across pages;

These mappings are stored in a file called **UI-map.js** .

This file may can be shared by the Selenium IDE and CORE TESTRUNNER, and RC/GRID so It also offers a single point of update should the user interface of the AUT change.

This map allows developer and testers re-use of shortcuts to write tests quickly, and removes the need for them to re-write XPATH mappings.



# UI-Element terminology

## Page

a web page: an unique URL, and the contents available by accessing that URL.

Also considered as a Document Object Model (DOM) document object.

## Page element

An element on the web page. An element is anything the user might interact with, or anything that contains meaningful content.

More specifically, a DOM node and its contents.

## Pageset

A set of pages that share some set of common page elements.

For example, I might be able to log into my application from several different pages. If certain page elements on each of those pages appear similarly (i.e. their DOM representations are identical), those pages could be grouped into a pageset with respect to these page elements.

There is no restriction on how many pagesets a given page can be a member of, also a UI element can belong to multiple pagesets.

A pageset is commonly represented by a regular expression which matches the URL's that uniquely identify pages; however, there are cases when the page content must be considered to determine pageset membership.

## UI element

A mapping between a meaningful name for a page element, and the means to locate that page element's DOM node. The page element is located via a locator. UI elements belong to pagesets.

## UI argument

An optional piece of logic that determines how the locator is generated by a UI element. Used most when similar page elements appear multiple times on the same page, and you want to use a single UI element.

For example, if a page presents 20 clickable search results, the index of the search result might be a UI argument.

## UI map

A collection of pagesets, which in turn contain UI elements.

## Rollup rule

Selenium commands can be grouped into a single command. The single command may be expanded into its component Selenium commands. The single command is referred to as a "rollup".

## Command matcher

Typically folded into a rollup rule, it matches one or more Selenium commands and optionally sets values for rollup arguments based on the matched commands. A rollup rule usually has one or more command matcher's.

## Rollup argument

An optional piece of logic that modifies the command expansion of a rollup.

# UI-Map setup

This is the general format of a map file:

The map object is initialized by creating a new **UIMap** object.

```
var map = new UIMap();
```

Next, a pageset is defined.

```
map.addPageset({  
  name: 'aPageset'  
});
```

More page sets can be defined, and page sets can have descriptions and be for specific URLs ...

```
map.addPageset({  
  name: 'allPages',  
});  
map.addPageset({  
  name: 'articlePages',  
  description: 'all Articles pages',  
  pathRegexp: '(Articles/.+)?'  
});
```

# Adding UI-elements

Then one or more UI elements are defined for a pageset.

```
map.addElement('allPages', { ... });  
map.addElement('allPages', { ... });  
...
```

The format of a UI-element in its simplest form is, using just the required properties;

- Name: The name you can use to call your ui-element
- Description: The main body of the description this will show up in the selenium-IDE reference
- Locator: The stored locator from your test, xpath, dom, ect

```
map.addElement('allPages', {  
  name: 'about_link',  
  description: 'link to the about page',  
  locator: "//a[contains(@href, 'about.php')]"  
});
```

