

Tellurium Automated Testing Framework

Reference Documentation

**Jian Fang
Ajay Ravichandran
Rajan Pokhrel**

Tellurium Automated Testing Framework: Reference Documentation

by Jian Fang, Ajay Ravichandran, and Rajan Pokhrel

v0.7.0

Publication date May 15, 2010

Copyright © 2010 TelluriumSource

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

Table of Contents

1. Overview of Tellurium	1
What is Tellurium	1
Tellurium, the New Approach for Web Testing	4
Tellurium Architecture	6
History of Tellurium	8
Tellurium Team and Community	9
Tellurium Sub-projects	9
2. What's New in Tellurium 0.7.0	11
Introduction	11
New features	11
Changes in Engine	22
Changes in Maven Build	22
Reference Project	24
How to Obtain Tellurium 0.7.0	24
3. Getting Started	25
Create a Tellurium Project	25
Tellurium Maven Archetypes	25
Tellurium Ant Projects	26
Setup Tellurium Project in IDEs	26
IntelliJ IDEA	26
NetBeans	26
Eclipse	26
Create a UI Module	27
Create Tellurium Test Cases	28
4. Tellurium UI Objects	29
Basic UI Object	29
UI Object Default Attributes	31
UI Object Description	32
Button	32
Submit Button	32
Check Box	32
Div	33
Image	33
Icon	33
Radio Button	34
Text Box	34
Input Box	34
URL Link	35
Repeat Object	35
Selector	37
Container	40
Form	40
Table	40
Standard Table	41
List	43
Frame	43
Window	44
Option	44
All Purpose Object	45
5. Tellurium Core Basics	46
UiID Attribute	46
Locator Attributes	47
Group Attribute	47
self attribute	48
Respond Attribute	49

CSS Selector	50
New DSL Methods	50
Additional CSS Attribute Selectors	50
Locator Agnostic Methods	51
UI Templates	54
"Include" Frequently Used Sets of Elements in UI Modules	55
Logical Container	56
toggle	58
getHTMLSource	58
type and keyType	59
Customize Individual Test Settings Using setCustomConfig	59
User Custom Selenium Extensions	60
The Dump Method	61
Engine State Offline Update	62
Testing Support	63
Tellurium JUnit Test Case	63
Tellurium TestNG Test Case	63
Tellurium Groovy Test Case	64
TelluriumMockJUnitTestCase and TelluriumMockTestNGTestCase	64
Tellurium Configuration	66
Run DSL Script	69
useTelluriumEngine	70
Trace	70
Methods Accessible in Test Cases	71
Environment	72
Get UIs by Tag Name	73
Misc	74
6. Tellurium Core APIs	76
DSL Methods	76
Data Access Methods	78
UI Module APIs	81
Example	81
dump	81
toString	82
toHTML	83
getHTMLSource	83
show	85
validate	85
Closest Match	87
Test Support DSLs	88
Example:	89
Example:	89
7. Tellurium Core Advanced Topics	91
Data Driven Testing	91
Data Provider	91
TelluriumDataDrivenModule	93
Tellurium Data Driven Test	96
Implementations	98
Selenium Grid Support	100
Mock Http Server	103
Mock Http Handler Class	103
Mock Http Server	104
Generate Html Source From UI Modules	106
Implementation	106
Usage	108
Tellurium Powerful Utility: Diagnose	109
Implementation	110
Usage	111

Internationalization support in Tellurium	114
Better Reporting With ReportNG	117
Macro Command	118
Groovy Features Used in Tellurium	119
BuildSupport	119
Dynamic Scripting	120
GroovyInterceptable	121
methodMissing	121
Singleton	121
GString	122
Optional Type	122
Closure	123
Groovy Syntax	123
Varargs	124
propertyMissing	124
Delegate	124
Grape	125
8. Tellurium UID Description Language	126
Introduction	126
Tellurium UID Description Language	127
UDL Grammars	127
Routing	132
Implementation	135
Antlr 3	135
Data Structure	136
Embedded Java Code	137
Parser	139
Unit Test	139
9. Tellurium Widgets	141
Introduction	141
Widget Implementation	141
Tellurium Widget Archetype	143
10. Tellurium Engine	144
Code Structure	144
CSS Selector Support	145
:te_text	148
:group	148
:styles	149
:nextToLast	150
outerHTML	151
UI Module Group Locating	151
Basic Flow	151
Data Structures	152
Locate	154
Relax	158
Usage	159
UI Module Caching	161
New APIs	163
Debug Tellurium Engine	166
Tellurium UI Module Visual Effect	169
Build a Snapshot Tree	169
Demo	174
Engine Logging	176
JavaScript Error Stack	177
11. Tellurium Maven Archetypes	180
Prerequisites	180
settings.xml	180
Tellurium JUnit Archetype	181

Tellurium TestNG Archetype	182
Tellurium Widget Archetype	182
12. Tellurium UI Module Firefox Plugin	183
Install TrUMP	183
TrUMP Workflow	183
How TrUMP Works	184
The UI Module Generating Algorithm	187
13. Tellurium Reference Projects	194
Introduction	194
Tellurium Website Project	194
Create Custom UI objects	195
Create UI modules	196
Create Java Test Cases	197
Create and Run DSL Scripts	199
Data Driven Testing	201
Tellurium ui-examples Project	203
Selector	203
Container	204
Form	205
List	205
Table	207
Repeat	208
14. Tellurium Reference	211
Maven	211
DocBook	212
Book	212
Chapter	213
section	214
Link	214
Image	214
screen	214
programlist	215
list	215
table	216
Tools	216
A. FAQs	217
When Did Tellurium Start?	217
What Are the Main Differences Between Tellurium and Selenium?	217
Do I Need to Know Groovy Before I Use Tellurium?	217
What Unit Test and Functional Test Frameworks Does Tellurium Support?	218
Does Tellurium Provide Any Tools to Automatically Generate UI Modules?	218
What Build System Does Tellurium Use?	219
What is the Best Way to Create a Tellurium Project?	219
Where Can I Find API Documents for Tellurium?	219
Is There a Tellurium Tutorial Available?	220
Where Can I Find a Sample Tellurium Configuration File?	220
Tellurium Dependencies	220
What Is the ui. in UI Module?	221
How Do I Add My Own UI Object to Tellurium?	222
How to Build Tellurium from Sourc	222
What is the Issue with Selenium XPath Expressions and Why is There a Need to Create a UI Module?	223
How to write assertions in Tellurium DSL scripts	225
How to upgrade Firefox version in Selenium server	226
How to run Selenium server remotely in Tellurium	227
Differences among Tellurium Table, List, and Container	228
How do I use a Firefox profile in Tellurium	228
How to Overwrite Tellurium Settings in My Test Class	229

How to reuse a frequently used set of elements	229
How to handle Table with multiple tbody elements	230
How to Run Tellurium Tests in Different Browsers	231
How to use the new XPath library in Selenium	232
How to Debug Selenium Core	232
How to Debug Tellurium in IE	233
How to use jQuery Selector with weird characters in its ID	233
How to Use Tellurium for XHTML	233
What Are the Differences Between connectUrl and openUrl	234
How to do Attribute Partial Matching in Tellurium	234
What are the rules to define Tellurium UIDs	235
How to load Tellurium configuration from a String	235
How to register a custom method in Tellurium API	236
How to Access Tellurium Maven Repo Behind a Firewall	237
How to Generate IDE project files	237
How to Run Tellurium Tests in Google Chrome	238
How to run headless tests with Xvfb	238
How to setup Groovy Grape	239
How to Search Tellurium Documents	239
How to Contribute to Tellurium	240
Tellurium Future Directions	240
B. Tellurium Project Setup with IntelliJ	241
Prerequisites	241
Project Setup	241
Summary	247
C. Integration Tests with Maven Cargo Plugin	249
Introduction	249
Example	249
Maven Update	249
Tellurium Tests	252
Cargo Maven Plugin	253
D. Use Firebug and JQuery to Trace Problems in Tellurium Tests	256
Prerequisites	256
Firefox Profile	256
Firebug Support	256
Debug and Trace	258
Debug JavaScript Using Firebug	258
Trace Problems Using jQuery	260
E. Resources	262
Tellurium Community	262
Users' Experiences	262
Interviews	262
Presentations and Videos	262
IDEs	263
Build	263
Related	263
F. Sample Maven Configurations	264
settings.xml	264
Sample Tellurium Project Maven POM	264
G. Sample Tellurium Project Configuration	269
TelluriumConfig.groovy	269
JSON String	270
H. Sample Ant Build Script	272
build.properties	272
build.xml	272
I. Sample Run DSL Script	275
Sample Groovy Grape Configuration	275
rundsl.groovy	275

List of Tables

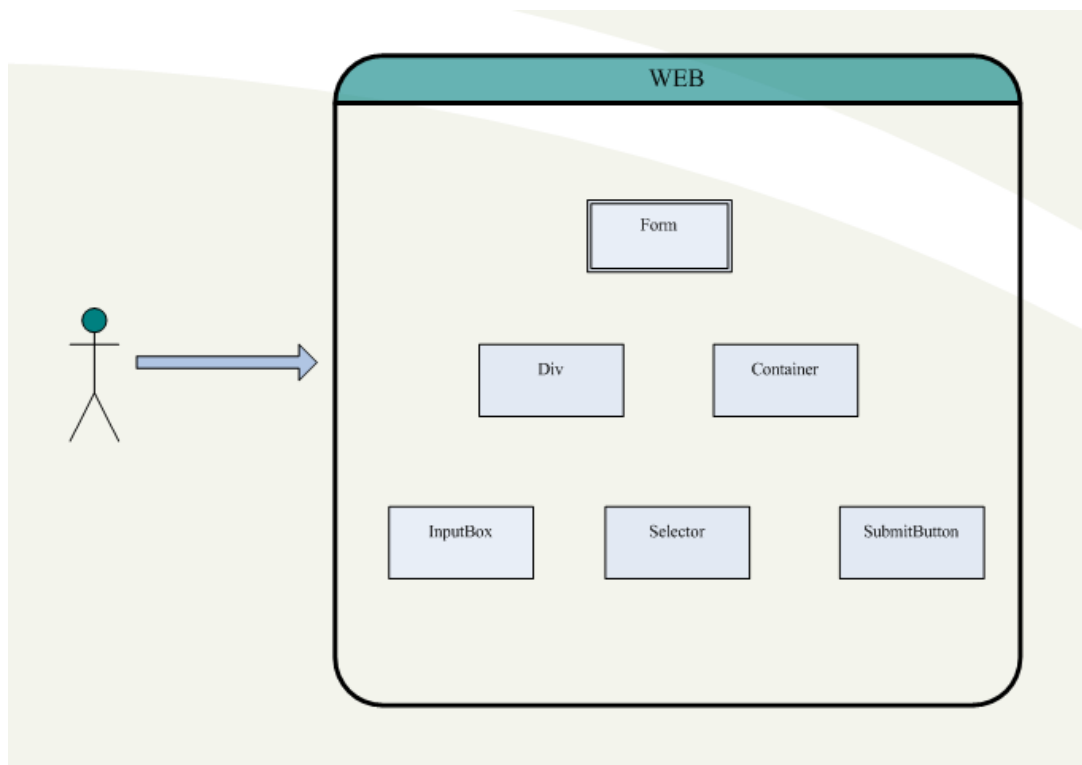
4.1. UI Object Attributes	30
4.2. UI Object Default Attributes	31

Chapter 1. Overview of Tellurium

What is Tellurium

The Tellurium Automated Testing Framework [<http://code.google.com/p/aost/>] (Tellurium) is an automated testing framework for web applications. Tellurium grew up from the Selenium framework [<http://seleniumhq.org/>], but with a different testing concept. Starting from Tellurium 0.7.0, Tellurium added Tellurium Engine to replace the Selenium Core to better support Tellurium.

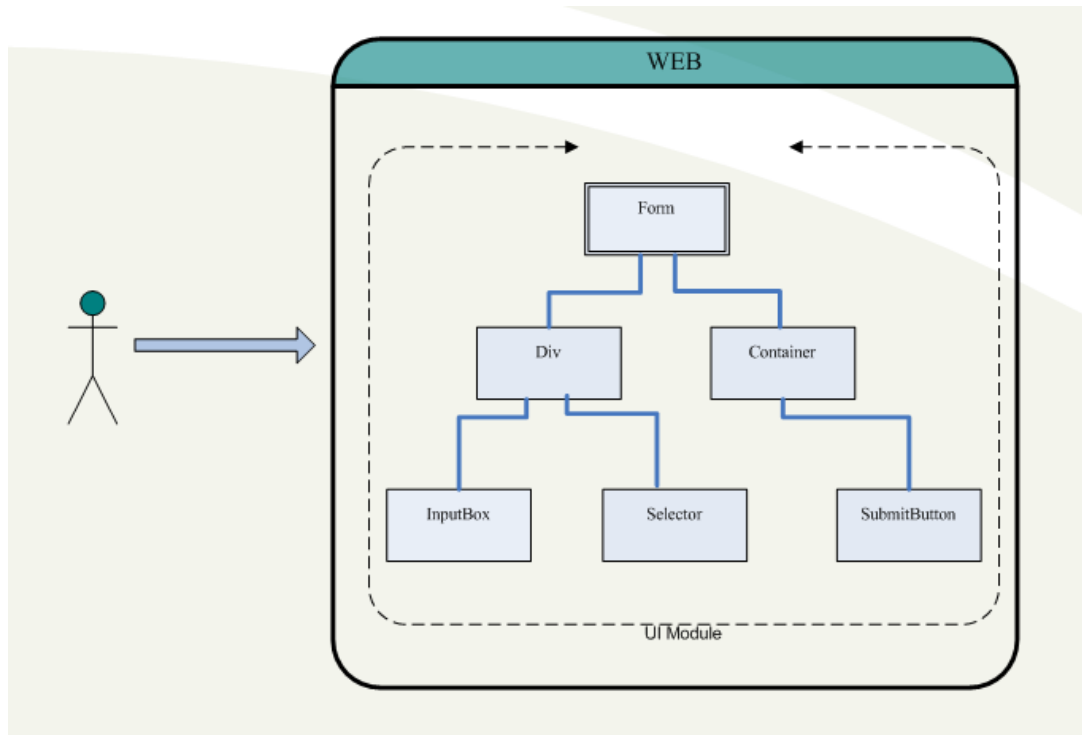
For most existing web testing frameworks like Selenium, they mainly focus on individual UI elements as follows. For most existing web testing frameworks like Selenium, they mainly focus on individual UI elements as follows.



For example:

```
selenium.click("//div[3]/input[@value='Create']");
```

For Tellurium, we treated the whole UI elements as a widget and we call it a UI module.



We define the UI module as follows.

```

ui.Form(uid: "Form", clocator: [tag: "form"]){
  Div(uid: "User", clocator: [:]){
    Selector(uid: "Sex", clocator: [:])
    InputBox(uid: "Input", clocator: [tag: "input", type: "text",
      name: "j_username"])
  }
  Container(uid: "Finish", clocator: [tag: "tr"]){
    SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit",
      value: "Login", name: "submit"])
  }
}

```

If we think Selenium as the "C" language, Tellurium is like the "C++" language, which uses a different testing concept. There are couple advantages to describe the UI elements as a UI module.

Expressive is obvious. For example, you can see clearly what the UI you are testing against. For the test code, you have DSL style test code such as:

```
type "Form.User.Input", "TelluriumSource"
```

Robust to Changes Test robust is always a big issue for Selenium. To solve this problem is one of the main motivations that Tellurium was created for. Tellurium uses UI attributes to describe UI instead of fixed locators. If we change the attributes, new runtime locators will be generated by the framework so that Tellurium can self-adapt to UI changes to some degree. The Santa algorithm [<http://code.google.com/p/aost/wiki/SantaUiModuleGroupLocatingAlgorithm>] in Tellurium new engine further improves the test robust by using UI partial matching.

Represent Dynamic Web Content Easily The Tellurium UI templates [http://code.google.com/p/aost/wiki/UserGuide070TelluriumBasics#UI_Templates] are used to represent dynamic web content very easily. For example, Tellurium issue search result widget [<http://code.google.com/p/aost/issues/list>] can be easily represented as follows.

```

ui.Table(uid: "issueResult", clocator: [id: "resultstable", class: "results"],
  group: "true") {
  //Define the header elements
  UrlLink(uid: "{header: any} as ID", clocator: [text: "**ID"])
  UrlLink(uid: "{header: any} as Type", clocator: [text: "**Type"])
  UrlLink(uid: "{header: any} as Status", clocator: [text: "**Status"])
  UrlLink(uid: "{header: any} as Priority", clocator: [text: "**Priority"])
  UrlLink(uid: "{header: any} as Milestone", clocator: [text: "**Milestone"])
  UrlLink(uid: "{header: any} as Owner", clocator: [text: "**Owner"])
  UrlLink(uid: "{header: any} as Summary",
    clocator: [text: "**Summary + Labels"])
  UrlLink(uid: "{header: any} as Extra", clocator: [text: "**..."])

  //Define table body elements
  //Column "Extra" are TextBoxes
  TextBox(uid: "{row: all, column -> Extra}", clocator: [:])
  //For the rest, they are UrlLinks
  UrlLink(uid: "{row: all, column: all}", clocator: [:])
}

```

Easy to Maintain Tellurium emphasizes the decoupling of UI from test code. The structured test code makes Tellurium easier to maintain and refactor.

Motivation Automated web testing has always been one of the hottest and most important topics in the software testing arena when it comes to the rising popularity of Rich Internet applications (RIA) and Ajax-based web applications. With the advent of new web techniques such as RIA and Ajax, automated web testing tools must keep current with changes in technology and be able to address the following challenges:

- **JavaScript Events:** JavaScript is everywhere on the web today. Many web applications are JavaScript heavy. To test JavaScript, the automated testing framework should be able to trigger JavaScript events in a convenient way.
- **Ajax for Dynamic Web Content:** Web applications have many benefits over desktop applications. For example, these applications have no installation and updates are instantaneous and easier to support. Ajax is a convenient way to update a part of the web page without refreshing the whole page. AJAX makes web applications richer and more user-friendly. The web context for an Ajax application is usually dynamic. For example, in a data grid, the data and number of rows keeps changing at runtime.
- **Robust/Responsive to Changes:** A good automated web-testing tool should be able to address the changes in the web context to some degree so that users do not need to keep updating the test code.
- **Easy to Maintain:** In an agile testing world, software development is based on iterations, and new features are added on in each sprint. The functional tests or user acceptance tests must be refactored and updated for the new features. The testing framework should provide the flexibility for users to maintain the test code easily.
- **Re-usability:** Many web applications use the same UI module for different parts of the application. The adoption of JavaScript frameworks such as Dojo and ExtJS increases the chance of using the same UI module for different web applications. A good testing framework should also be able to provide the re-usability of test modules.
- **Expressiveness:** The testing framework provides users without much coding experience the ability to easily write test code or scripts in a familiar way, such as using a domain specific language (DSL).