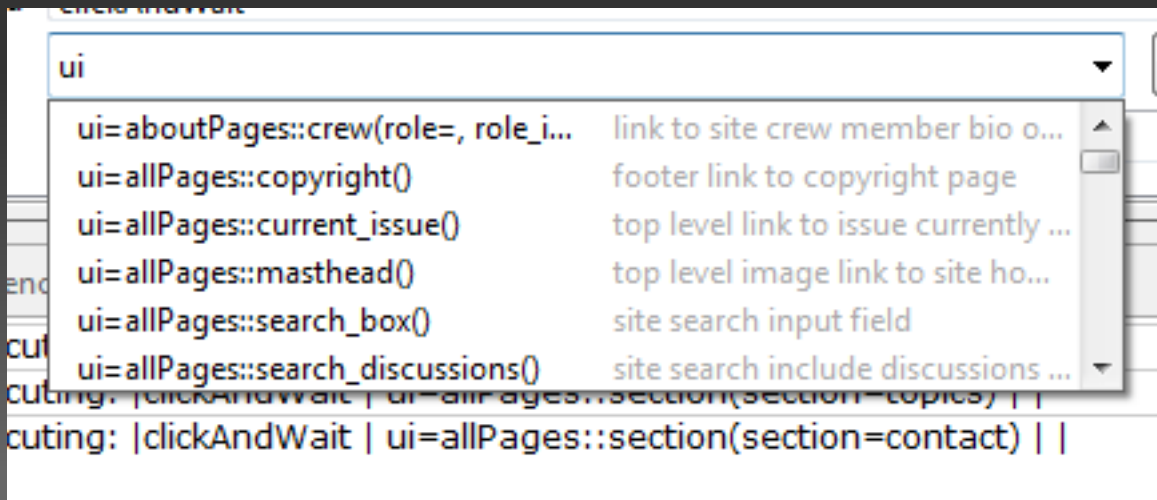
In selenium IDE we can now write a locator parameter as;

`ui=allPages::about_link()`

for example;

Command	Target	Value
click	ui=allPages::about_link()	

After a developer has written a ui-map of shortcuts, when testers, or developers start writing tests in Selenium IDE, it will now auto suggest ui-elements that you've written.



# UI-arguments modifying the XPATH

Previously we used a locator which stored an XPATH to the name about\_link, but we can use 'get locator' instead of locator to use a function which will allow us to pass in arguments to modify the locator. This is handy if we have many links which are similar and we want to access them in one ui-element, and just differ the argument...

```
map.addElement('allPages', {
  name: 'any_link',
  description: 'link to a result page',
  args: [{
    name: 'href',
    description: 'the href of the link',
    defaultValues: [ 'about.php', 'news.php' ]
  }],
  getLocator: function(args) {
    var type = args.href;
    return "//a[contains(@href, '"+ type + "')]";
  }
});
```

# UI-arguments modifying the XPATH

now we can type;

```
ui=allPages::any_link(href=about.php)
```

or

```
ui=allPages::any_link(href=news.php)
```

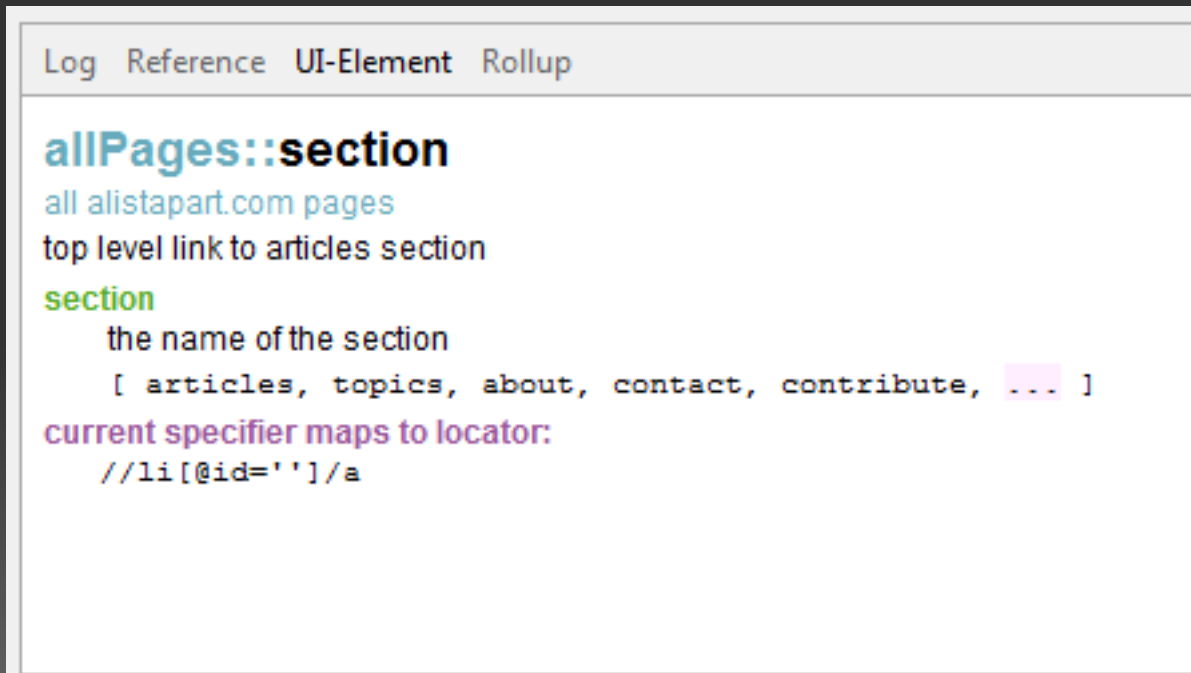
# UI-arguments properties

With the added 'args' property to our UI-element, these have properties to name, and describe the arguments that could get passed in.

- `name: 'href',`
- `description: 'the href of the link',`
- `defaultValues: [ 'about.php', 'news.php' ]`  
or to get these dynamically
- `getDefaultValues: function(inDocument) {  
 var defaultValues = [];  
 var links = inDocument.getElementsByTagName('a');  
 for (var i = 0; i < links.length; ++i) {  
 var link = links[i];  
 if (link.className == 'category') {  
 defaultValues.push(link.href);  
 }  
 }  
 return defaultValues;  
} ...`

Also In the IDE it will show the description propertie and any default argument properties you've defined for the ui-element.

Making it much easier to use the selenium test as you now don't need to look at the HTML, for the example shown here: the HTML id, to know how to access the page element.



```
Log Reference UI-Element Rollup

allPages::section
all alistapart.com pages
top level link to articles section
section
  the name of the section
  [ articles, topics, about, contact, contribute, ... ]
current specifier maps to locator:
  //li[@id='']/a
```

# Testcases without ui-element

So as we saw before the selenium test without ui-element and ui-maps in its html form, looks something like this;

```
<tr>
  <td>clickAndWait</td>
  <td>///li[@id='post-13387']/div/div[2]/h2/a</td>
  <td></td>
</tr>
<tr>
  <td>click</td>
  <td>link=exact:http://bit.ly/1Wt4mV</td>
  <td></td>
</tr>
<tr>
  <td>click</td>
  <td>///li[@id='comment-168515']/div/div[2]/a</td>
</tr>
<tr>
  <td>click</td>
  <td>///li[@id='comment-168637']/div/div[2]/a</td>
</tr>
.....
.....
```

# TestCases with UI-element example

Using UI-Element with UI-map you testcases can look like this...

```
<tr>
  <td>clickAndWait</td>
  <td>ui=allPages::section(section=topics)</td>
  <td></td>
</tr>
<tr>
  <td>clickAndWait</td>
  <td>ui=subtopicListingPages::subtopic(subtopic=Creativity)</td>
  <td></td>
</tr>
<tr>
  <td>click</td>
  <td>ui=subtopicArticleListingPages::article(index=2)</td>
  <td></td>
</tr>
```

# Rollups

A big part of UI-element is also writing Rollups.

A Rollup is a collection of UI-element executed from one command in selenium with optional arguments to modify the rollup.

They define whole actions, i.e. Login, and provide a single point of update for that action, a natural progression of ui-element thinking.