The Tellurium Automated Testing Framework (Tellurium) is designed around these considerations and has defined as its focus the following goals:

- Robust/responsive to changes; allow changes to be localized
- Addresses dynamic web contexts such as JavaScript events and Ajax

- Easy to refactor and maintain
- Modular; test modules are reusable
- Expressive and easy to use

Tellurium, the New Approach for Web Testing

The Tellurium Automated Testing Framework (Tellurium) is an open source automated testing framework for web applications that addresses the challenges and problems of today's web testing.

Most existing web testing tools/frameworks focus on individual UI elements such as links and buttons. Tellurium takes a new approach for automated web testing by using the concept of the UI module.

The *UI module* is a collection of UI elements grouped together. Usually, the UI module represents a composite UI object in the format of nested basic UI elements. For example, the Google search UI module can be expressed as follows:

```
ui.Container(uid: "GoogleSearchModule", clocator: [tag: "td"], group: "true"){
    InputBox(uid: "Input", clocator: [title: "Google Search"])
    SubmitButton(uid: "Search", clocator: [name: "btnG", value: "Google Search"])
    SubmitButton(uid: "ImFeelingLucky", clocator: [value: "I'm Feeling Lucky"])
}
```

Tellurium is built on the foundation of the UI module. The UI module makes it possible to build locators for UI elements at runtime. First, this makes Tellurium robust and responsive to changes from internal UI elements. Second, the UI module makes Tellurium expressive. UI elements can be referred to simply by appending the names (uid) along the path to the specific element. This also enables *Tellurium's Group Locating* feature, making composite objects reusable, and addressing dynamic web pages.

Tellurium is implemented in Groovy and Java. The test cases can be written in Java, Groovy, or pure Domain Specific Language (DSL) scripts. Tellurium evolved out of Selenium. However, the UI testing approach is completely different. Tellurium is not a "record and replay" style framework, and it enforces the separation of UI modules from test code, making refactoring easy.

For example, once the Google Search UI module is defined as previously shown, the test code is written as follows:

```
type "GoogleSearchModule.Input", "Tellurium test"
click "GoogleSearchModule.Search"
```

Tellurium sets the Object to Locator Mapping (OLM) automatically at runtime so that UI objects can be defined simply by their attributes using Composite Locators. Tellurium uses the Group Locating Concept (GLC) to exploit information inside a collection of UI components so that locators can find their elements.

Tellurium also defines a set of DSLs for web testing. Furthermore, Tellurium uses UI templates to define sets of dynamic UI elements at runtime. As a result, Tellurium is robust, expressive, flexible, reusable, and easy to maintain.

The main features of Tellurium include:

- Abstract UI objects to encapsulate web UI elements

- UI module for structured test code and re-usability
- DSL for UI definition, actions, and testing
- Composite Locator to use a set of attributes to describe a UI element
- Group locating to exploit information inside a collection of UI components
- Dynamically generate runtime locators to localize changes
- UI templates for dynamic web content
- XPath support
- jQuery selector support to improve test speed in IE
- Locator caching to improve speed
- Javascript event support
- Use Tellurium Firefox plugin, Trump, to automatically generate UI modules
- Dojo and ExtJS widget extensions
- Data driven test support
- Selenium Grid support
- JUnit and TestNG support
- Ant and Maven support

How Challenges and Problems Are Addressed in Tellurium?

First, Tellurium does not use "record and replay". Instead, it uses the Tellurium Firefox plugin TrUMP to generate the UI module (not test code) for you. Then test code based on the UI module is created.

In this way, the UI and the test code are decoupled. The structured test code in Tellurium makes it much easier to refactor and maintain the code.

The composite locator uses UI element attributes to define the UI, and the actual locator (for example, XPath or jQuery selector), is generated at runtime. Any updates to the composite locator lead to different runtime locators, and the changes inside the UI module are localized.

The Group locating is used to remove the dependency of the UI objects from external UI elements (for example, external UI changes do not affect the current UI module for most cases), so that test code is robust and responsive to changes up to a certain level.

Tellurium uses the *respond* attribute in a UI object to specify JavaScript events, and the rest is handled automatically by the framework itself.

UI templates are a powerful feature in Tellurium used to represent many identical UI elements or a dynamic size of different UI elements at runtime. This is extremely useful in testing dynamic web contexts such as a data grid.

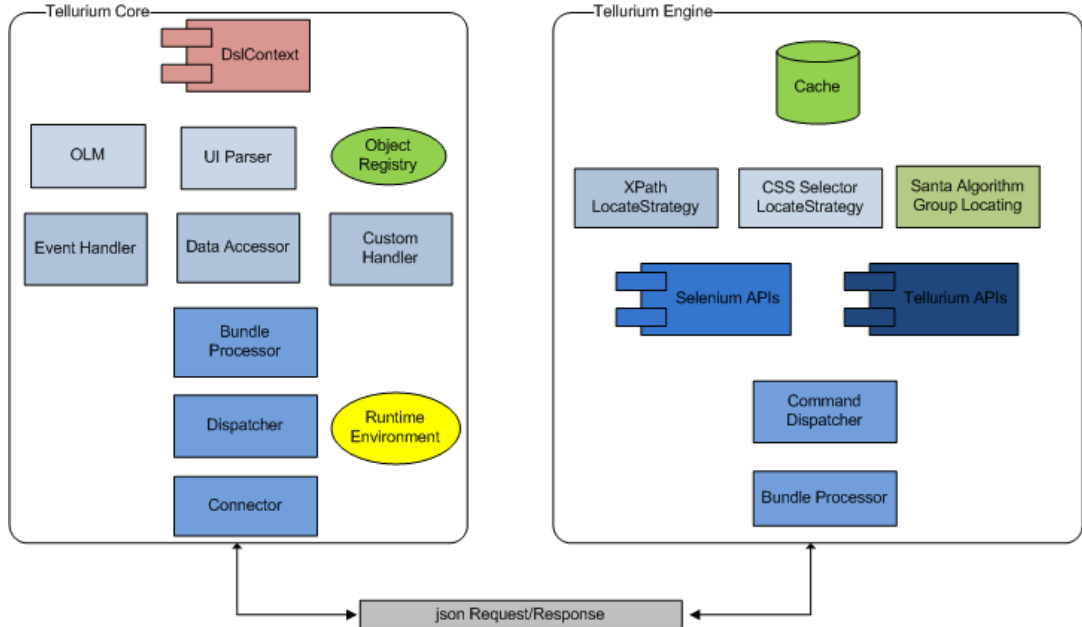
The *Option* UI object is designed to automatically address dynamic web contexts with multiple possible UI patterns.

Re-usability is achieved by the UI module when working within one application and by Tellurium Widgets when working across different web applications. With the Domain Specific Language (DSL) in Tellurium, UI modules can be defined and test code written in a very expressive way.

Tellurium also provides flexibility to write test code in Java, Groovy, or pure DSL scripts.

Tellurium Architecture

Tellurium architecture is shown in the following diagram.



There are two major parts, i.e., the Tellurium Core, which does Java/Groovy object to runtime locator mapping, event handling, and command bundling. The Tellurium Engine is embedded inside the Selenium server and is a test driving engine for Tellurium. The two are connected by Selenium RC.

The DSL parser consists of the DSL Object Parser, Object Builders, and the Object Registry.

Using Groovy builder pattern, UI objects are defined expressively and in a nested fashion. The DSL object parser parses the DSL object definition recursively and uses object builders to build the objects on the fly. An object builder registry is designed to hold all predefined UI object builders in the Tellurium framework, and the DSL object parser looks at the builder registry to find the appropriate builders.

Since the registry is a hash map, you can override a builder with a new one using the same UI name. Users can also add their customer builders into the builder registry. The DSL object definition always comes first with a container type object. An object registry (a hash map) is used to store all top level UI Objects. As a result, for each DSL object definition, the top object IDs must be unique in the DslContext. The object registry is used by the framework to search for objects by their IDs and fetch objects for different actions.

The Object Locator Mapping (OLM) is the core of the Tellurium framework and it includes UI ID mapping, XPath builder, jQuery selector builder, and Group Locating.

The UI ID supports nested objects. For example, "menu.wiki" stands for a URL Link "wiki" inside a container called "menu".

The UI ID also supports one-dimensional and two-dimensional indices for tables and lists. For example, "main.table[2][3]" stands for the UI object of the 2nd row and the 3rd column of a table inside the container "main".

XPath builder builds the XPath from the composite locator. For example, a set of attributes. Starting with version 0.6.0, Tellurium supports jQuery selectors to address the problem of poor performance of XPath in Internet Explorer.

jQuery selector builders are used to automatically generate jQuery selectors instead of XPath with the following advantages:

- Provides faster performance in IE
- Leverages the power of jQuery to retrieve bulk data from the web by testing with one method call
- Provides new features using jQuery attribute selectors

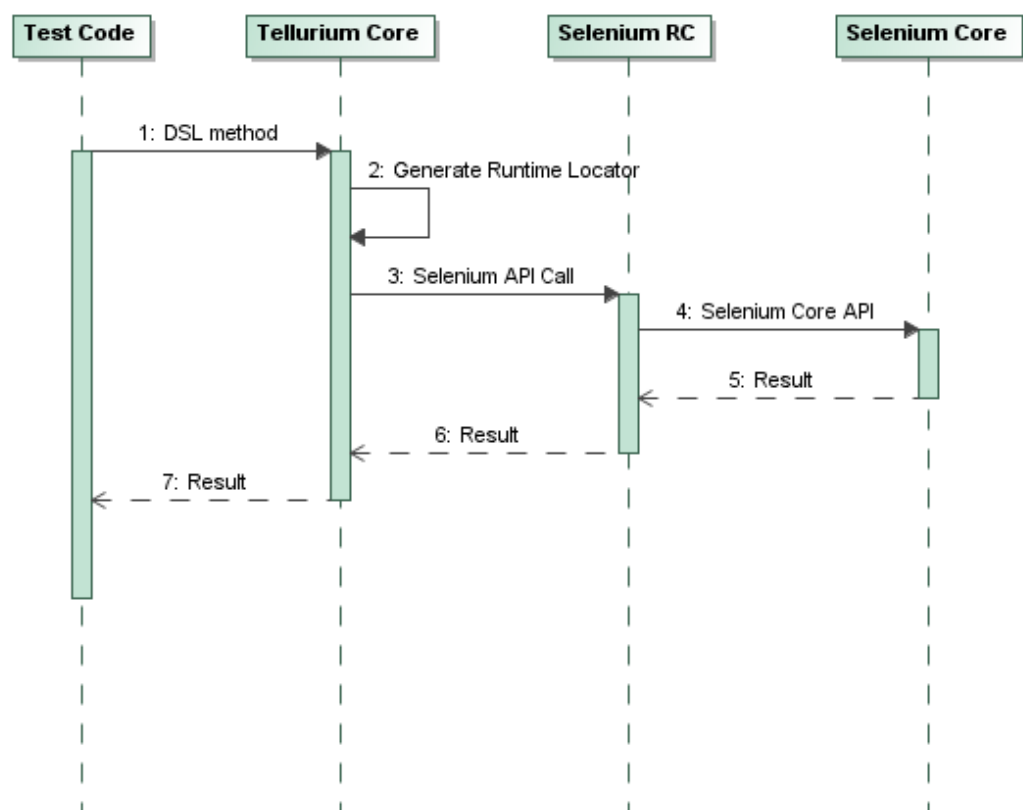
The Group Locating Concept (GLC) exploits the group information inside a collection of UI objects to assist in finding the locator of the UI objects collection.

The Eventhandler handles all events such as "click", "type", "select", etc.

The Data Accessor fetches data or UI status from the DOM. The dispatcher delegates all calls it receives from the Eventhandler and the data accessor attached to the connector is also connected to the Tellurium engine.

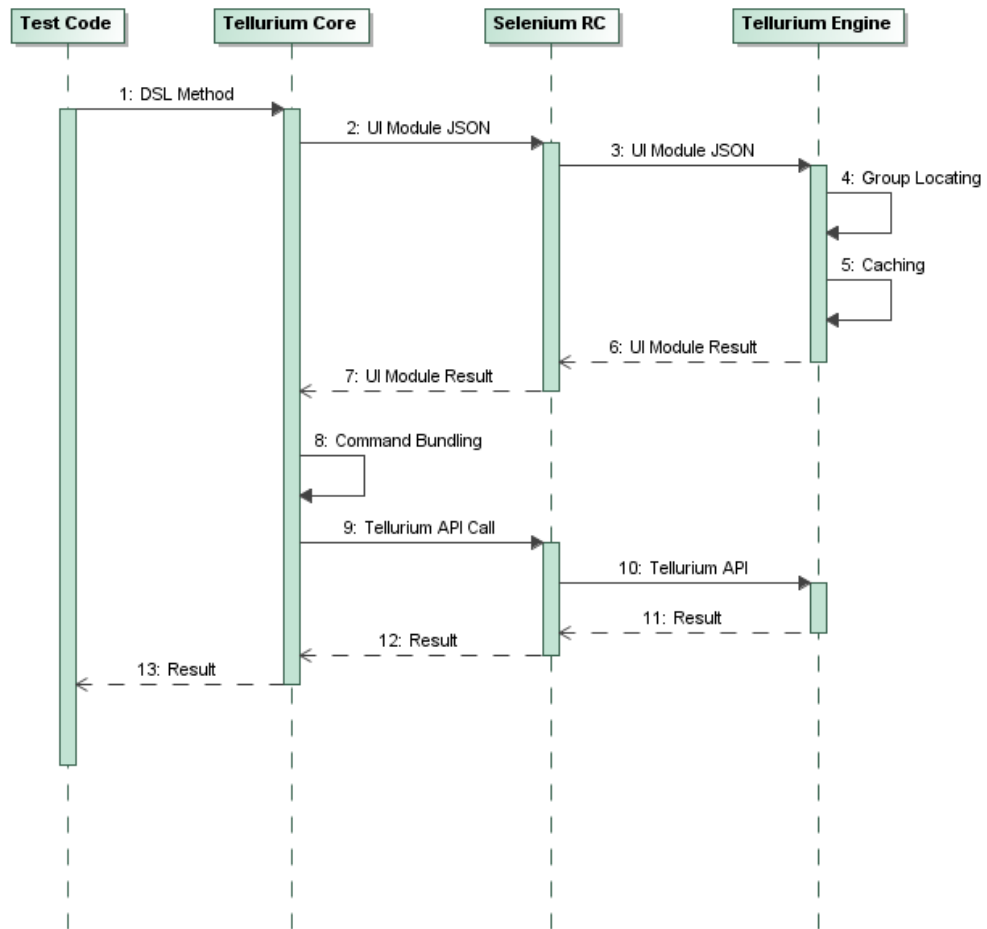
The dispatcher is designed to decouple the rest of the Tellurium framework from the base test driving engine so that it can be switched to a different test driving engine by simply changing the dispatcher logic.

Tellurium works in two mode. The first one is to work as a wrap of the Selenium framework.



That is to say, Tellurium core generates the runtime locator based on the attributes in a UI module and then pass the selenium calls to Selenium core with Tellurium extensions.

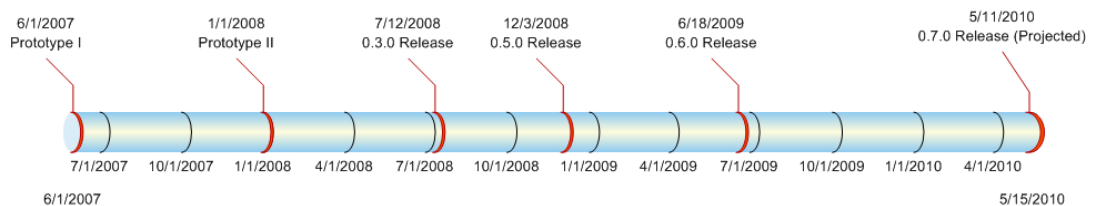
Tellurium is evolving to its own test driving Engine and the work mode is different as shown in the following sequence diagram.



The Tellurium Core will convert the UI module into a JSON representation and pass it to Tellurium Engine for the first time when the UI module is used. The Tellurium Engine uses the Santa algorithm to locate the whole UI module and put it into a cache. For the sequent calls, the cached UI module will be used instead of locating them again. Another new features in Tellurium 0.7.0 is the Macro Command [http://code.google.com/p/aost/wiki/Tellurium070Update#Macro_Command], which combine multiple commands into one batch and then send them to Tellurium engine in one call to reduce the round trip latency. For more new features in 0.7.0, please read What's New in Tellurium 0.7.0 [<http://code.google.com/p/aost/wiki/Tellurium070Update>].

History of Tellurium

Tellurium was started in June 2007 as an internal project and became an open source project on Google Code [<http://code.google.com/p/aost>] in June 2008. We release on a regular basis and the latest release 0.7.0 will be out next month.

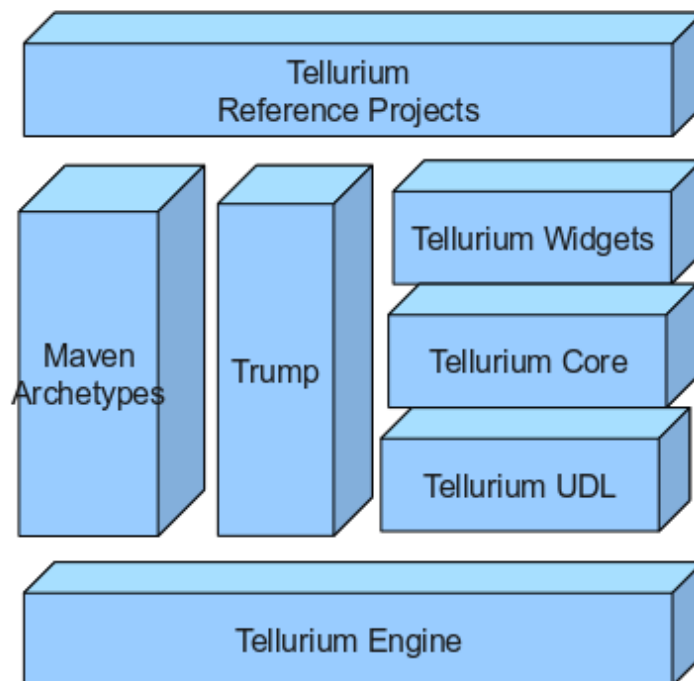


Tellurium Team and Community

Tellurium Team [<http://code.google.com/p/aost/wiki/ContributorList>] consists of 4-5 active team members and couple more contributors. We have Tellurium user group [<http://groups.google.com/group/tellurium-users>] and Tellurium developer group [<http://groups.google.com/group/tellurium-developers>]. Latest update information are available on <http://twitter.com/TelluriumSource> twitter []. Tellurium also provides a community repository on GitHub [<http://github.com/telluriumsource/tellurium>] so that anyone can fork Tellurium and contribute back to us. Tellurium wiki documents will be migrated to TelluriumSource.org [<http://telluriumsource.org/display/TE/Home>], a domain owned by us.