First of all we need to set up a new rollup manager object similar to the `map = new uimap()` we set up earlier;

```
var manager = new RollupManager();
```

To this we can add new rollups;

```
manager.addRollupRule({ ... });
```

```
manager.addRollupRule({ ... });
```

# Writing a Rollup

for instance if we'd like to write;

Command	Target	Value
rollup	do_search	term=Artciles

This Rollup will take the term 'Artciles' to do a search on the website, and click the search button

Firstly we add a name and a description, like a UI-element;

```
manager.addRollupRule({  
  name: 'do_search',  
  description: 'performs a search',  
  ...  
})
```

# Writing a Rollup

Then we can optionally define a **pre**, and **post** property;

**pre**: a detailed summary of the preconditions that must be satisfied for the rollup to execute successfully.

**post**: a detailed summary of the postconditions that will exist after the rollup has been executed.

example;

**pre**: 'current page contains the search box (most pages should)',

**post**: 'navigated to the search results list page for the specified term' ,

Next a description of the arguments that could be passed in, as with ui-elements;

```
args: [ {  
  name: 'term',  
  description: 'the search term'  
}], ...
```

The next properties `commandMatchers`, and `getExpandedCommands`, I'll show these in a list then come back and break them down....

# Writing a Rollup

```
.....
commandMatchers: [
  {
    command: 'type',
    target: 'ui=searchPages::search_box\\(.+',
    updateArgs: function(command, args) {
      var uiSpecifier = new UISpecifier(command.target);
      args.term = uiSpecifier.args.term;
      return args;
    }
  },
  {
    command: 'click.+',
    target: 'ui=searchPages::search_go\\(.+'
  }
],
getExpandedCommands: function(args) {
  var commands = [];
  var uiSpecifier = new UISpecifier (
    'searchPages',
    'search_box',
    { term: args.term }
  );
  commands.push({
    command: 'type',
    target: 'ui=' + uiSpecifier.toString()
  },
  {
    command: 'clickAndWait',
    target: 'ui=searchPages::search_go()'
  });
  return commands;
}
});
```

# Writing a Rollup

## getExpandedCommands

the `getexpanded` function: is where we create an array of command objects to execute in order, for the rollup action, a list of ui-elements;

```
var commands = [];  
  
commands.push({  
  command: 'type',  
  target: 'ui=' + uiSpecifier.toString()  
},  
{  
  command: 'clickAndWait',  
  target: 'ui=searchPages::search_go()'  
});
```

although in our example we also have the `uiSpecifier` bit, this is to illustrate how we can generate a ui-element with the passed in argument from selenium...

# Writing a Rollup

UISpecifier: So for the term argument used in the rollup: 'Articles'

```
var uiSpecifier = new UISpecifier(  
    'searchPages',  
    'search_box',  
    { term: args.term }  
);  
  
commands.push({  
    command: 'type',  
    target: 'ui=' + uiSpecifier.toString()  
});
```

The target that will in end up in the list of commands would be;  
`ui=searchPages::search_box(term=Articles)`

# Writing a Rollup

we could also write;

```
commands.push({  
  command: 'type',  
  target: 'ui=searchPages::search_box(term=' + args.term +  
' )  
  });
```

# Writing a Rollup

## commandMatchers

The command matchers part matches commands, and can modify them as the rollup is executing;

we have;

```
{  
  command: 'click.+',  
  target: 'ui=searchPages::search_go\\(.+'  
}
```

This matches all click or clickAndWait commands, with the target shown, from the getExpandedCommands function, although we haven't modified the command in this instance...

# Writing a Rollup

Also in the command matcher the;

```
{  
  command: 'type',  
  target: 'ui=searchPages::search_box\\(.+',  
  updateArgs: function(command, args) {  
    var uiSpecifier = new UISpecifier(command.target);  
    args.term = uiSpecifier.args.term;  
    return args;  
  }  
}
```

we match any 'type' command which has the target 'ui=searchPages::search\_box' with any term,

then we show here how we can update the commands terms if we wanted to

We take the command, parse the matched command target into a ui-element object with UISpecifier, and return the new ui-elements term.