Tellurium Sub-projects

Tellurium began as a small core project and quickly generated multiple sub-projects including: UDL, Core, Engine, Widget extensions, Maven Archetypes, and Trump sub-projects as shown in the following diagram.



- Tellurium UDL: Tellurium UID Description Language [<http://code.google.com/p/aost/wiki/TelluriumUIDDescriptionLanguage>] (UDL) Parser.
- Tellurium Engine: Based on Selenium Core with UI module, CSS selector, command bundle, and exception hierarchy support.
- Tellurium Core: UI module, APIs, DSL, Object to Runtime Locator mapping (OLM), and test support.
- Tellurium Extensions: Dojo Javascript widgets and ExtJS Javascript widgets.
- Tellurium UI Module Plugin (Trump): A Firefox plugin [<http://code.google.com/p/aost/wiki/TrUMP>] to automatically generate the UI module after users select the UI elements from the web under testing.

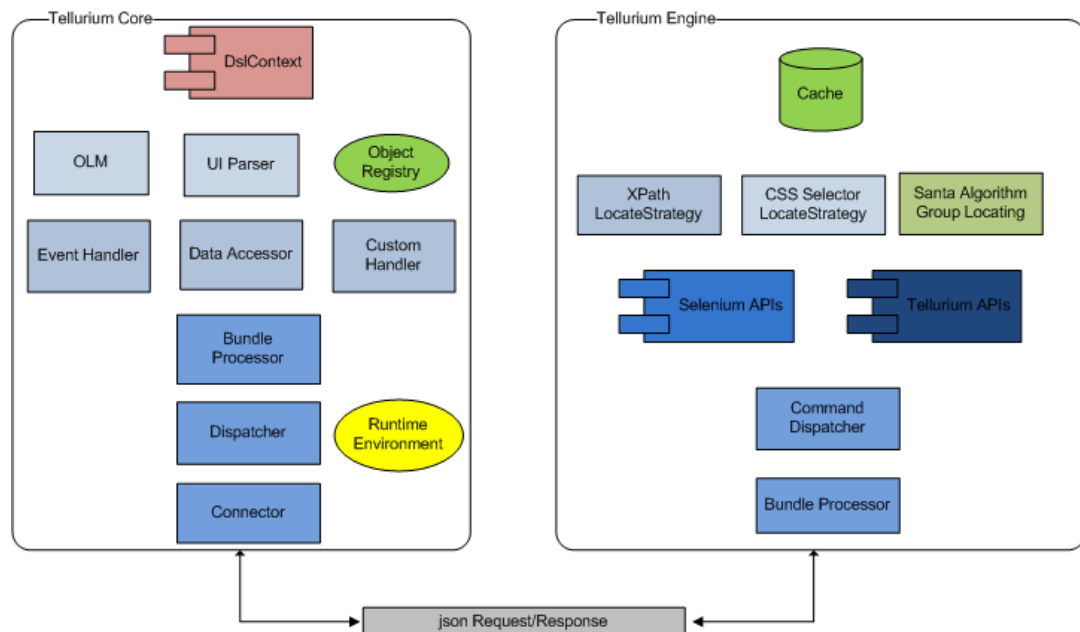
- Tellurium Maven Archetypes: Maven archetypes [http://code.google.com/p/aost/wiki/UserGuide070TelluriumSubprojects#Tellurium_Maven_Archetypes] to generate skeleton Tellurium JUnit and Tellurium TestNG projects using one Maven command.
- Tellurium Reference Projects: Use Tellurium project site as examples to illustrate how to use different features in Tellurium and how to create Tellurium test cases.

Chapter 2. What's New in Tellurium 0.7.0

Introduction

We have closed over 200 issues and have included several interesting new features. There have been a lot of fundamental changes in Tellurium 0.7.0 as compared to 0.6.0 such as the group locating algorithm, module caching, macro command, jquery based APIs and i18n support.

The architecture of Tellurium 0.7.0 has been changed and the system diagram is shown as follows,



New features

- **Package Name change**

Tellurium Core used the package name "org.tellurium", but we do not actually own the domain "tellurium.org". Our Tellurium team came out with a domain name "telluriumsource.org" and have registered this domain name. As a result, we changed the package name from "org.tellurium" to "org.telluriumsource".

In your code, if you have the following import statement,

```
import org.tellurium.dsl.DslContext
```

please change it to

```
import org.telluriumsource.dsl.DslContext
```

Accordingly, the Maven dependency should be changed as follows,

```
<groupId>org.telluriumsource</groupId>
<artifactId>tellurium-core</artifactId>
<version>0.7.0-SNAPSHOT</version>
```

- **Package Structure**

The original core package structure was a flat one and it has been changed as follows,

```
`-- org
   |-- telluriumsource
   |   |-- component
   |   |   |-- bundle
   |   |   |-- client
   |   |   |-- connector
   |   |   |-- custom
   |   |   |-- data
   |   |   |-- dispatch
   |   |   |-- event
   |   |-- crosscut
   |   |   |-- i18n
   |   |   |-- log
   |   |   |-- trace
   |   |-- dsl
   |   |-- entity
   |   |-- exception
   |   |-- framework
   |   |   |-- bootstrap
   |   |   |-- config
   |   |-- server
   |   |-- test
   |   |   |-- ddt
   |   |   |   |-- mapping
   |   |   |   |   |-- bind
   |   |   |   |   |-- builder
   |   |   |   |   |-- io
   |   |   |   |   |-- mapping
   |   |   |   |   |-- type
   |   |   |   |-- validator
   |   |   |-- groovy
   |   |   |-- java
   |   |   |-- mock
   |   |   |-- report
   |   |-- ui
   |   |   |-- builder
   |   |   |-- locator
   |   |   |-- object
   |   |   |   |-- routing
   |   |   |-- widget
   |-- util
   |   |-- grid
```

- **CSS Selector**

One feedback we got from the Rich Web Experience 2009 [<http://code.google.com/p/aost/wiki/TelluriumAtRichWebExperience2009>] is that we should use the name "CSS selector" instead of "jQuery selector" because jQuery implements a subset of CSS selectors. As a result, we changed the following two methods in DslContext.

```
public void enablejQuerySelector();
public void disablejQuerySelector();
```

to

```
public void enableCssSelector();
public void disableCssSelector();
```

- **Group Locating (Santa Algorithm)**

Up to Tellurium 0.6.0, Tellurium still generates runtime locators such as XPath and CSS selector on the Tellurium Core side, then pass them to Selenium Core, which is basically still to locate one element in the UI module at a time. With the new Engine in Tellurium 0.7.0, the UI module will be located as a whole first, the subsequent calls will reuse the already located UI element in the DOM. This, called the Santa algorithm, is the missing half of the Tellurium UI module concept. The algorithm can locate the whole UI module at the runtime DOM. After that, you can just pass in UI element's UID to find it in the cached UI module on Tellurium Engine. That is to say, you don't need Tellurium Core to generate the runtime locators any more.

The magic is that the new Tellurium Engine will locate the UI module as a whole. It may have trouble to find the individual UI element, but it has no trouble to find the whole UI module structure in the DOM. Apart from that, Tellurium 0.7.0 also provides a handy method for you to validate your UI module. For example, if you call

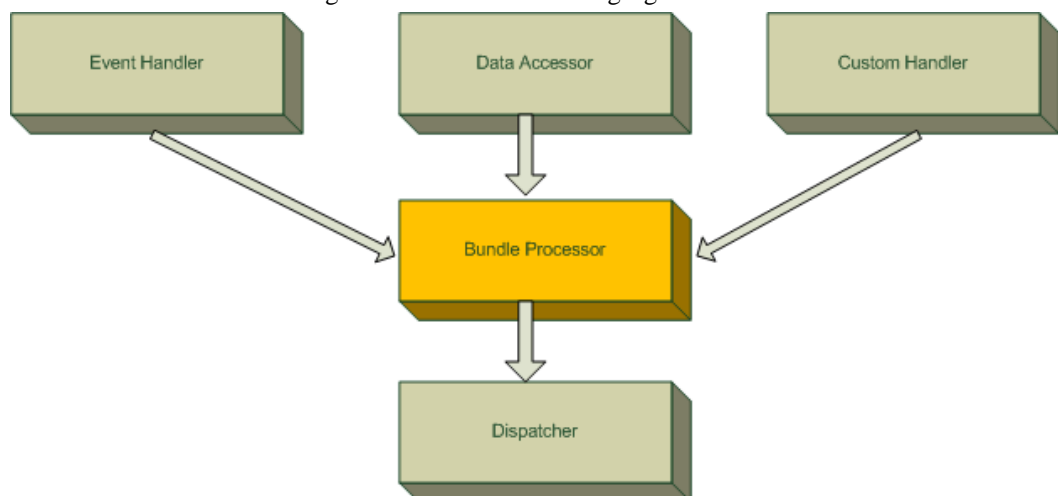
```
validate("ProblematicForm");
```

You will get the detailed validation results including the closest matches.

- **Macro command**

Macro Command is a set of Selenium commands that are bundled together and sent to Selenium Core in one call. This will reduce the round trip latency from Tellurium Core to Engine and thus, improve the speed performance. Another advantage for Macro Command is that Tellurium Engine can reuse the locator because many times the commands in the same bundle act on the same UI element or same sub-tree in the DOM.

To implement Macro Command, we added one more tier to Tellurium Core to automatically handle the Macro Command bundling as shown in the following figure.



- **UI Module Caching**

From Tellurium 0.6.0, Tellurium provides the cache capability for CSS selectors so that Tellurium can reuse them without doing re-locating. In 0.7.0, Tellurium moves a step further to cache the whole UI module on the Engine side. Each UI module cache holds a snapshot of the DOM references for the UI elements in the UI module. The exceptions are dynamic web elements defined by Tellurium UI templates. For these dynamic web elements, the Engine will try to get the DOM reference of its parent and then do locating inside this subtree with its parent node as the root, which will improve the locating speed a lot.

• Tellurium UID Description Language

The Tellurium UID Description Language (UDL) is designed to

1. address the dynamic factors in Tellurium UI templates
2. increase the flexibility of Tellurium UI templates.

UDL is implemented with the Antlr 3 parser generator. [<http://www.antlr.org/>] The implementation details can be found here. [<http://code.google.com/p/aost/wiki/BuildYourOwnJavaParserWithAntlr3/>] The grammars and technical details are recovered in Tellurium UID Description Language. [<http://code.google.com/p/aost/wiki/TelluriumUIDDescriptionLanguage/>]

To migrate your UI templates from 0.6.0 to 0.7.0, please make the following changes:

1. Put "{}" around 0.6.0 UI template uids
2. Replace "*" with "all"
3. Replace "all" in table with "row: all, column: all"
4. You can add an ID for each UI template object, where the ID starts with a letter and is followed by digits, letters, and "_".
5. The implicit tokens defined in UDL such as "header", "footer", "row", "column", and "tbody" are all reserved and they cannot be used as IDs. But UDL is case-sensitive, you can still use "Header", "Footer", "Row", "Column", and "TBody" as IDs.

For example the following TableExampleModule UI module in the ui-examples reference project.

```
class TableExampleModule extends DslContext {

    public void defineUi() {
        ui.StandardTable(uid: "GT", clocator: [id: "xyz"], ht: "tbody"){
            TextBox(uid: "header: all", clocator: [:])
            TextBox(uid: "row: 1, column: 1", clocator: [tag: "div", class: "abc"])
            Container(uid: "row: 1, column: 2"){
                InputBox(uid: "Input", clocator: [tag: "input", class: "123"])
                Container(uid: "Some", clocator: [tag: "div", class: "someclass"]){
                    Span(uid: "Span", clocator: [tag: "span", class: "x"])
                    UrlLink(uid: "Link", clocator: [:])
                }
            }
        }
    }

    public void work(String input){
        keyType "GT[1][2].Input", input
        click "GT[1][2].Some.Link"
        waitForPageToLoad 30000
    }
}
```

Now, this UI module can be re-defined in a more flexible way with UDL as follows.

```
class TableExampleModule extends DslContext {

    public void defineUi() {
        ui.StandardTable(uid: "GT", clocator: [id: "xyz"], ht: "tbody"){
            TextBox(uid: "{header: first} as One", clocator: [tag: "th", text: "one"], self: true)
            TextBox(uid: "{header: 2} as Two", clocator: [tag: "th", text: "two"], self: true)
            TextBox(uid: "{header: last} as Three", clocator: [tag: "th", text: "three"], self: true)
            TextBox(uid: "{row: 1, column -> One} as A", clocator: [tag: "div", class: "abc"])
            Container(uid: "{row: 1, column -> Two} as B"){
                InputBox(uid: "Input", clocator: [tag: "input", class: "123"])
                Container(uid: "Some", clocator: [tag: "div", class: "someclass"]){
                    Span(uid: "Span", clocator: [tag: "span", class: "x"])
                    UrlLink(uid: "Link", clocator: [:])
                }
            }
            TextBox(uid: "{row: 1, column -> Three} as Hello", clocator: [tag: "td"], self: true)
        }
    }

    public void work(String input){
        getText "GT.A"
        keyType "GT.B.Input", input
        click "GT.B.Some.Link"
        waitForPageToLoad 30000
    }
}
```

Apart from that, you need to add the UDL dependency to you project.

```
<dependency>
  <groupId>org.telluriumsource</groupId>
  <artifactId>tellurium-udl</artifactId>
  <version>0.7.0</version>
  <scope>compile</scope>
</dependency>
```

- **Tellurium New APIs**

Tellurium Engine in 0.7.0 re-implemented a set of Selenium APIs by exploiting jQuery, plus many more new APIs. For example,

```
TelluriumApi.prototype.getCSS = function(locator, cssName) {
    var element = this.cacheAwareLocate(locator);
    var out = [];
    var $e = tejQuery(element);
    $e.each(function() {
        out.push(tejQuery(this).css(cssName));
    });
    return JSON.stringify(out);
};
```

To switch between Tellurium new API and Selenium Core API, you can call the following method.

```
public void useTelluriumApi(boolean isUse);
```

Or use the following DSL methods from DslContext.

```
public void enableTelluriumApi();
public void disableTelluriumApi()
```

- **Trace**

Tellurium 0.7.0 provides built-in support for the command execution time including execution time for each command, total run time, and aggregated times for each command.

We added the following settings to TelluriumConfig.groovy

```
test{
    execution{
        //whether to trace the execution timing
        trace = false
    }
}
```

You can use the follow methods in DslContext to turn on or off the trace, and get the trace data.

```
public void enableTrace();
public void disableTrace();
public void showTrace();
```

- **Methods Accessible in Test Cases**

There are many Tellurium APIs that used to be available only in DslContext. That is to say, you have to extend DslContext to use them. For example, you often see code in a test case likes this,

```
GoogleSearchModule gsm = new GoogleSearchModule();
gsm.defineUi();
gsm.usejQuerySelector();
gsm.registerNamespace("te", te_ns);
```

Now, many of them, which are not really tied to a specific UI module, are made available in Tellurium test cases. For example, the above code can be changed as follows,

```
GoogleSearchModule gsm = new GoogleSearchModule();
gsm.defineUi();
useCssSelector();
registerNamespace("te", te_ns);
```

- **Environment**

We added an Environment class to Tellurium Core so that you can change the runtime environment.

- **Get UIs by Tag Name**

As requested by users, Tellurium 0.7.0 added the following two methods.

```
UiByTagResponse getUiByTag(String tag, Map filters);  
void removeMarkedUids(String tag);
```

The first method passes in the tag name and the attributes as filters and get back the UI elements associated with the tag. The second method cleans up all the temporally assigned IDs by the Tellurium Engine.

- **DIV object change**

The Div object has been changed to be a Container type object. For example, you can define the following UI module.

```
ui.Div(uid: "div1", clocator: [id: "div1"]) {  
    Div(uid: "div2", clocator: [id: "div2"]) {  
        List(uid: "list1", clocator: [tag: "ul"], separator: "li"){  
            UrlLink(uid: "{all}", clocator: [:])  
        }  
    }  
}
```

- **Selector changes**

For the Selector UI object, the following DSL methods have been implemented in the Tellurium new Engine:

```
* select(String uid, String target): Equals to selectByLabel.  
* selectByLabel(String uid, String target): Select based on the text attribute.  
* selectByValue(String uid, String target): Select based on the value attribute.  
* selectByIndex(String uid, int target): Select based on index (starting from 1).  
* selectById(String uid, String target): Select based on the ID attribute.  
* String[] getSelectOptions(String uid): Get all option texts.  
* String[] getSelectValues(String uid): Get all option values.  
* String[] getSelectedLabels(String uid): Get selected texts.  
* String getSelectedLabel(String uid): Get selected text, for multiple selections,  
    return the first one.  
* String[] getSelectedValues(String uid): Get selected values.  
* String getSelectedValue(String uid): Get selected value, for multiple selections,  
    return the first one.  
* addSelectionByLabel(String uid, String target): Add selection based on text.  
* addSelectionByValue(String uid, String target): Add selection based on value.  
* removeSelectionByLabel(String uid, String target): Remove selection based on text.  
* removeSelectionByValue(String uid, String target): Remove selection based on value.  
* removeAllSelections(String uid): Remove all selections.
```

- **Toggle**

Tellurium 0.7.0 provides a toggle method to animate the UI element on the web page. For example, you can the following commands to show the UI element under testing.

```
toggle "Form.Username.Input"  
pause 500  
toggle "Form.Username.Input"
```

- **UI Module Live Show**

The show command is used to show the UI module that you defined on the actual web page.

```
public show(String uid, int delay);
```

where delay is in milliseconds. Be aware that show is available in Tellurium API. Example:

```
useCache(true);
useTelluriumApi(true);
JettyLogonModule jlm = new JettyLogonModule();
jlm.show("Form", 5000);
```

The UI module on the web page is outlined and if a user hives over the UI module, the UIDs of the selected UI element's and its ancestors' are shown as a tooltip.

- **getHTMLSource**

Use getHTMLSource, users can get back the runtime html source for a UI module. Tellurium provided two methods for this purpose. .

```
public Map getHTMLSourceResponse(String uid);
public void getHTMLSource(String uid);
```

The first method get back the html source as a uid-to-html map and the second one simply print out the html source on the console. Be aware, getHTMLSource is only available in Tellurium new APIs.