This just shows that we could modify the terms in the command/s matched as the rollup expands, for instance we could do;

```
args.term = uiSpecifier.args.term + 'in cms2';
```

resulting in the search, with the term 'Artcile' searching for 'Articles in cms2'

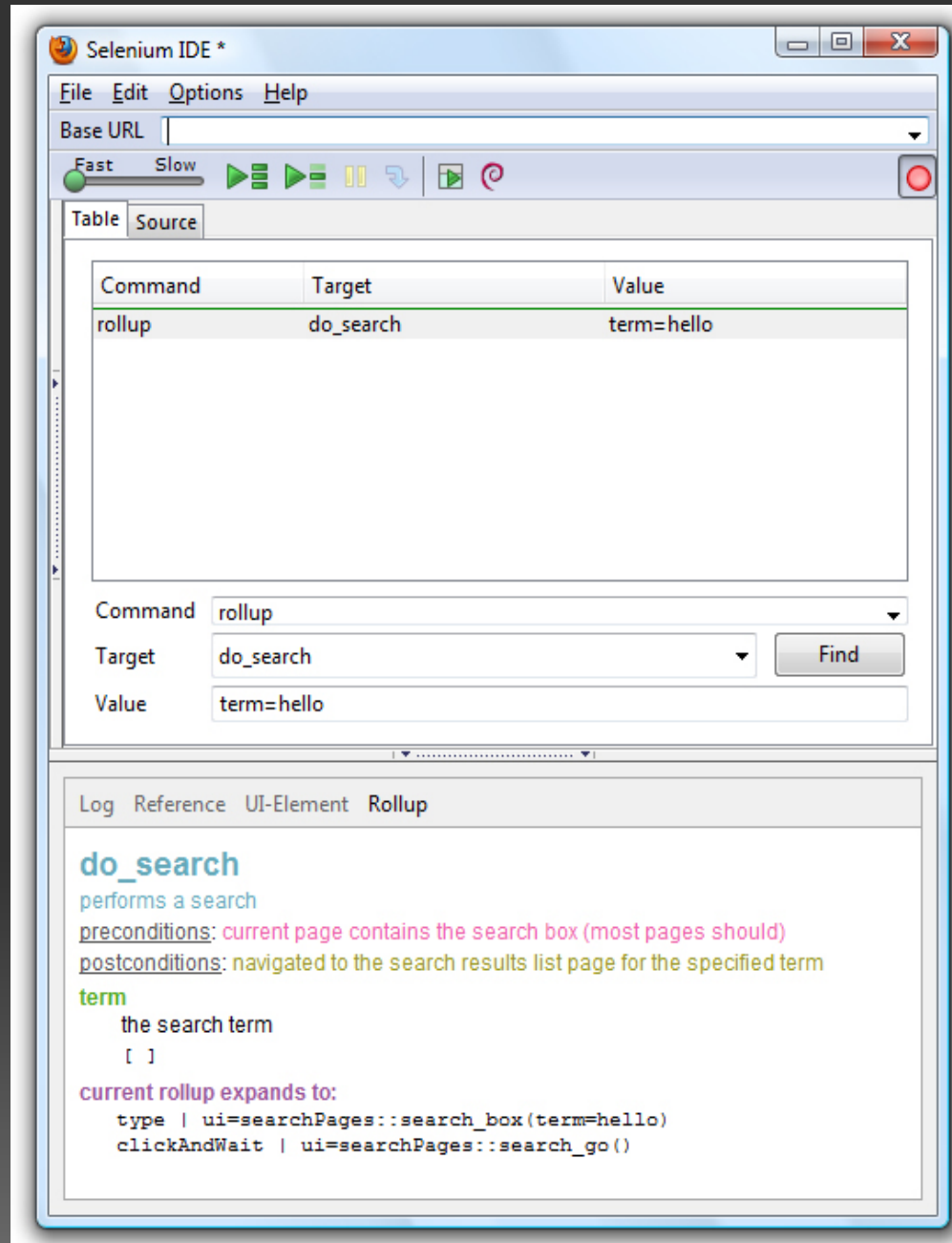
although in this example we don't modify them.