- **type and keyType**

type and keyType now accept different types of objects and convert them to a String automatically by calling the toString() method. For example, you can use the following commands:

```
type "Google.Input", "Tellurium"
type "Google.Input", 12.15
type "Google.Input", true
```

- **i18n Support**

Tellurium now provides support for internationalization of strings and exception messages. Internationalized strings for each locale is provided through a ResourceBundle for a specific locale which is of the format

```
<MessageBundleName>_<language-code>_<country code>.properties.
```

- **Excel File Reader in Data Driven Test**

Excel file reader has been added to Tellurium Data Driven Test. If you have excel input files, you can specify the reader in the configuration file TelluriumConfig.groovy as follows,

```
datadriven{
    dataprovider{
        //specify which data reader you like the data provider to use
        //the valid options include "PipeFileReader", "CVSFileReader", and "ExcelFileReader" at this
        reader = "ExcelFileReader"
    }
}
```

```
}
```

- **Engine State Offline Update**

For some reason, you may need to make Engine state calls before you actually connect to the Engine. We added an Engine State tracer in the bundle tier to record all the requests if the Engine is not connected and aggregate them. Once the Engine is connected, Tellurium will automatically send out the Engine state update request.

- **Repeat Object**

Very often, we need to use the same UI elements for multiple times on the web page. We can use the Repeat object for this purpose. The Repeat object inherits the Container object and You can use it just like a Container. The difference is that you should use index to refer to a Repeat object.

- **All Purpose Object**

Tellurium 0.7.0 added an all purpose object for internal usage only. This object includes all methods for non-Container type objects.

- **Groovy and GMaven**

Groovy has been upgraded to the latest version 1.7.0. For GMaven, we use the latest version 1.2 and removed the Maven Antrun plugin.

```
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-all</artifactId>
  <version>1.7.0</version>
</dependency>
```

- **JUnit**

JUnit is upgraded to 4.7 since it provides more features. One such a good feature is the Rule annotation. To be consistent with TestNG test case, the class TelluriumJavaTestCase is deprecated and you should use TelluriumJUnitTestCase instead.

- **TestNG**

TelluriumTestNGTestCase was changed to allow the setup and teardown procedures only work once for all the tests. The magic are the @BeforeTest and @AfterTest annotations. See the following code for more details,

```
public abstract class TelluriumTestNGTestCase extends BaseTelluriumJavaTestCase {

    @BeforeTest(alwaysRun = true)
    public static void setUpForTest() {
        tellurium = TelluriumSupport.addSupport();
        tellurium.start(customConfig);
        connector = (SeleniumConnector) tellurium.getConnector();
    }

    @AfterTest(alwaysRun = true)
    public static void tearDownForTest() {
        if(tellurium != null)
            tellurium.stop();
    }
}
```

- **TelluriumMockJUnitTestCase and TelluriumMockTestNGTestCase**

TelluriumMockJUnitTestCase and TelluriumMockTestNGTestCase incorporated the Mock Http Server so that you can load up a html web page and test against it with minimal configuration.

- **Tellurium Configuration**

The configuration parser has been refactored. The configuration file name is stored in the Environment class as follows.

```
@Singleton
public class Environment implements Configurable{

    protected String configFileFileName = "TelluriumConfig.groovy";

    protected String configString = "";

    public static Environment getEnvironment(){
        return Environment.instance;
    }

    public void useConfigString(String json){
        this.configString = json;
    }
    ...
}
```

That is to say, you can get the Environment singleton instance and change the file name before Tellurium core is loaded up if you have a good reason to do that. In the meanwhile, we add support to load the configuration from a JSON string, which will be stored in the configString variable in the Environment.

- **Run DSL Script**

Tellurium 0.7.0 provides a rundsl.groovy script for users to run DSL test script. The rundsl.groovy uses Groovy Grape to automatically download all dependencies and then run DSL script.

- **useTelluriumEngine**

To make it convenient for users, Tellurium provides a useTelluriumEngine command as follows,

```
void useTelluriumEngine(boolean isUse){
    useCache(isUse);
    useMacroCmd(isUse);
    useTelluriumApi(isUse);
}
```

As you can see, this command actually consists of three commands. In DslContext, Tellurium also provides two handy DSL style methods.

```
void enableTelluriumEngine();
void disableTelluriumEngine();
```

- **Selenium RC**

Selenium RC is 1.0.1 in Tellurium 0.7.0. Please use the following Maven dependencies.

```
<dependency>
    <groupId>org.seleniumhq.selenium.server</groupId>
    <artifactId>selenium-server</artifactId>
    <version>${selenium-server-version}</version>
</dependency>
```

```
<!--classifier>standalone</classifier-->
<exclusions>
  <exclusion>
    <groupId>ant</groupId>
    <artifactId>ant</artifactId>
  </exclusion>
</exclusions>
</dependency>
<dependency>
  <groupId>org.seleniumhq.selenium.client-drivers</groupId>
  <artifactId>selenium-java-client-driver</artifactId>
  <version>${selenium-version}</version>
  <exclusions>
    <exclusion>
      <groupId>org.codehaus.gmaven.runtime</groupId>
      <artifactId>gmaven-runtime-default</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.seleniumhq.selenium.core</groupId>
      <artifactId>selenium-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.seleniumhq.selenium.server</groupId>
      <artifactId>selenium-server</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

where the properties are defined as

```
<properties>
  ...
  <selenium-version>1.0.1</selenium-version>
  <selenium-server-version>1.0.1-te3</selenium-server-version>
</properties>
```

- **Miscellaneous Changes**

The default locator in Tellurium Engine is CSS selector by default. That is to say, we set

```
exploitCssSelector = true
```

in the `Environment` class and you can use

```
disableCssSelector()
```

or

```
useCssSelector(false)
```

to switch back to xpath if you like to.

Exposed the following Selenium APIs to `DslContext`.

```
* void addScript(String scriptContent, String scriptTagId)
* void removeScript(String scriptTagId)
* void captureEntirePageScreenshot(String filename, String kwargs)
* String captureScreenshotToString()
```

```
* String captureEntirePageScreenshotToString(String kwargs)
* String retrieveLastRemoteControlLogs()
```

The following methods in DslContext have been renamed

```
* String jsonify(String uid) is renamed to String toString(String uid)
* String generateHtml(String uid) is renamed to String toHTML(String uid) )
```

Changes in Engine

Tellurium 0.7.0 include a new Engine embedded in Selenium Core. The main functionalities of the Tellurium Engine include

- CSS Selector support based on jQuery
- New APIs based on jQuery
- UI module group locating
- UI module Caching

Other changes include:

- **jQuery**

The jQuery in Engine has been upgraded from 1.3.2 to 1.4.2.

- **Engine Logging**

Tellurium Engine uses Firebug Lite to add debug information to the console. By default the Firebug Lite is off.

- **Add JavaScript Error Stack to Selenium Errors**

We utilized the JavaScript Stack Trace project to refactor Selenium Errors in Selenium Core so that the JavaScript Error Stack will be passed back Tellurium Core.

Changes in Maven Build

Our Maven repository is moved and the Maven Repositories are changed as well.

```
<repository>
  <id>kungfuters-public-releases-repo</id>
  <name>Kungfuters.org Public Releases Repository</name>
  <url>http://maven.kungfuters.org/content/repositories/releases</url>
</repository>
<snapshotRepository>
  <id>kungfuters-public-snapshots-repo</id>
  <name>Kungfuters.org Public Snapshot Repository</name>
  <url>http://maven.kungfuters.org/content/repositories/snapshots</url>
</snapshotRepository>
```

telluriumsource.org becomes our official domain name and the web site is under construction.

```
<site>
  <id>tellurium-site</id>
```

```
<name>Tellurium Site</name>
<url>scp://telluriumsource.org/var/www/telluriumsource.org/public/maven</url>
</site>
```

After the domain name change, you need to use the following Tellurium dependencies for your project

```
<dependency>
  <groupId>org.telluriumsource</groupId>
  <artifactId>tellurium-core</artifactId>
  <version>0.7.0</version>
</dependency>
<dependency>
  <groupId>org.seleniumhq.selenium.server</groupId>
  <artifactId>selenium-server</artifactId>
  <version>1.0.1-te3</version>
</dependency>
```

For Maven archetypes, tellurium-junit-archetype becomes the following,

```
<groupId>org.telluriumsource</groupId>
<artifactId>tellurium-junit-archetype</artifactId>
<version>0.7.0</version>
```

Similarly, the tellurium-testng-archetype has been changed to

```
<groupId>org.telluriumsource</groupId>
<artifactId>tellurium-testng-archetype</artifactId>
<version>0.7.0</version>
```

To create a Tellurium project based on Tellurium 0.7.0, you should use the Maven archetype 0.7.0. To create a JUnit project, use the following Maven command:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id
-DarchetypeArtifactId=tellurium-junit-archetype -DarchetypeGroupId=org.telluriumsource
-DarchetypeVersion=0.7.0
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

Similarly, to create a TestNG project, use the following command:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id
-DarchetypeArtifactId=tellurium-testng-archetype -DarchetypeGroupId=org.telluriumsource
-DarchetypeVersion=0.7.0
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

To create a Tellurium UI widget project, we can use Tellurium Widget archetype as follows.

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id
-DarchetypeArtifactId=tellurium-widget-archetype -DarchetypeGroupId=org.telluriumsource
-DarchetypeVersion=0.7.0
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

Reference Project

The old two reference projects, i.e., tellurium-junit-java and tellurium-testng-java reference projects have been updated and merged into one reference project, tellurium-website. In addition, we add a new ui-examples reference project to show how to define different types of UI objects based on html snippets.

How to Obtain Tellurium 0.7.0

Tellurium 0.7.0 bundle can be downloaded from

- “Tellurium download site [<http://code.google.com/p/aost/downloads/list>]”

Tellurium 0.8.0-SNAPSHOT can be obtained from

- “Tellurium Core SNAPSHOT [<http://maven.kungfuters.org/content/repositories/snapshots/org/telluriumsource/tellurium-core/0.8.0-SNAPSHOT/>]”
- “Custom Selenium Server with Tellurium Engine [<http://maven.kungfuters.org/content/repositories/snapshots/org/seleniumhq/selenium/server/selenium-server/1.0.1-te4-SNAPSHOT/>]”

If you use Maven, you need the following dependencies

```
<dependency>
  <groupId>org.telluriumsource</groupId>
  <artifactId>tellurium-udl</artifactId>
  <version>0.7.0</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.telluriumsource</groupId>
  <artifactId>tellurium-core</artifactId>
  <version>0.7.0</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.seleniumhq.selenium.server</groupId>
  <artifactId>selenium-server</artifactId>
  <version>1.0.1-te3</version>
</dependency>
```

A sample POM file can be found in appendices.

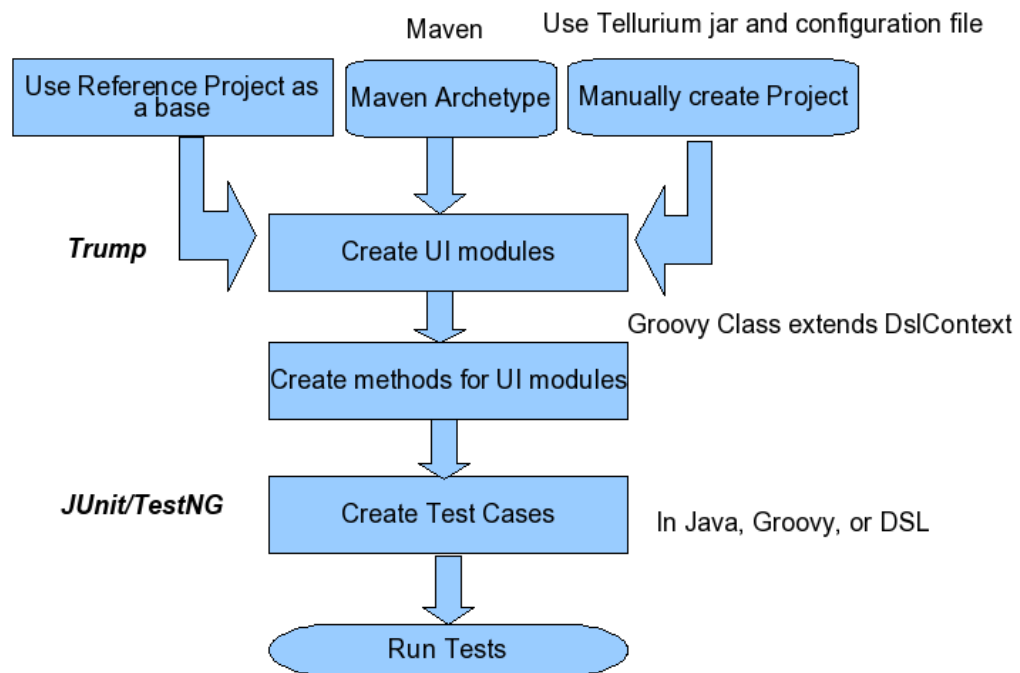
Chapter 3. Getting Started

This chapter discusses the Tellurium methods for creating a Tellurium project followed by descriptions of the primary components used for testing the newly created web framework including the UI Module, UI Object and attributes, Logical Container, jQuery Selector, UI Templates, and UI Testing Support.

Create a Tellurium Project

Create a Tellurium Project in one of the following methods:

- Use a Tellurium reference project as a base
- Use Tellurium Maven archetypes [http://code.google.com/p/aost/wiki/TelluriumMavenArchetypes]
- Manually create a Tellurium project using the tellurium jars [http://code.google.com/p/aost/downloads/list] and a Tellurium configuration file [http://code.google.com/p/aost/wiki/TelluriumSampleConfigurationFile]
- Alternatively, create a Tellurium Maven project manually using the sample POM file [http://code.google.com/p/aost/wiki/TelluriumTestProjectMavenSamplePom].



Tellurium Maven Archetypes

The easiest way to create a Tellurium project is to use Tellurium Maven archetypes. Tellurium provides two Maven archetypes for Tellurium JUnit test projects and Tellurium TestNG test projects respectively.

- tellurium-junit-archetype
- tellurium-testng-archetype

As a result, a user can create a Tellurium project using one Maven command. For example, for a Tellurium JUnit project, use:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-junit-archetype \
-DarchetypeGroupId=org.telluriumsource \
-DarchetypeVersion=0.7.0 \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/releases
```

Similarly, for a Tellurium TestNG project, use:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-testng-archetype \
-DarchetypeGroupId=org.telluriumsource \
-DarchetypeVersion=0.7.0 \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/releases
```

Tellurium Ant Projects

For an Ant user:

1. Download Tellurium 0.7.0 Release package from the Tellurium project download page [<http://code.google.com/p/aost/downloads/list>]
2. Unpack the Tellurium 0.7.0 Release package and copy dependencies to your project /lib directory together with the tellurium-core and tellurium-udl 0.7.0 jar files.
3. Name the Tellurium configuration file as TelluriumConfig.groovy and place it in the project root directory.

For Ant build scripts, refer to the sample Tellurium Ant build scripts [<http://code.google.com/p/aost/wiki/TelluriumSampleAntBuildScript>].

Setup Tellurium Project in IDEs

A Tellurium Project can be run in IntelliJ, NetBeans, Eclipse, or other IDEs that have Groovy support.

If using Maven, open the POM file to let the IDE automatically build the project files.

IntelliJ IDEA

IntelliJ IDEA Community edition is free and can be downloaded from <http://www.jetbrains.com/idea/download/> [<http://www.jetbrains.com/idea/download/>]. A detailed guide is found on [How to create your own Tellurium testing project with IntelliJ 9.0 Community Edition](http://code.google.com/p/aost/wiki/CustomTelluriumIntelliJProject) [<http://code.google.com/p/aost/wiki/CustomTelluriumIntelliJProject>].

NetBeans

For NetBeans users, detailed Guides can be found on [the NetBeans Starters' guide page](http://code.google.com/p/aost/wiki/TelluriumStarterUsingNetBeans) [<http://code.google.com/p/aost/wiki/TelluriumStarterUsingNetBeans>] and [How to create your own Tellurium testing project with NetBeans 6.5](http://code.google.com/p/aost/wiki/CustomTelluriumNetBeansProject) [<http://code.google.com/p/aost/wiki/CustomTelluriumNetBeansProject>].

Eclipse

For Eclipse users, download the Eclipse Groovy Plugin from <http://dist.codehaus.org/groovy/distributions/update/> [<http://dist.codehaus.org/groovy/distributions/update/>] to run the Tellurium project.

For detailed instructions, read [How to create your own Tellurium testing project with Eclipse](http://code.google.com/p/aost/wiki/CustomTelluriumEclipseProject) [http://code.google.com/p/aost/wiki/CustomTelluriumEclipseProject].

Create a UI Module

Tellurium provides TrUMP to automatically create UI modules. TrUMP can be downloaded from the Tellurium project site:

<http://code.google.com/p/aost/downloads/list> [http://code.google.com/p/aost/downloads/list]

Choose the Firefox 2 or Firefox 3 version depending upon the user's Firefox version, or download the Firefox 3 version directly from the Firefox addons site at:

<https://addons.mozilla.org/en-US/firefox/addon/11035> [https://addons.mozilla.org/en-US/firefox/addon/11035]

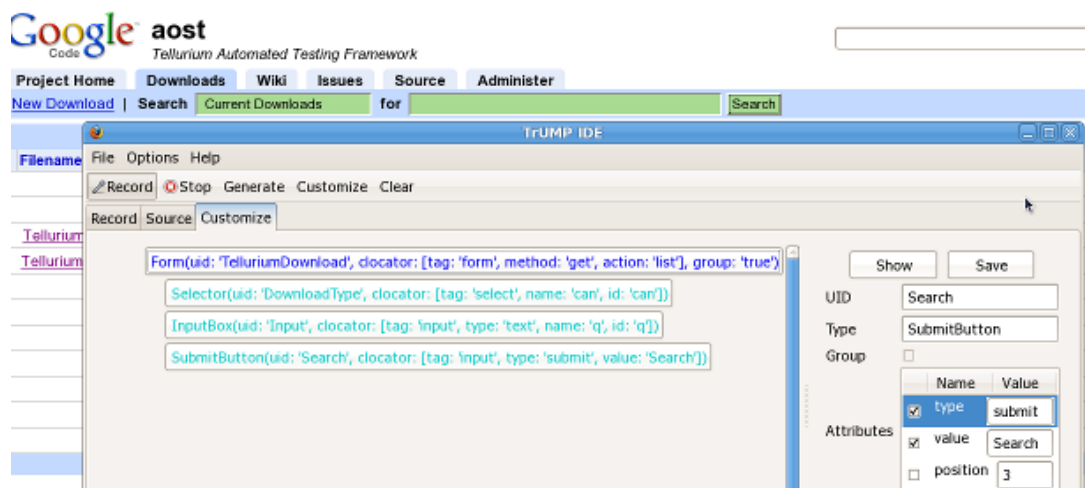
Once installed, restart Firefox. Record UI modules by simply clicking UI elements on the web and then click the "Generate" button.

To customize the UI, click the "Customize" button.

In the example, open the Tellurium download page found on:

<http://code.google.com/p/aost/downloads/list> [http://code.google.com/p/aost/downloads/list]

Record the download search module as follows:



After the UI module is customized, export it as the module file `NewUiModule.groovy` to the demo project and add a couple of methods to the class:

```
class NewUiModule extends DslContext {

    public void defineUi() {
        ui.Form(uid: "TelluriumDownload", clocator: [tag: "form", method: "get",
            action: "list"], group: "true")
        {
            Selector(uid: "DownloadType", clocator: [tag: "select", name: "can", id: "can"])
            InputBox(uid: "Input", clocator: [tag: "input", type: "text", name: "q", id: "q"])
            SubmitButton(uid: "Search", clocator: [tag: "input", type: "submit",
                value: "Search"])
        }
    }

    public void searchDownload(String keyword) {
        keyType "TelluriumDownload.Input", keyword
        click "TelluriumDownload.Search"
    }
}
```

```
        waitForPageToLoad 30000
    }

    public String[] getAllDownloadTypes() {
        return getSelectOptions("TelluriumDownload.DownloadType")
    }

    public void selectDownloadType(String type) {
        selectByLabel "TelluriumDownload.DownloadType", type
    }
}
```

Create Tellurium Test Cases

Once the UI module is created, create a new Tellurium test case `NewTestCase` by extending `TelluriumJUnitTestCase` class

```
public class NewTestCase extends TelluriumJUnitTestCase {
    private static NewUiModule app;

    @BeforeClass
    public static void initUi() {
        app = new NewUiModule();
        app.defineUi();
    }

    @Before
    public void setUpForTest() {
        connectUrl("http://code.google.com/p/aost/downloads/list");
    }

    @Test
    public void testTelluriumProjectPage() {
        String[] allTypes = app.getAllDownloadTypes();
        assertNotNull(allTypes);
        assertTrue(allTypes[1].contains("All Downloads"));
        app.selectDownloadType(allTypes[1]);
        app.searchDownload("TrUMP");
    }
}
```

Compile the project and run the new test case.

Chapter 4. Tellurium UI Objects

The UI Module is the heart of Tellurium. The UI module is a collection of UI elements grouped together. Usually, the UI module represents a composite UI object in the format of nested basic UI elements. For example, the download search module in Tellurium's project site is defined as follows:

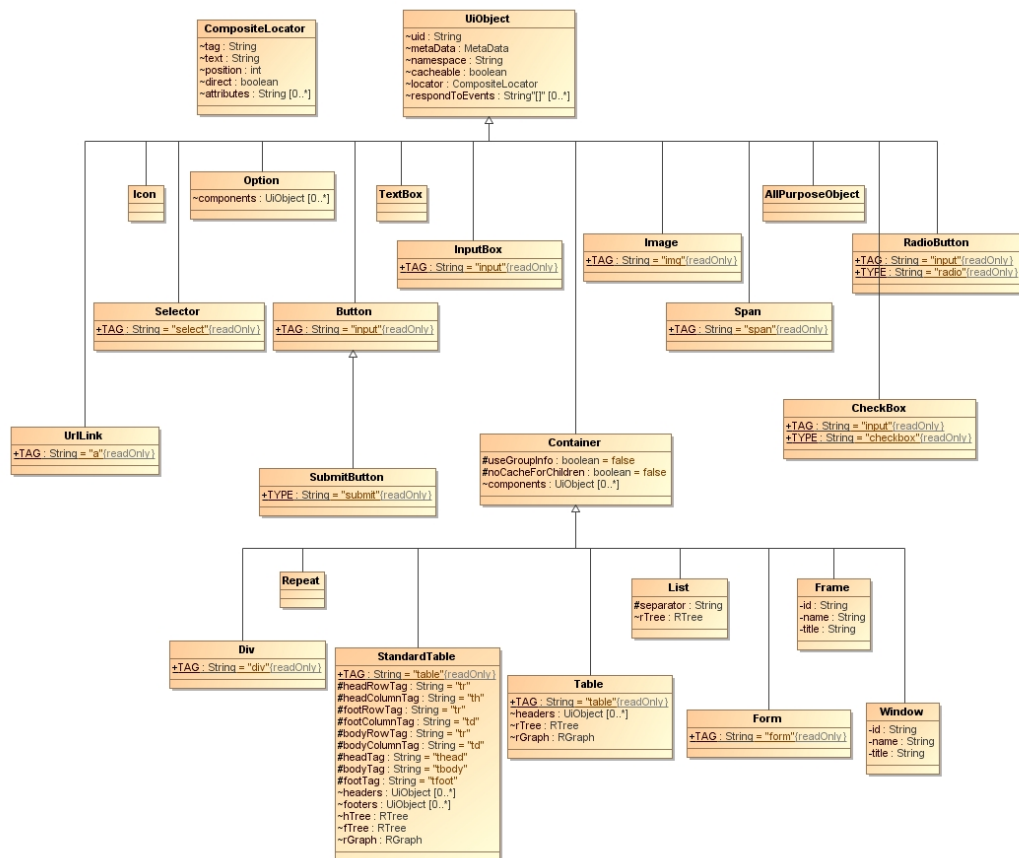
```
ui.Form(uid: "downloadSearch", clocator: [action: "list", method: "get"], group: "true") {  
  Selector(uid: "downloadType", clocator: [name: "can", id: "can"])  
  InputBox(uid: "searchBox", clocator: [name: "q"])  
  SubmitButton(uid: "searchButton", clocator: [value: "Search"])  
}
```

Tellurium is built on the foundation of the UI module. The UI module makes it possible to build locators for UI elements at runtime. First, this makes Tellurium robust and responsive to changes from internal UI elements. Second, the UI module makes Tellurium expressive.

A UI element is referred to simply by appending the names (uids) along the path to the specific element. This also enables Tellurium's "Group Locating" feature, making composite objects reusable and addressing dynamic web pages.

This frees up the testers to write better tests rather than spending precious testing time identifying and resolving test failures due to XPath changes.

The following class diagram illustrates the relationship among UI objects.



Basic UI Object

Tellurium provides a set of predefined UI objects to be used directly when setting up a test.

The basic UI object is an abstract class. Users cannot instantiate it directly. The basic UI Object works as the base class for all UI objects and it includes the following attributes:

Table 4.1. UI Object Attributes

ATTRIBUTE	DESCRIPTION
UI Object	Basic Tellurium component
UiID	UI object's identifier
Namespace	Used for XHTML
Locator	UI Object Locator including Base Locator and Composite Locator
Group	Group Locating attribute that only applies to a collection type of UI object such as Container, Table, List, Form
Respond	JavaScript events the UI object responds to

The value is a list and the base UI object provides the following methods:

- boolean isElementPresent()
- boolean isVisible()
- boolean isDisabled()
- waitForElementPresent(int timeout), where the time unit is ms.
- waitForElementPresent(int timeout, int step)
- String getText()
- getAttribute(String attribute)

All UI Objects inherit the above attributes and methods. Do not call these methods directly but use DSL syntax instead.

For example, use:

```
click "GoogleSearchModule.Search"
```

In this way, Tellurium first maps the UiID "GoogleSearchModule.Search" to the actual UI Object. Then the user calls the click method on the UI Object. If that UI Object does not have the click method defined, an error displays.

The UML class diagram is shown as follows.

UiObject		
f	locator	Locator
f	namespace	String
f	respondToEvents	String[]
f	uid	String
<hr/>		
m	customMethod()	Object
m	diagnose()	void
m	dragAndDrop(String)	void
m	generateHtml()	String
m	getAttribute(String)	String
m	getCSS(String)	String[]
m	getSelector()	String
m	getText()	String
m	getXPath()	String
m	hasCssClass()	boolean
m	isDisabled()	boolean
m	isElementPresent()	boolean
m	isVisible()	boolean
m	mouseOut()	void
m	mouseOver()	void
m	waitForElementPresent(int)	boolean
m	waitForElementPresent(int, int)	boolean
m	waitForText(int)	String
m	walkTo(WorkflowContext, UiID)	UiObject

UI Object Default Attributes

Tellurium UI objects have default attributes as shown in the following table

Table 4.2. UI Object Default Attributes

Tellurium Object	Locator Attributes	Default	Extra Attributes	UI Template
Button	tag: "input"			no
Container			group	no
CheckBox	tag: "input", type: "checkbox"			no
Div	tag: "div"			no
Form	tag: "form"		group	no
Image	tag: "img"			no
InputBox	tag: "input"			no

Tellurium Object	Locator Attributes	Default	Extra Attributes	UI Template
RadioButton	tag: "input", type: "radio"			no
Selector	tag: "select"			no
Span	tag: "span"			no
SubmitButton	tag: "input", type: "submit"			no
UrlLink	tag: "a"			no
Repeat				no
List			separator	yes
Table	tag: "table"		group, header	yes
StandardTable	tag: "table"		group, header, footer	yes
Frame			group, id, name, title	no
Window			group, id, name, title	no

UI Object Description

Button

Button represents various Buttons on the web and its default tag is "input". The following methods can be applied to Button:

- click()
- doubleClick()
- clickAt(String coordination)

Example:

```
Button(uid: "searchButton", clocator: [value: "Search", name: "btn"])
```

Submit Button

SubmitButton is a special Button with its type being "submit".

Example:

```
SubmitButton(uid: "search_web_button", clocator: [value: "Search the Web"])
```

Check Box

The CheckBox on the web is abstracted as "CheckBox" Ui object. The default tag for CheckBox is "input" and its type is "checkbox". CheckBox comes with the following methods:

- check()
- boolean isChecked()

- `uncheck()`
- `String getValue()`

Example:

```
CheckBox(uid: "autoRenewal", clocator: [dojoattachpoint: 'dap_auto_renew'])
```

Div

Div is often used in the Dojo framework and it can represent many objects. Its tag is "div" and it has the following method:

- `click()`

Example:

```
Div(uid: "dialog", clocator: [class: 'dojoDialog', id: 'loginDialog'])
```

The Div object has been changed to be a Container type object. For example, you can define the following UI module.

```
ui.Div(uid: "div1", clocator: [id: "div1"]) {  
  Div(uid: "div2", clocator: [id: "div2"]) {  
    List(uid: "list1", clocator: [tag: "ul"], separator: "li"){  
      UrlLink(uid: "{all}", clocator: [:])  
    }  
  }  
}
```

Image

Image is used to abstract the "img" tag and it comes with the following additional methods:

- `getImageSource()`
- `getImageAlt()`
- `String getImageTitle()`

Example:

```
Image(uid: "dropDownArrow", clocator: [src: 'drop_down_arrow.gif'])
```

Icon

Icon is similar to the Image object, but user can perform actions on it. As a result, it can have the following additional methods:

- `click()`
- `doubleClick()`

- `clickAt(String coordination)`

Example:

```
Icon(uid: "taskIcon", clocator:[tag: "p", dojoonclick: 'doClick', img: "Show_icon.gif"])
```

Radio Button

RadioButton is the abstract object for the Radio Button Ui. As a result, its default tag is "input" and its type is "radio". RadioButton has the following additional methods:

- `check()`
- `boolean isChecked()`
- `uncheck()`
- `String getValue()`

Example:

```
RadioButton(uid: "autoRenewal", clocator: [dojoattachpoint: 'dap_auto_renew'])
```

Text Box

TextBox is the abstract Ui object from which the user returns to the text. For example, it comes with the following method:

- `String waitForText(int timeout)`

Note: TextBox can have various types of tags.

Example:

```
TextBox(uid: "searchLabel", clocator: [tag: "span"])
```

Input Box

InputBox is the Ui where user types in input data. As its name stands, InputBox's default tag is "input". InputBox has the following additional methods:

- `type(String input)`
- `keyType(String input)`, used to simulate keyboard typing
- `typeAndReturn(String input)`
- `clearText()`
- `boolean isEditable()`
- `String getValue()`

Example:

```
InputBox(uid: "searchBox", clocator: [name: "q"])
```

URL Link

UrlLink stands for the web url link, i.e., its tag is "a". UrlLink has the following additional methods:

- String getLink()
- click()
- doubleClick()
- clickAt(String coordination)

Example:

```
UrlLink(uid: "Grid", clocator: [text: "Grid", direct: "true"])
```

Repeat Object

Very often, we need to use the same UI elements for multiple times on the web page. For example, we have the following html.

```
<div class="segment clearfix">
  <div class="option">
    <ul class="fares">
      <li>
        <input type="radio">&nbsp;   
        <label>Economy</label>
      </li>
      <li>
        <input type="radio">&nbsp;   
        <label>Flexible</label>
      </li>
    </ul>
    <div class="details">
      <dl>
        <dt>Ship:</dt>
        <dd>A</dd>
        <dt>Departs</dt>
        <dd>
          <em>08:00</em>
        </dd>
        <dt>Arrives</dt>
        <dd>
          <em>11:45</em>
        </dd>
      </dl>
    </div>
  </div>
  <div class="option">
    <ul class="fares">
      <li>
        <input type="radio">&nbsp;   
        <label>Economy</label>
      </li>
      <li>
        <input type="radio">&nbsp;   
        <label>Flexible</label>
      </li>
    </ul>
    <div class="details">
      <dl>
        <dt>Ship:</dt>
```

```

        <dd>B</dd>
        <dt>Departs</dt>
        <dd>
            <em>17:30</em>
        </dd>
        <dt>Arrives</dt>
        <dd>
            <em>21:15</em>
        </dd>
    </dl>
</div>
</div>
</div>
<div class="segment clearfix">
    <div class="option">
        <ul class="fares">
            <li>
                <input type="radio">&nbsp;
                <label>Economy</label>
            </li>
            <li>
                <input type="radio">&nbsp;
                <label>Flexible</label>
            </li>
        </ul>
        <div class="details">
            <div class="photo"><img/></div>
            <dl>
                <dt>Ship:</dt>
                <dd>C</dd>
                <dt>Departs</dt>
                <dd>
                    <em>02:00</em>
                </dd>
                <dt>Arrives</dt>
                <dd>
                    <em>06:00</em>
                </dd>
            </dl>
        </div>
    </div>
    <div class="option">
        <ul class="fares">
            <li>
                <input type="radio">&nbsp;
                <label>Economy</label>
            </li>
            <li>
                <input type="radio">&nbsp;
                <label>Flexible</label>
            </li>
        </ul>
        <div class="details">
            <dl>
                <dt>Ship:</dt>
                <dd>D</dd>
                <dt>Departs</dt>
                <dd>
                    <em>12:45</em>
                </dd>
                <dt>Arrives</dt>
                <dd>
                    <em>16:30</em>
                </dd>
            </dl>
        </div>
    </div>
</div>
</form>

```

You can see the elements `<div class="segment clearfix">` and `<div class="option">` repeated for couple times. We can use the Repeat object for this UI module. The Repeat object inherits the Container object and You can use it just like a Container. The difference is that you should use index to refer to a Repeat object.

For the above html source, we can create the following UI module

```

ui.Form(uid: "SailingForm", clocator: [name: "selectedSailingsForm"] ){
  Repeat(uid: "Section", clocator: [tag: "div", class: "segment clearfix"]){
    Repeat(uid: "Option", clocator: [tag: "div", class: "option",
      direct: "true"]){
      List(uid: "Fares", clocator: [tag: "ul", class: "fares", direct: "true"],
        separator: "li"){
        Container(uid: "all"){
          RadioButton(uid: "radio", clocator: [:], respond: ["click"])
          TextBox(uid: "label", clocator: [tag: "label"])
        }
      }
    }
    Container(uid: "Details", clocator: [tag: "div", class: "details"]){
      Container(uid: "ShipInfo", clocator: [tag: "dl"]){
        TextBox(uid: "ShipLabel", clocator: [tag: "dt", position: "1"])
        TextBox(uid: "Ship", clocator: [tag: "dd", position: "1"])
        TextBox(uid: "DepartureLabel", clocator: [tag: "dt", position: "2"])
        Container(uid: "Departure", clocator: [tag: "dd", position: "2"]){
          TextBox(uid: "Time", clocator: [tag: "em"])
        }
        TextBox(uid: "ArrivalLabel", clocator: [tag: "dt", position: "3"])
        Container(uid: "Arrival", clocator: [tag: "dd", position: "3"]){
          TextBox(uid: "Time", clocator: [tag: "em"])
        }
      }
    }
  }
}

```

To test the UI module, we can create the following test case.