# Writing a Rollup

here's the simplified version, without any command matchers, although the properties are required, it can be empty..

```
manager.addRollupRule({  
  
  name: 'do_search',  
  description: 'performs a search',  
  pre: 'current page contains the search box (most pages should)',  
  post: 'navigated to the search results list page for the specified term',  
  args: [ {  
    name: 'term',  
    description: 'the search term'  
  } ],  
  commandMatchers: [],  
  getExpandedCommands: function(args) {  
    var commands = [];  
    commands.push({  
      command: 'type',  
      target: 'ui=searchPages::search_box(term=' + args.term + ')'  
    },  
    {  
      command: 'clickAndWait',  
      target: 'ui=searchPages::search_go()'  
    });  
    return commands;  
  }  
});
```

# Writing a Rollup



The screenshot shows the Selenium IDE interface. At the top, there is a menu bar with 'File', 'Edit', 'Options', and 'Help'. Below the menu bar is a 'Base URL' field. A toolbar contains a speed slider (set to 'Fast'), play, pause, and refresh icons. The main area is divided into 'Table' and 'Source' tabs. The 'Table' tab displays a table with the following content:

Command	Target	Value
rollup	do_search	term=hello

Below the table, there are input fields for 'Command' (set to 'rollup'), 'Target' (set to 'do\_search'), and 'Value' (set to 'term=hello'). A 'Find' button is located to the right of the 'Target' field. At the bottom of the window, there is a 'Log' tab with sub-tabs for 'Log', 'Reference', 'UI-Element', and 'Rollup'. The 'Rollup' sub-tab is active, showing the following details for the 'do\_search' command:

**do\_search**  
performs a search  
preconditions: current page contains the search box (most pages should)  
postconditions: navigated to the search results list page for the specified term  
**term**  
the search term  
[ ]  
**current rollup expands to:**  
type | ui=searchPages::search\_box(term=hello)  
clickAndWait | ui=searchPages::search\_go()

# UI-Element and UI-Maps Conclusion

- **Easier for non-developers to write tests.**
- **Saves time when writing tests,**
- **Removes the need to write very much Code to access each element,**
- **Promotes code re-use,**
- **Adds the ability to 'roll-up' shortcuts into bunches of commands, called with one line.**
- **provides a single point of update for locators, and rollup actions**

# Alistapart UI-map example

An example of a full ui-map of ui-elements and roll-ups is also included in the latest release of selenium core, created by [www.Alistapart.com](http://www.Alistapart.com) for their site as a reference.

To show a full ui-map in action we can load the sample map and use it on the [www.alistapart.com](http://www.alistapart.com) website with selenium IDE...



# Our UI-map

CMS2 have started to write our own ui-map with ui-elements, and rollups.

UI-elements for navigation

UI-elements for most articles elements

Login rollup

Navigatie rollup

we now have the folders;

individual tests: for singular tests, wich could be reused, or selenium ide test.

widget tests: for widget tests, specifically for a widget.

modular tests: login, test a module, or section of the appication, and logg out.

suites: folders containing suites i.e modular smoke suite: which contains an index file, calling in tests from the above folders to create a batch of tests to run.

# Our UI-Map

Example in Testrunner of cms-trunks modular smoke tests...

# Links

## **Main Selenium site**

<http://seleniumhq.org/>

## **A More Readable Selenium API Documentation**

<http://cavaliere.org/posts/34>

## **Selenium commands library**

[http://wiki.expecco.de/index.php5/Selenium\\_Library](http://wiki.expecco.de/index.php5/Selenium_Library)

## **Contributed user extentions**

<http://wiki.openqa.org/display/SEL/Contributed+User-Extensions>

## **UI-element- all about it!**

<http://functionaltestautomation.blogspot.com/2009/09/ui-elements-all-about-it.html>

## **UI-map with selenium-RC**

<http://functionaltestautomation.blogspot.com/2009/09/using-ui-elements-with-selenium-rc.html>

## **Selenium pluggin for hudson builds**

<http://wiki.hudson-ci.org/display/HUDSON/Selenium+Plugin>

Thanks