```

@Test
public void testRepeat(){
  connect("JForm");
  int num = jlm.getRepeatNum("SailingForm.Section");
  assertEquals(2, num);
  num = jlm.getRepeatNum("SailingForm.Section[1].Option");
  assertEquals(2, num);
  int size = jlm.getListSize("SailingForm.Section[1].Option[1].Fares");
  assertEquals(2, size);
  String ship = jlm.getText("SailingForm.Section[1].Option[1].
    Details.ShipInfo.Ship");
  assertEquals("A", ship);
  String departureTime = jlm.getText("SailingForm.Section[1].Option[1].
    Details.ShipInfo.Departure.Time");
  assertEquals("08:00", departureTime);
  String arrivalTime = jlm.getText("SailingForm.Section[1].Option[1].
    Details.ShipInfo.Arrival.Time");
  assertEquals("11:45", arrivalTime);
}

```

Selector

Selector represents the Ui with tag "select". The user can select from a set of options. Selector has methods, such as:

- selectByLabel(String target)
- selectByValue(String value)
- addSelectionByLabel(String target)
- addSelectionByValue(String value)

- `removeSelectionByLabel(String target)`
- `removeSelectionByValue(String value)`
- `removeAllSelections()`
- `String getSelectOptions()`
- `String getSelectedLabels()`
- `String getSelectedLabel()`
- `String getSelectedValues()`
- `String getSelectedValue()`
- `String getSelectedIndexes()`
- `String getSelectedIndex()`
- `String getSelectedIds()`
- `String getSelectedId()`
- `boolean isSomethingSelected()`

Example:

```
Selector(uid: "issueType", clocator: [name: "can", id: "can"])
```

For the Selector UI object, the following DSL methods have been implemented in the Tellurium new Engine:

- `select(String uid, String target):` Equals to `selectByLabel`.
- `selectByLabel(String uid, String target):` Select based on the text attribute.
- `selectByValue(String uid, String target):` Select based on the value attribute.
- `selectByIndex(String uid, int target):` Select based on index (starting from 1).
- `selectById(String uid, String target):` Select based on the ID attribute.
- `String[] getSelectOptions(String uid):` Get all option texts.
- `String[] getSelectValues(String uid):` Get all option values.
- `String[] getSelectedLabels(String uid):` Get selected texts.
- `String getSelectedLabel(String uid):` Get selected text, for multiple selections, return the first one.
- `String[] getSelectedValues(String uid):` Get selected values.
- `String getSelectedValue(String uid):` Get selected value, for multiple selections, return the first one.
- `addSelectionByLabel(String uid, String target):` Add selection based on text.
- `addSelectionByValue(String uid, String target):` Add selection based on value.

- `removeSelectionByLabel(String uid, String target)`: Remove selection based on text.
- `removeSelectionByValue(String uid, String target)`: Remove selection based on value.
- `removeAllSelections(String uid)`: Remove all selections.

Be aware, the above target variable can be a partial matching as follows.

- `^text`: starts with text.
- `$text`: ends with text.
- `*text`: contains text.
- `!text`: does not contain text.

We can use the following example to demonstrate the use of the new Selector APIs.

HTML snippet:

```
<form method="POST" action="check_phone">
  <select name="Profile/Customer/Telephone/@CountryAccessCode" style="font-size:92%">
    <option value="1" selected=selected>US</option>
    <option value="2" id="uk">UK</option>
    <option value="3">AT</option>
    <option value="4">BE</option>
    <option value="4" id="ca">CA</option>
    <option value="6">CN</option>
    <option value="7">ES</option>
    <option value="8">VG</option>
  </select>
  <input type="text" class="medium paxFerryNameInput" value=""
    name="Profile/Customer/Telephone/@PhoneNumber"
    maxlength="16" id="phone1" tabindex="26">
  <input name="submit" type="submit" value="Check">
</form>
```

UI Module definition:

```
ui.Form(uid: "Form", clocator: [method: "POST", action: "check_phone"]){
  Selector(uid: "Country", clocator: [name: "\$CountryAccessCode"])
  InputBox(uid: "Number", clocator: [name: "\$PhoneNumber"])
  SubmitButton(uid: "check", clocator: [value: "Check"])
}
```

Test Cases:

```
@Test
public void testSelect(){
  String[] countries = sm.getSelectOptions("Form.Country");
  for(String country: countries){
    System.out.println("Country: " + country);
  }
  String[] values = sm.getSelectValues("Form.Country");
  for(String value: values){
    System.out.println("Value: " + value);
  }
  sm.selectByLabel("Form.Country", "US");
  String selected = sm.getSelectedLabel("Form.Country");
  assertEquals("US", selected);
}
```

```
sm.selectByLabel("Form.Country", "$E");
selected = sm.getSelectedLabel("Form.Country");
assertEquals("BE", selected);
sm.selectByLabel("Form.Country", "^E");
selected = sm.getSelectedLabel("Form.Country");
assertEquals("ES", selected);
sm.selectByValue("Form.Country", "8");
selected = sm.getSelectedLabel("Form.Country");
assertEquals("VG", selected);
sm.selectByIndex("Form.Country", 6);
selected = sm.getSelectedLabel("Form.Country");
assertEquals("CN", selected);
sm.selectById("Form.Country", "uk");
selected = sm.getSelectedLabel("Form.Country");
assertEquals("UK", selected);
}
```

Container

Container is an abstract object that can hold a collection of Ui objects. As a result, the Container has a special attribute "useGroupInfo" and its default value is false. If this attribute is true, the Group Locating is enabled. But make sure all the Ui objects inside the Container are children nodes of the Container in the DOM, otherwise, you should not use the Group Locating capability.

Example:

```
ui.Container(uid: "google_start_page", clocator: [tag: "td"], group: "true"){
  InputBox(uid: "searchbox", clocator: [title: "Google Search"])
  SubmitButton(uid: "googlesearch", clocator: [name: "btnG", value: "Google Search"])
  SubmitButton(uid: "Imfeelinglucky", clocator: [value: "I'm Feeling Lucky"])
}
```

Form

Form is a type of Container with its tag being "form" and it represents web form. Like Container, it has the capability to use Group Locating and it has a special method:

- submit()

This method is useful and can be used to submit input data if the form does not have a submit button.

Example:

```
ui.Form(uid: "downloadSearch", clocator: [action: "list", method: "get"],
  group: "true") {
  Selector(uid: "downloadType", clocator: [name: "can", id: "can"])
  TextBox(uid: "searchLabel", clocator: [tag: "span"])

  InputBox(uid: "searchBox", clocator: [name: "q"])
  SubmitButton(uid: "searchButton", clocator: [value: "Search"])
}
```

Table

Table is one of the most complicated Ui Object and also the most often used one. Obviously, its tag is "table" and a table can have headers besides rows and columns. Table is a good choice for a data grid. Tellurium can handle its header, rows, and columns automatically. Table has one important feature: a different UiID than other Ui objects.

For example, if the id of the table is "table1", then its i-th row and j-th column is referred as "table1[i][j]" and its m-th header is "table1.header[m]". Another distinguished feature of Table is that a user can define Ui templates for its elements.

For example, the following example defines different table headers and the template for the first column, the element on the second row and the second column, and the template for all the other elements in other rows and columns.

```
ui.Table(uid: "downloadResult", clocator: [id: "resultstable", class: "results"],
        group: "true")
{
    //define table header
    //for the border column
    TextBox(uid: "{header: 1}", clocator: [:])
    UriLink(uid: "{header: 2} as Filename", clocator: [text: "**Filename"])
    UriLink(uid: "{header: 3} as Summary", clocator: [text: "**Summary + Labels"])
    UriLink(uid: "{header: 4} as Uploaded", clocator: [text: "**Uploaded"])
    UriLink(uid: "{header: 5} as Size", clocator: [text: "**Size"])
    UriLink(uid: "{header: 6} as DownloadCount", clocator: [text: "**DownloadCount"])
    UriLink(uid: "{header: 7} as Extra", clocator: [text: "**..."])

    //define Ui object for the second row and the second column
    InputBox(uid: "{row: 2, column: 2}" clocator: [:])
    //define table elements
    //for the border column
    TextBox(uid: "{row: all, column: 1}", clocator: [:])
    //For the rest, just UriLink
    UriLink(uid: "{row: all, column: all}", clocator: [:])
}
```

Be aware, the templates inside the Table follow the name convention:

- For the i-th row, j-th column, the uid should be "{row: i, column: j} as whatever_id"
- The wild case for row or column is "all"

As a result, "row : all, column : 3" refers to the 3rd column for all rows. Once the templates are defined for the table, Tellurium uses a special way to find a matching for a Ui element "table[i][j]" in the table.

For more details on how Tellurium maps the runtime uid to UI templates, please see Tellurium UID Description Language [<http://code.google.com/p/aost/wiki/TelluriumUIDDescriptionLanguage>].

Generally speaking, Tellurium always searches for the special case first, then broadening the search for more general cases, until all matching cases are found. In this way, user can define very flexible templates for tables.

Table is a type of Container and thus, it can use the Group Locating feature. Table has the following special methods:

- boolean hasHeader()
- int getTableHeaderColumnNum()
- int getTableMaxRowNum()
- int getTableMaxColumnNum()

Standard Table

A StandardTable is a table in the following format

```
table
thead
```

```
tr
  td
  ...
  td
tbody
  tr
    td
    ...
    td
  ...
tbody (multiple tbodies)
  tr
    td
    ...
    td
  ...
tfoot
  tr
    td
    ...
    td
```

To overwrite the default layout, Tellurium core provides the following attributes in the `StandardTable` object.

- `ht`: header tag.
- `hrt`: header row tag, the default tag is "tr".
- `hct`: header column tag, the default tag is "th".
- `bt`: body tag.
- `brt`: body row tag, the default tag is "tr".
- `bct`: body column tag, the default tag is "td".
- `ft`: footer tag.
- `frt`: footer row tag, the default tag is "tr".
- `fmt`: footer column tag, the default tag is "td".

To overwrite the above tags, simply define them in the object definition. For example,

```
ui.StandardTable(uid: "table3", clocator: [:], hct: "td"){
  TextBox(uid: "header: all", clocator: [:])
  TextBox(uid: "footer: all", clocator: [:])
}
```

The header column tag is overwritten to be "td" instead of the default tag "th". To be more accurate, the footer definition has been changed from "foot" to "footer".

For a `StandardTable`, you can specify UI templates for different tbodies. Apart from the methods in `Table`, it has the following additional methods:

- `int getTableFootColumnNum()`
- `int getTableMaxTbodyNum()`
- `int getTableMaxRowNumForTbody(int tbody_index)`
- `int getTableMaxColumnNumForTbody(int body_index)`

Example:

```
ui.StandardTable(uid: "table", clocator: [id: "std"]) {
  UrlLink(uid: "{header: 2} as Filename", clocator: [text: "*Filename"])
  UrlLink(uid: "{header: 3} as Uploaded", clocator: [text: "*Uploaded"])
  UrlLink(uid: "{header: 4} as Size", clocator: [text: "*Size"])
  TextBox(uid: "{header: all}", clocator: [:])

  Selector(uid: "{tbody: 1, row:1, column: 3} as Select", clocator: [name: "can"])
  SubmitButton(uid: "{tbody: 1, row:1, column:4} as Search", clocator: [value:
    "Search", name: "btn"])
  InputBox(uid: "{tbody: 1, row:2, column:3} as Words", clocator: [name: "words"])
  InputBox(uid: "{tbody: 2, row:2, column:3} as Without", clocator: [name: "without"])
  InputBox(uid: "{tbody: 2, row:*, column:1} as Labels", clocator: [name: "labels"])

  TextBox(uid: "{foot: all}", clocator: [tag: "td"])
}
```

List

List is also a Container type abstract Ui object and it can be used to represent any list like Ui objects. Very much like Table, users can define Ui templates for List and following rule of "the special case first and then the general case". The index number is used to specify an element and "all" is used to match all elements in the List. List also uses TextBox as the default Ui if no template could be found. Since List is a Container type, it can use the Group Locating feature.

List has one special attribute "separator", which is used to indicate the tag used to separate different List UI elements. If "separator" is not specified or "", all UI elements must be under the same DOM node, i.e., they are siblings.

Example:

```
ui.List(uid: "subcategory", locator: "", separator: "p"){
  InputBox(uid: "{2} as Search", clocator: [title: "Google Search"])
  UrlLink(uid: "{all}", clocator: [:])
}
```

The List includes the following additional methods:

- `int getListSize(id)`: Gets the item count of a list

Frame

Frame is a type of Container and is used to mode Frame or IFrame. It includes the following attributes:

- `id`
- `name`
- `title`

and the following methods

- `selectParentFrame()`
- `selectTopFrame()`
- `selectFrame(locator)`

- `getWhetherThisFrameMatchFrameExpression(currentFrameString, target)`
- `waitForFrameToLoad(frameAddress, timeout)`

When you test website with IFrames, you should use multiple window mode, i.e., set the option `useMultiWindows` to be true in `TelluriumConfig.groovy`. Be aware that Selenium uses the name attribute to locate a Frame.

Example:

```
ui.Frame(uid: "SubscribeFrame", name: "subscribe"){
    Form(uid: "LoginForm", clocator: [name: "loginForm"]){
        InputBox(uid: "UserName", clocator: [id: "username", type: "text"])
        InputBox(uid: "Password", clocator: [id: "password", type: "password"])
        Button(uid: "Login", clocator: [type: "image", class: "login"])
        CheckBox(uid: "RememberMe", clocator: [id: "rememberme"])
    }
}
```

Window

Window is a type of Container and is used to mode Popup Window. It includes the following attributes:

- `id`
- `name`
- `title`

and the following methods

- `openWindow(String UID, String url)`
- `selectWindow(String UID)`
- `closeWindow(String UID)`
- `boolean getWhetherThisWindowMatchWindowExpression(String currentWindowString, String target)`
- `waitForPopup(String UID, int timeout)`

Example:

```
ui.Window(uid: "HelpWindow", name: "HelpWindow"){
    ...
}

openWindow helpUrl, "HelpWindow"
waitForPopUp "HelpWindow", 2000
selectWindow "HelpWindow"
...
selectMainWindow()
```

Option

Option is also designed to be adaptive the dynamic web. Option is a pure abstract object and it holds multiple UIs with each representing a possible UI pattern at runtime. For example, the List/Grid selector on the issue page can described as:

```
//The selector to choose the data grid layout as List or Grid
ui.Option(uid: "layoutSelector"){
    Container(uid: "layoutSelector", clocator: [tag: "div"], group: "true") {
        TextBox(uid: "List", clocator: [tag: "b", text: "List", direct: "true"])
        UrlLink(uid: "Grid", clocator: [text: "Grid", direct: "true"])
    }
    Container(uid: "layoutSelector", clocator: [tag: "div"], group: "true") {
        UrlLink(uid: "List", clocator: [text: "List", direct: "true"])
        TextBox(uid: "Grid", clocator: [tag: "b", text: "Grid", direct: "true"])
    }
}
```

Note, the option's uid must be the same as the next UI objects it represent and in this way, you do not need to include option's uid in the UiID. For example, you can just use

```
click "layoutSelector.List"
```

instead of

```
click "layoutSelector.layoutSelector.List"
```

The option object will automatically detect which UI pattern you need to use at runtime.

All Purpose Object

Tellurium 0.7.0 added an all purpose object for internal usage only. The object is defined as

```
class AllPurposeObject extends UiObject {
}
```

This object includes all methods for non-Container type objects.

Chapter 5. Tellurium Core Basics

UiID Attribute

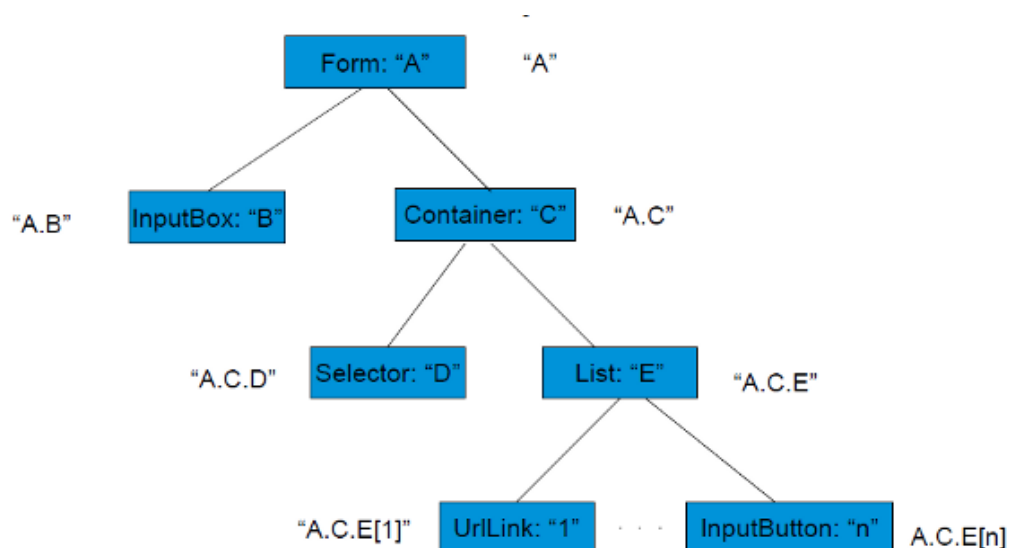
In Tellurium, the UI object is referred to by its UiID, i.e., the UI object identifier. For nested UI objects, the UiID of the UI Object is a concatenated UI objects' uids along its path to the UI Object. For example, in the following nested UI Module shown below, the TextBox is referred to as the "parent_ui.child_ui.grand_child.textbox1".

```
ui.Container(uid: "parent_ui"){
  InputBox(uid: "inputbox1", locator: "...")
  Button(uid: "button1", locator: "...")
  Container(uid: "child_ui"){
    Selector(uid: "selector1", locator: "...")
    ...
    Container(uid: "grand_child"){
      TextBox(uid: "textbox1", locator: "...")
      ...
    }
  }
}
```

The exceptions are tables and lists, which use [x][y] or [x] to reference the elements inside. For example, labels_table[2][1] and GoogleBooksList.subcategory[2]. The Table header can be referred in the format of issueResult.header[2].

More general cases are shown below - General Cases Search Example:

```
ui.Form(uid: "A", clocator: [:]){
  InputBox(uid: "B", clocator: [:])
  Container(uid: "C", clocator: [tag: "div"]){
    Selector(uid: "D", clocator: [:])
    List(uid: "E", clocator: [tag: "ul", separator: "li"]){
      UrlLink(uid: "{1} as Link", clocator: [:])
      InputBox(uid: "{all}", clocator: [:])
    }
  }
}
```



For example, the UiID of the List E in the above diagram is A.C.E and the InputButton in the List E is referred by its index n. For example: A.C.En.

Locator Attributes

Tellurium supports two types of UI Object locators:

1. Base locator
2. Composite locator

The Base locator is a relative XPath.

The Composite locator, denoted by "clocator", specifies a set of attributes for the UI object. The actual locator is derived automatically by Tellurium at runtime. The Composite locator is defined as follows:

```
class CompositeLocator {
    String header
    String tag
    String text
    String trailer
    def position
    boolean direct
    Map<String, String> attributes = [:]
}
```

To use the Composite locator, use "clocator" with a map as its value. For example:

```
clocator: [key1: value1, key2: value2, ...]
```

The default attributes include "header", "tag", "text", "trailer", "position", and "direct". They are all optional. The "direct" attribute specifies whether this UI object is a direct child of its parent UI, and the default value is "false".

If there are additional attributes, they are defined in the same way as the default attributes. For example:

```
clocator: [tag: "div", value: "Tellurium home"]
```

Most Tellurium objects come with default values for certain attributes. For example, the tag attribute.

If these attributes are not specified, the default attribute values are used. In other words, if the default attribute values of a Tellurium UI object are known, omit them in clocator.

For example, if the RadioButton Object's default tag is "input", and the default type is "radio", omit them and write the clocator as follows:

```
clocator: [:]
```

which is equivalent to:

```
clocator: [tag: "input", type: "radio"]
```

Group Attribute

In the Tellurium UI module, the "group" attribute is seen often. For example:

```
ui.Container(uid: "google_start_page", clocator: [tag: "td", group: "true"]){
  InputBox(uid: "searchbox", clocator: [title: "Google Search"])
  SubmitButton(uid: "googlesearch", clocator: [name: "btnG", value: "Google Search"])
  SubmitButton(uid: "Imfeelinglucky", clocator: [value: "I'm Feeling Lucky"])
}
```

The group attribute is a flag for the Group Locating Concept. Usually, the XPath generated by Selenium IDE, XPather, or other tools is a single path to the target node such as:

```
//div/table[@id='something']/div[2]/div[3]/div[1]/div[6]
```

Sibling node information is not used in this example as the XPath depends too much on information from nodes far away from the target node. In Tellurium, every effort is made to localize the information and reduce this dependency by using sibling information or local information.

For example, in the above google UI module example, the group locator concept searches for the location of the "td" tag with its children as "InputBox", "googlesearch" button, and "Imfeelinglucky" button. In this way, the dependencies of the UI elements inside a UI module on external UI elements are reduced, making the UI definition more robust.

self attribute

Some times, elements inside a Table usually are inside its parent tag, for instance, we have the following HTML source.

```
<div id="table">
<div>
  <div id="name">
    <div>Data</div>
    <div>
      <img/>
    </div>
  </div>
  <div id="shortname">
    <div>Bezeichnung</div>
    <div>
      <img/>
    </div>
  </div>
  <div id="type">
    <div>Typ</div>
    <div>
      <img/>
    </div>
  </div>
</div>
<div id="client-area">
  <div>
    <div>Bildsystem</div>
    <div>Bildsystem</div>
    <div>Bildserver</div>
  </div>
  <div>
    <div>Partner</div>
    <div>Partner</div>
    <div>Bestandssystem</div>
  </div>
  <div>
    <div>MS</div>
    <div>MS</div>
    <div>MS</div>
  </div>
</div>
</div>
```


where the "Data" element and many others are inside its parent tag "div". To module this, we added a self attribute to the UiObject class and the default is "false".

To describe the above html, we can define the following UI module.

```
ui.StandardTable(uid: "Table", clocator: [id: "table"], bt: "div", brt: "div", bct: "div"){
    TextBox(uid: "{tbody: 1, row: all, column: 1}", clocator: [tag: "div"], self: "true")
    Image(uid: "{tbody: 1, row: all, column: 2}", clocator: [:])
    TextBox(uid: "{tbody: 2, row: all, column: 1}", clocator: [tag: "div"], self: "true")
    TextBox(uid: "{tbody: 2, row: all, column: 2}", clocator: [tag: "div"], self: "true")
    TextBox(uid: "{tbody: 2, row: all, column: 3}", clocator: [tag: "div"], self: "true")
}
```

To test the UI module, you can simply call the following api.

```
getHTMLSource("Table");
```

Be aware that the self can be "true" **ONLY** for UI elements inside a List, a Table, or a StandardTable Object.

Respond Attribute

Tellurium provides a "respond" attribute used to define any event requiring the UI object to respond.

Most web applications include Javascript, and thus the web testing framework must be able to handle Javascript events. What is important is firing the appropriate events to trigger the event handlers.

Selenium has already provided methods to generate events such as:

```
fireEvent(locator, "blur")
fireEvent(locator, "focus")
mouseOut(locator)
mouseOver(locator)
```

Tellurium was born with Javascript events in mind since it was initially designed to test applications written using the DOJO JavaScript framework.

For example, we have the following radio button:

```
<input type='radio' name='mas_address_key' value='5779' onClick='SetAddress_5779()'>
```

Alternately one can define the radio button as follows:

```
RadioButton(uid: "billing", clocator: [name: 'mas_address_key', value: '5779'])
```

The above code does not respond to the Click event as the Tellurium RadioButton only supports the "check" and "uncheck" actions. This is enough for the normal case. As a result, no "click" event/action is generated during testing.

To address this problem, Tellurium added the "respond" attribute to Tellurium UI objects. The "respond" attribute is used to define any event requiring the UI object to respond. The Radio Button is redefined as:

```
ui.Container(uid: "form", clocator: [whatever]){
  RadioButton(uid: "billing", clocator: [name: 'mas_address_key', value: '5779'],
    respond: ["click"])
}
```

Then issue the following command:

```
click "form.billing"
```

Even if the RadioButton does not have the click method defined by default, it is still able to dynamically add the click method at runtime and call it. Be aware, you have to explicitly call the click method and Event Handler will not automatically trigger the click event.

A more general example is:

```
InputBox(uid: "searchbox", clocator: [title: "Google Search"],
  respond: ["click", "focus", "mouseover", "mouseout", "blur"])
```

Except for the "click" event, all of the "focus", "mouseover", "mouseout", and "blur" events are automatically fired by Tellurium during testing. Do not worry about the event order for the respond attribute as Tellurium automatically re-orders the events and then processes them appropriately. That is to say, the Event Handler will automatically trigger these events.

CSS Selector

Use the CSS Selector to improve the test speed in IE as IE lacks native XPath support and the XPath is slow. Tellurium exploits CSS selector capability to improve test speed dramatically.

Tellurium supports both XPath and CSS selector and still uses CSS selector as the default locator.

New DSL Methods

CSS Selector provides the following additional Selenium methods, utilized in DslContext to form a set of new DSL methods:

String getAllText(String locator):

1. Get all the text from the set of elements corresponding to the CSS Selector. *String getCSS(String locator, String cssName):*
2. Get the CSS properties for the set of elements corresponding to the CSS Selector. *Number getjQuerySelectorCount(String locator):*
3. Get the number of elements matching the corresponding CSS Selector.

Additional CSS Attribute Selectors

jQuery also supports the following attribute selectors:

attribute: have the specified attribute. attribute=value: have the specified attribute with a certain value. attribute!=value: either do not have the specified attribute or have the specified attribute but not with a certain value. attribute^=value: have the specified attribute and it starts with a certain value. attribute\$=value: have the specified attribute and it ends with a certain value. attribute*=value: have the specified attribute and it contains a certain value.

Locator Agnostic Methods

Apart from the above, Tellurium provides a set of locator agnostic methods.

For example, the method automatically decides to use XPath or jQuery dependent on the `exploreJQuerySelector` flag, which can be turned on and off by the following two methods:

1. `public void useCssSelector()`
2. `public void disableCssSelector()`

Tellurium also provides the corresponding XPath specific and CSS selector specific methods for your convenience. However, it is recommended that you use the locator agnostic methods until there is a good reason not to.

The new XPath and CSS Selector specific methods are as follows:

```
String getLocator(String uid)
```

```
String getSelector(String uid)
```

```
String getXPath(String uid)
```

- Locator agnostic:
- CSS selector specific:
- XPath specific:

```
Number getLocatorCount(String locator)
```

```
Number getJQuerySelectorCount(String jquerySelector)
```

- Locator agnostic:
- CSS selector specific:

```
int getTableHeaderColumnNum(String uid)
```

```
int getTableFootColumnNum(String uid)
```

```
int getTableMaxRowNum(String uid)
int getTableMaxColumnNum(String uid)
int getTableMaxRowNumForTbody(String uid, int ntbody)
int getTableMaxColumnNumForTbody(String uid, int ntbody)
int getTableMaxTbodyNum(String uid)
```

```
int getTableHeaderColumnNumBySelector(String uid)
int getTableFootColumnNumBySelector(String uid)
int getTableMaxRowNumBySelector(String uid)
int getTableMaxColumnNumBySelector(String uid)
int getTableMaxRowNumForTbodyBySelector(String uid, int ntbody)
int getTableMaxColumnNumForTbodyBySelector(String uid, int ntbody)
int getTableMaxTbodyNumBySelector(String uid)
```

```
int getTableHeaderColumnNumByXPath(String uid)
int getTableFootColumnNumByXPath(String uid)
int getTableMaxRowNumByXPath(String uid)
int getTableMaxColumnNumByXPath(String uid)
int getTableMaxRowNumForTbodyByXPath(String uid, int ntbody)
int getTableMaxColumnNumForTbodyByXPath(String uid, int ntbody)
int getTableMaxTbodyNumByXPath(String uid)
```

- Locator agnostic:
- CSS selector specific:
- XPath specific:

```
boolean isDisabled(String uid)
```

```
boolean isDisabledBySelector(String uid)
```

```
boolean isDisabledByXPath(String uid)
```

- Locator agnostic:
- CSS selector specific:

- XPath specific:

```
def getAttribute(String uid, String attribute)
```

- Locator agnostic:

```
def hasCssClass(String uid, String cssClass)
```

- Locator agnostic

```
String[] getCSS(String uid, String cssName)
```

- CSS selector specific:

```
String[] getAllTableCellText(String uid)  
String[] getAllTableCellTextForTbody(String uid, int index)
```

- CSS selector specific:

```
int getListSize(String uid)
```

```
getListSizeBySelector(String uid)
```

```
getListSizeByXPath(String uid)
```

- Locator agnostic:

- CSS selector specific:

- XPath specific:

1. Get the Generated locator from the UI module
2. Get the Number of Elements matching the Locator
3. Table Methods
4. Verify if an Element is Disabled
5. Get the Attribute
6. Check the CSS Class
7. Get CSS Properties

8. Get All Data from a Table

9. Get List Size

There are issues to be aware of with CSS Selector:

- If you have a duplicate "id" attribute on the page, CSS selector always returns the first DOM reference, ignoring other DOM references with the same "id" attribute.
- Some attributes may not be working in jQuery. For example, the "action" attribute in a form. Tellurium has a black list to automatically filter out the attributes that are not honored by CSS selector.
- The "src" attribute in Image has to be a full URL such as <http://www.google.com>. One workaround is to put '*' before the URL.

UI Templates

Tellurium UI templates have two purposes:

When there are many identical UI elements, use one template to represent them all. When there are variable/dynamic sizes of UI elements at runtime, the patterns are known, but not the size.

More specifically, Table and List are two Tellurium objects that define UI templates.

Table defines two dimensional UI templates. List defines one dimensional UI templates.

The Template has special UUIDs such as "2", "all", or "row: 1, column: 2".

Looking at use case (1), the HTML source is:

```
<ul class="a">
  <li>
    <A HREF="site?tcid=a" class="b">
      AA
    </A>
  </li>
  <li>
    <A HREF="site?tcid=b" class="b">
      BB
    </A>
  </li>
  <li>
    <A HREF="site?;tcid=c" class="b">
      CC
    </A>
  </li>
  <li>
    <A HREF="site?tcid=d" class="b">
      DD
    </A>
  </li>
  <li>
    <A HREF="site?tcid=e" class="b">
      EE
    </A>
  </li>
  <li>
    <A HREF="site?tcid=f" class="b">
      FF
    </A>
  </li>
</ul>
```

In this example there are six links. Without templates, one would put six `UrlLink` objects in the UI module. By using the templates, the work is easier and simplified.

```
ui.List(uid: "list", clocator: [tag: "ul", class: "a"], separator:"li")
{
    UrlLink(uid: "{all}", clocator: [class: "b"])
}
```

For use case (2), a common application is the data grid. Look at the "issueResult" data grid on the Tellurium Issues page for an easy and simplified result as shown below:

```
ui.Table(uid: "issueResult", clocator: [id: "resultstable", class: "results"],
        group: "true")
{
    TextBox(uid: "{header: 1}", clocator: [:])
    UrlLink(uid: "{header: 2} as ID", clocator: [text: "**ID"])
    UrlLink(uid: "{header: 3} as Type", clocator: [text: "**Type"])
    UrlLink(uid: "{header: 4} as Status", clocator: [text: "**Status"])
    UrlLink(uid: "{header: 5} as Priority", clocator: [text: "**Priority"])
    UrlLink(uid: "{header: 6} as Milestone", clocator: [text: "**Milestone"])
    UrlLink(uid: "{header: 7} as Owner", clocator: [text: "**Owner"])
    UrlLink(uid: "{header: 9} as Summary", clocator: [text: "**Summary + Labels"])
    UrlLink(uid: "{header: 10} as Extra", clocator: [text: "**..."])

    //define table elements
    //for the border column
    TextBox(uid: "{row: all, column: 1}", clocator: [:])
    //For the rest, just UrlLink
    UrlLink(uid: "{row: all, column: all}", clocator: [:])
}
```

The resulting definitions shown are very simple, time-saving and easy to use.

If the user has multiple templates such as the "issueResult" shown in the table above, the rule generally applied to the templates is: "specific one first, general one later".

"Include" Frequently Used Sets of Elements in UI Modules

When there is a frequently used set of elements, re-defining them repeatedly in your UI module is not necessary. Simply use the Tellurium "Include" syntax to re-use the pre-defined UI elements.

```
Include(uid: UID, ref: REFERRED_UID)
```

Use "ref" to reference the object to be included, then specify the UID for the object. Note: If a different UID is required, there is no need to specify it.

If the Object UID is not the same as the original one, Tellurium clones a new object for users so that multiple objects with different UIDs are available.

For example, first define the following reused UI module:

```
ui.Container(uid: "SearchModule", clocator: [tag: "td"], group: "true") {
    InputBox(uid: "Input", clocator: [title: "Google Search"])
    SubmitButton(uid: "Search", clocator: [name: "btnG", value: "Google Search"])
    SubmitButton(uid: "ImFeelingLucky", clocator: [value: "I'm Feeling Lucky"])
}
```

Then, include it into the new UI module as follows:

```
ui.Container(uid: "Google", clocator: [tag: "table"]) {
    Include(ref: "SearchModule")
    Include(uid: "MySearchModule", ref: "SearchModule")
    Container(uid: "Options", clocator: [tag: "td", position: "3"], group: "true") {
        UrlLink(uid: "LanguageTools", clocator: [tag: "a", text: "Language Tools"])
        UrlLink(uid: "SearchPreferences", clocator: [tag: "a", text: "Search Preferences"])
        UrlLink(uid: "AdvancedSearch", clocator: [tag: "a", text: "Advanced Search"])
    }
}
```

Logical Container

The Container object in Tellurium is used to hold child objects that are in the same subtree in the DOM object. However, there are always exceptions. For example, the Logical Container (or Virtual Container - Logical Container is preferred) can violate this rule.

What is a Logic Container It is a Container with an empty locator. For instance:

```
Container(uid: "logical"){
    .....
}
```

But empty != nothing. There are some scenarios where the Logical Container can play an important role. The Container includes an uid for a reference and it logically groups the UI element together.

For example, in the following example the UI elements under the Tag li are different:

```
<div class="block_content">
    <ul>
        <li>
            <h5>
                <a href="" title="">xxx</a>
            </h5>
            <p class="product_desc">
                <a href=".." title="More">...</a>
            </p>
            <a href="..." title=".." class="product_image">
                
            </a>
            <p>
                <a class="button" href=".." title="View">View</a>
                <a title="Add to cart">Add to cart</a>
            </p>
        </li>
        <li>
            similar UI
        </li>
        <li>
            similar UI
        </li>
    </ul>
</div>
```

The issue is how to write the UI template for them. The logical Container then comes into play. For example, the UI module is written as follows:


```

ui.Container(uid: "content", clocator: [tag: "div", class: "block_content"]){
    List(uid: "list", clocator: [tag: "ul", separator:"li"]){
        Container("{all}") {
            UrlLink(uid: "title", clocator: [title: "View"])
            .....
            other elements inside the li tag
        }
    }
}

```

Another usage of the logical Container is to convert the test case recorded by Selenium IDE to Tellurium test cases. For example, using the search UI on the Tellurium download page, first record the following Selenium test case using Selenium IDE:

```

import com.thoughtworks.selenium.SeleniumTestCase;

public class SeleniumTestCase extends SeleniumTestCase {
    public void setUp() throws Exception {
        setUp("http://code.google.com/", "*chrome");
    }

    public void testNew() throws Exception {
        selenium.open("/p/aost/downloads/list");
        selenium.select("can", "label=regexp:\\sAll Downloads");
        selenium.type("q", "TrUMP");
        selenium.click("//input[@value='Search']");
        selenium.waitForPageToLoad("30000");
    }
}

```

Do not be confused by the locator "can" and "q", as they are UI element IDs and are easily expressed in XPath. The "label=regexp:\\sAll Downloads" part shows that Selenium uses regular express to match the String and the "\\s" stands for a space. As a result, write the UI module based on the above code.

```

public class TelluriumDownloadPage extends DslContext {

    public void defineUi() {
        ui.Container(uid: "downloads") {
            Selector(uid: "downloadType", locator: "//*[@id='can']")
            InputBox(uid: "input", locator: "//*[@id='q']")
            SubmitButton(uid: "search", locator: "//input[@value='Search']")
        }
    }

    public void searchDownload(String downloadType, String searchKeyWords) {
        selectByLabel "downloads.downloadType", downloadType
        keyType "downloads.input", searchKeyWords
        click "downloads.search"
        waitForPageToLoad 30000
    }
}

```

The Tellurium test case is created accordingly:

```

public class TelluriumDownloadPageTestCase extends TelluriumJUnitTestCase {

    protected static TelluriumDownloadPage ngsp;

    @BeforeClass
    public static void initUi() {
        ngsp = new TelluriumDownloadPage();
        ngsp.defineUi();
    }
}

```

```
    }

    @Test
    public void testSearchDownload(){
        connectUrl("http://code.google.com/p/aost/downloads/list");
        ngsp.searchDownload(" All Downloads", "TrUMP");
    }
}
```

toggle

Tellurium 0.7.0 provides a toggle method to animate the UI element on the web page. For example, you can the following commands to show the UI element under testing.

```
toggle "Form.Username.Input"
pause 500
toggle "Form.Username.Input"
```

getHTMLSource

Use getHTMLSource, users can get back the runtime html source for a UI module. Tellurium provided two methods for this purpose.

```
public Map getHTMLSourceResponse(String uid);
public void getHTMLSource(String uid);
```

The first method get back the html source as a uid-to-html map and the second one simply print out the html source on the console. Be aware, getHTMLSource is only available in Tellurium new APIs.

Example:

```
@Test
public void testGetHTMLSource(){
    useEngineLog(true);
    useTelluriumApi(true);
    useCache(true);
    connect("JettyLogon");
    jlm.getHTMLSource("Form");
}
```

and the output is as follows:

Form:

```
<form method="POST" action="j_security_check">
  <table border="0" cellpadding="1" cellspacing="2">
    <tbody><tr>
      <td>Username:</td>
      <td><input size="12" value="" name="j_username" maxlength="25" type="text"></td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input size="12" value="" name="j_password" maxlength="25" type="password"></td>
    </tr>
  </tbody>
</table>
```

```
        <td colspan="2" align="center">
            <input name="submit" value="Login" type="submit">
        </td>
    </tr>
</tbody></table>
</form>

Form.Username:

    <tr>
        <td>Username:</td>
        <td><input size="12" value="" name="j_username" maxlength="25" type="text"></td>
    </tr>

Form.Username.Label:

    <td>Username:</td>

Form.Username.Input:

    <input size="12" value="" name="j_username" maxlength="25" type="text">

Form.Password:

    <tr>
        <td>Password:</td>
        <td><input size="12" value="" name="j_password" maxlength="25" type="password"></td>
    </tr>

Form.Password.Label:

    <td>Password:</td>

Form.Password.Input:

    <input size="12" value="" name="j_password" maxlength="25" type="password">

Form.Submit:

    <input name="submit" value="Login" type="submit">
```

type and keyType

type and keyType now accept different types of objects and convert them to a String automatically by calling the toString() method. For example, you can use the following commands:

```
type "Google.Input", "Tellurium"
type "Google.Input", 12.15
type "Google.Input", true
```

Customize Individual Test Settings Using setCustomConfig

TelluriumConfig.groovy provides project level test settings. Use setCustomConfig to customize individual test settings.

```
public void setCustomConfig(boolean runInternally, int port, String browser,
                           boolean useMultiWindows, String profileLocation)

public void setCustomConfig(boolean runInternally, int port, String browser,
                           boolean useMultiWindows, String profileLocation, String serverHost)
```

For example, specify custom settings as follows:

```
public class GoogleStartPageJavaTestCase extends TelluriumJavaTestCase {
    static{
        setCustomConfig(true, 5555, "*chrome", true, null);
    }

    protected static NewGoogleStartPage ngsp;

    @BeforeClass
    public static void initUi() {
        ngsp = new NewGoogleStartPage();
        ngsp.defineUi();
        ngsp.useJavascriptXPathLibrary();
    }

    @Test
    public void testGoogleSearch() {
        connectUrl("http://www.google.com");
        ngsp.doGoogleSearch("tellurium selenium Groovy Test");
    }

    .....
}
```

User Custom Selenium Extensions

To support user custom selenium extensions, Tellurium Core adds the following configurations to TelluriumConfig.groovy.

```
embeddedserver {
    .....

    //user-extension.js file, for example "target/user-extensions.js"
    userExtension = ""
}
connector{
    .....

    //user's class to hold custom selenium methods associated with user-extensions.js
    //should in full class name, for instance, "com.mycom.CustomSelenium"
    customClass = ""
}
```

Where the userExtension points to the user's user-extensions.js file. For example, use the following user-extensions.js:

```
Selenium.prototype.doTypeRepeated = function(locator, text) {
    // All locator-strategies are automatically handled by "findElement"
    var element = this.page().findElement(locator);

    // Create the text to type
    var valueToType = text + text;

    // Replace the element text with the new text
    this.page().replaceText(element, valueToType);
};
```

The customClass is the user's class for custom Selenium methods, extending Tellurium org.tellurium.connector.CustomCommand class:

```
public class MyCommand extends CustomCommand{

    public void typeRepeated(String locator, String text){
        String[] arr = {locator, text};
        commandProcessor.doCommand("typeRepeated", arr);
    }
}
```

Tellurium core automatically loads up user-extensions.js and custom commands. The n, user uses:

```
customUiCall(String uid, String method, Object[] args)
```

to call the custom methods. For instance:

```
customUiCall "Google.Input", typeRepeated, "Tellurium Groovy"
```

The customUiCall method handles all the UI to locator mapping for users. Tellurium also provides the following method for users to make direct calls to the Selenium server.

```
customDirectCall(String method, Object[] args)
```

The Dump Method

Tellurium Core added a dump method to print out the UI object's and its descendants' runtime locators that Tellurium Core generates.

```
public void dump(String uid)
```

where the uid is the UI object id to be dumped. An important feature is that the dump() method does not require the user to run the actual tests. That is to say, the user does not need to run the Selenium server and launch the web browser. Simply define the UI module, then call the dump() method.

For example, to define the UI module for Tellurium Issue Search module, complete as follows:

```
public class TelluriumIssueModule extends DslContext {

    public void defineUi() {

        //define UI module of a form include issue type selector and issue search
        ui.Form(uid: "issueSearch", clocator: [action: "list", method: "get"], group: "true"){
            Selector(uid: "issueType", clocator: [name: "can", id: "can"])
            TextBox(uid: "searchLabel", clocator: [tag: "span", text: "**for"])
            InputBox(uid: "searchBox", clocator: [type: "text", name: "q"])
            SubmitButton(uid: "searchButton", clocator: [value: "Search"])
        }
    }
}
```

The user can use the dump method in the following way:

```
TelluriumIssueModule tisp = new TelluriumIssueModule();
tisp.defineUi();
tisp.dump("issueSearch");
```

The above code prints out the runtime XPath locators.

Dump locator information for issueSearch is as follows:

```
-----
issueSearch: //descendant-or-self::form[@action="list" and @method="get"]
issueSearch.issueType: //descendant-or-self::form[descendant::select[@name="can" and
    @id="can"] and descendant::span[contains(text(),"for")] and
    descendant::input[@type="text" and @name="q"] and
    descendant::input[@value="Search" and @type="submit"] and
    @action="list" and @method="get"]/descendant-or-self::select
    [@name="can" and @id="can"]
issueSearch.searchLabel: //descendant-or-self::form[descendant::select[@name="can"
    and @id="can"] and descendant::span[contains(text(),"for")] and
    descendant::input[@type="text" and @name="q"] and descendant::input[
    @value="Search" and @type="submit"] and @action="list" and @method="get"]
    /descendant-or-self::span[contains(text(),"for")]
issueSearch.searchBox: //descendant-or-self::form[descendant::select[@name="can"
    and @id="can"] and descendant::span[contains(text(),"for")] and
    descendant::input[@type="text" and @name="q"] and descendant::input[
    @value="Search" and @type="submit"] and @action="list" and @method="get"]
    /descendant-or-self::input[@type="text" and @name="q"]
issueSearch.searchButton: //descendant-or-self::form[descendant::select[@name="can"
    and @id="can"] and descendant::span[contains(text(),"for")] and
    descendant::input[@type="text" and @name="q"] and descendant::input[
    @value="Search" and @type="submit"] and @action="list" and @method="get"]
    /descendant-or-self::input[@value="Search" and @type="submit"]
-----
```

Engine State Offline Update

For some reasons, you may need to make Engine state calls before you actually connect to the Engine. For examples, call one of the following methods.

```
public void enableCache();
public void disableCache();
public void useTelluriumApi(boolean isUse);
public void useClosestMatch(boolean isUse);
```

We added an Engine State tracer in the bundle tier to record all the requests if the Engine is not connected and aggregate them. Once the Engine is connected, Tellurium will automatically send out the Engine state update request.

For example, the following test code works fine in 0.7.0.

```
@Test
public void testOfflineEngineStateUpdate(){
    JettyLogonModule jlm = new JettyLogonModule();
    jlm.defineUi();
    useCssSelector(true);
}
```

```
useTrace(true);  
//Engine state offline update  
useTelluriumApi(true);  
useCache(true);  
connectSeleniumServer();  
connectUrl("http://localhost:8080/logon.html");  
jlm.logon("test", "test");  
//Back to state online update  
useClosestMatch(true);  
connectUrl("http://localhost:8080/logon.html");  
jlm.plogon("test", "test");  
}
```

Testing Support

In the package `org.tellurium.test`, Tellurium provides three different ways to write Tellurium tests:

1. `TelluriumJUnitTestCase`
2. `TelluriumTestNGTestCase`
3. `TelluriumGroovyTestCase`
4. `TelluriumMockJUnitTestCase`
5. `TelluriumMockTestNGTestCase`

Tellurium JUnit Test Case

Used for JUnit, and supports the following JUnit 4 annotations: JUnit is upgraded to 4.7 since it provides more features. One such a good feature is the Rule annotation.

To be consistent with TestNG test case, the class `TelluriumJavaTestCase` is deprecated and you should use `TelluriumJUnitTestCase` instead.

- `@BeforeClass`
- `@AfterClass`
- `@Before`
- `@After`
- `@Test`
- `@Ignore`

Tellurium TestNG Test Case

Used for TestNG. Similarly, using the following annotations:

- `@BeforeSuite`
- `@AfterSuite`
- `@BeforeTest`
- `@AfterTest`
- `@BeforeGroups`

- `@AfterGroups`
- `@BeforeClass`
- `@AfterClass`
- `@BeforeMethod`
- `@AfterMethod`
- `@DataProvider`
- `@Parameters`
- `@Test`

`TelluriumTestNGTestCase` was changed to allow the setup and teardown procedures only work once for all the tests. The magic are the `@BeforeTest` and `@AfterTest` annotations. See the following code for more details,

```
public abstract class TelluriumTestNGTestCase extends BaseTelluriumJavaTestCase {

    @BeforeTest(alwaysRun = true)
    public static void setUpForTest() {
        tellurium = TelluriumSupport.addSupport();
        tellurium.start(customConfig);
        connector = (SeleniumConnector) tellurium.getConnector();
    }

    @AfterTest(alwaysRun = true)
    public static void tearDownForTest() {
        if(tellurium != null)
            tellurium.stop();
    }
}
```

Tellurium Groovy Test Case

Used for Groovy test cases.

Data Driven Testing: Tellurium provides the class `TelluriumDataDrivenModule` for users to define data driven testing modules.

Class `TelluriumDataDrivenTest` is used to drive the actual tests.

Tellurium also provides users the capability of writing Tellurium tests and Tellurium data driven tests in pure DSL scripts. The `DslScriptExecutor` is used to run the `.dsl` files.

TelluriumMockJUnitTestCase and TelluriumMockTestNGTestCase

`TelluriumMockJUnitTestCase` and `TelluriumMockTestNGTestCase` incorporated the Mock Http Server so that you can load up a html web page and test against it with minimal configuration.

One example is as follows,

```
public class JettyLogonJUnitTestCase extends TelluriumMockJUnitTestCase {
    private static JettyLogonModule jlm;
```



```
@BeforeClass
public static void initUi() {
    registerHtmlBody("JettyLogon");

    jlm = new JettyLogonModule();
    jlm.defineUi();
    useCssSelector(true);
    useTelluriumApi(true);
    useTrace(true);
    useCache(true);
}

@Before
public void connectToLocal() {
    connect("JettyLogon");
}

@Test
public void testJsonfyUiModule(){
    String json = jlm.jsonify("Form");
    System.out.println(json);
}

@AfterClass
public static void tearDown(){
    showTrace();
}
}
```

where the html file "JettyLogon" lives in the class path "org/telluriumsource/html". You can change this by the following method:

```
public static void setHtmlClassPath(String path);
```

The implementation is simple as shown as follows.

```
public class TelluriumMockJUnitTestCase extends TelluriumJUnitTestCase {
    protected static MockHttpServer server;

    @BeforeClass
    public static void init(){
        server = new MockHttpServer(8080);
        server.start();
        connectSeleniumServer();
    }

    public static void registerHtml(String testname){
        server.registerHtml("/" + testname + ".html", server.getHtmlFile(testname));
    }

    public static void registerHtmlBody(String testname){
        server.registerHtmlBody("/" + testname + ".html", server.getHtmlFile(testname));
    }

    public static void setHtmlClassPath(String path){
        server.setHtmlClassPath(path);
    }

    public static void connect(String testname){
        connectUrl("http://localhost:8080/" + testname + ".html");
    }

    @AfterClass
    public static void tearDown(){
        server.stop();
    }
}
```

Tellurium Configuration

The configuration parser has been refactored. The configuration file name is stored in the Environment class as follows.

```
@Singleton
public class Environment implements Configurable{

    protected String configFileNames = "TelluriumConfig.groovy";

    protected String configString = "";

    public static Environment getEnvironment(){
        return Environment.instance;
    }

    public void useConfigString(String json){
        this.configString = json;
    }

    ...
}
```

That is to say, you can get the Environment singleton instance and change the file name before Tellurium core is loaded up if you have a good reason to do that. In the meanwhile, we add support to load the configuration from a JSON string, which will be stored in the configString} variable in the Environment.

If the configString variable is not null or empty, Tellurium core will honor the JSON string and will ignore the configuration file. One way to use the JSON string is illustrated by the following example.

```
public class JettyLogonJUnitTestCase extends TelluriumMockJUnitTestCase {
    private static JettyLogonModule jlm;
    static{
        Environment env = Environment.getEnvironment();
        env.useConfigString(JettyLogonModule.JSON_CONF);
    }

    ...
}
```

where variable JSON_CONF is the JSON configuration string.

```
public class JettyLogonModule extends DslContext {
    public static String JSON_CONF = ""{"tellurium":{"test":{"result":{"reporter":"XMLResultReporter","filer
...
}}
```

If we want a pretty look of the JSON String, we can use JSON Visualization to format it as follows.

```
{
  "tellurium": {
    "test": {
      "result": {
        "reporter": "XMLResultReporter",
        "filename": "TestResult.output",
        "output": "Console"
      },
      "exception": {
        "filenamePattern": "Screenshot.png",

```

```
        "captureScreenshot": false
    },
    "execution": {
        "trace": false
    }
},
"accessor": {
    "checkElement": false
},
"embeddedserver": {
    "port": "4444",
    "browserSessionReuse": false,
    "debugMode": false,
    "ensureCleanSession": false,
    "interactive": false,
    "avoidProxy": false,
    "timeoutInSeconds": 30,
    "runInternally": true,
    "trustAllSSLCertificates": true,
    "useMultiWindows": false,
    "userExtension": "",
    "profile": ""
},
"uiobject": {
    "builder": { }
},
"eventhandler": {
    "checkElement": false,
    "extraEvent": false
},
"i18n": {
    "locale": "en_US"
},
"connector": {
    "baseUrl": "http://localhost:8080",
    "port": "4444",
    "browser": "*chrome",
    "customClass": "",
    "serverHost": "localhost",
    "options": ""
},
"bundle": {
    "maxMacroCmd": 5,
    "useMacroCommand": true
},
"datadriven": {
    "dataproducer": {
        "reader": "PipeFileReader"
    }
},
"widget": {
    "module": {
        "included": ""
    }
}
}
```

If the JSON String is empty or null, Tellurium loads the configuration file `TelluriumConfig.groovy` first from the project root. If not found, it loads it from the class path. As a result, you can put the `TelluriumConfig.groovy` file under the resources directory if you use Maven.

The configuration file `TelluriumConfig.groovy` includes three extra sections. That is to say, the bundle tier,

```
//the bundling tier
bundle{
    maxMacroCmd = 5
    useMacroCommand = true
}
```

the i18n,

```
i18n{
    //locale = "fr_FR"
    locale = "en_US"
}
```

and the trace.

```
test{
    execution{
        //whether to trace the execution timing
        trace = false
    }
}
```

The sample configure file for 0.7.0 is available [here](#).

Tellurium Core 0.7.0 added support for Configuration file check as suggested by our user. That can be easily demonstrated by the following example, if we comment out the following section in TelluriumConfig.groovy

```
//the bundling tier
// bundle{
//     maxMacroCmd = 5
//     useMacroCommand = true
// }
```

What will happen Tellurium will throw the following exception

```
java.lang.ExceptionInInitializerError
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:169)
    at org.telluriumsource.bootstrap.TelluriumSupport.class$(TelluriumSupport.groovy)
    at org.telluriumsource.bootstrap.TelluriumSupport.$get$$class$org$telluriumsource$framework
        $TelluriumFrameworkMetaClass(TelluriumSupport.groovy)
    at org.telluriumsource.bootstrap.TelluriumSupport.addSupport(TelluriumSupport.groovy:17)
    at org.telluriumsource.test.java.TelluriumTestNGTestCase.setUpForTest(TelluriumTestNGTestCase.java:17)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:597)
    at org.testng.internal.MethodHelper.invokeMethod(MethodHelper.java:607)
    at org.testng.internal.Invoker.invokeConfigurationMethod(Invoker.java:417)
    at org.testng.internal.Invoker.invokeConfigurations(Invoker.java:154)
    at org.testng.internal.Invoker.invokeConfigurations(Invoker.java:88)
    at org.testng.TestRunner.beforeRun(TestRunner.java:510)
    at org.testng.TestRunner.run(TestRunner.java:478)
    at org.testng.SuiteRunner.runTest(SuiteRunner.java:332)
    at org.testng.SuiteRunner.runSequentially(SuiteRunner.java:327)
    at org.testng.SuiteRunner.privateRun(SuiteRunner.java:299)
    at org.testng.SuiteRunner.run(SuiteRunner.java:204)
    at org.testng.TestNG.createAndRunSuiteRunners(TestNG.java:867)
    at org.testng.TestNG.runSuitesLocally(TestNG.java:832)
    at org.testng.TestNG.run(TestNG.java:748)
    at org.testng.remote.RemoteTestNG.run(RemoteTestNG.java:73)
    at org.testng.remote.RemoteTestNG.main(RemoteTestNG.java:124)
Caused by: org.telluriumsource.exception.ConfigNotFoundException: Cannot find Tellurium Configuration
    tellurium.bundle.maxMacroCmd, please check http://code.google.com/p/aost/wiki/TelluriumConfig070 for
    updated TelluriumConfig.groovy, or report to Tellurium user group at http://groups.google.com/group/tellurium
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:39)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:107)
```

```
at java.lang.reflect.Constructor.newInstance(Constructor.java:513)
at org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:77)
at org.codehaus.groovy.runtime.callsite.ConstructorSite$ConstructorSiteNoUnwrapNoCoerce.
    callConstructor(ConstructorSite.java:107)
at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:52)
at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:192)
at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:200)
at org.telluriumsource.config.TelluriumConfigurator.checkConfig(TelluriumConfigurator.groovy:41)
```

Run DSL Script

Tellurium 0.7.0 provides a `rundsl.groovy` script for users to run DSL test script. The `rundsl.groovy` uses Groovy Grape to automatically download all dependencies and then run DSL script.

First, you need to configure Grape. Put the following `grapeConfig.xml` file into your `home/.groovy/`.

```
<ivysettings>
  <settings defaultResolver="downloadGrapes"/>
  <property
    name="local-maven2-pattern"
    value="${user.home}/.m2/repository/[organisation]/[module]/[revision]/[module]-[revision](-[classifier])-[revision].[extension]"
    override="false" />
  <resolvers>
    <chain name="downloadGrapes">
      <filesystem name="cachedGrapes">
        <ivy pattern="${user.home}/.groovy/grapes/[organisation]/[module]/ivy-[revision].xml"/>
        <artifact pattern="${user.home}/.groovy/grapes/[organisation]/[module]/[type]/[artifact]-[revision].[extension]"/>
      </filesystem>
      <filesystem name="local-maven-2" m2compatible="true" local="true">
        <ivy pattern="${local-maven2-pattern}"/>
        <artifact pattern="${local-maven2-pattern}"/>
      </filesystem>
      <!-- todo add 'endorsed groovy extensions' resolver here -->
      <ibiblio name="kungfuters.3rdparty" root="http://maven.kungfuters.org/content/repositories/thirdparty"
        m2compatible="true"/>
      <ibiblio name="codehaus" root="http://repository.codehaus.org/" m2compatible="true"/>
      <ibiblio name="ibiblio" m2compatible="true"/>
      <ibiblio name="java.net2" root="http://download.java.net/maven/2/" m2compatible="true"/>
      <ibiblio name="openqa" root="http://archiva.openqa.org/repository/releases/" m2compatible="true"/>
      <ibiblio name="kungfuters.snapshot" root="http://maven.kungfuters.org/content/repositories/snapshots"
        m2compatible="true"/>
      <ibiblio name="kungfuters.release" root="http://maven.kungfuters.org/content/repositories/releases/"
        m2compatible="true"/>
    </chain>
  </resolvers>
</ivysettings>
```

Then run

```
groovy rundsl.groovy -f DSL_script_name
```

If you are behind a firewall

```
groovy -Dhttp.proxyHost=proxy_host -Dhttp.proxyPort=proxy_port rundsl.groovy -f DSL_script_name
```

If you defined custom UI objects in your project, you should first build the jar artifact, and install it to your local Maven repo. Then update the `rundsl.groovy` file to add the new dependency. For example, in the `tellurium-website` reference project, we defined custom UI objects, we added the following two lines into the `rundsl.groovy` script.

```
Grape.grab(group:'org.telluriumsource', module:'tellurium-website', version:'0.7.0-SNAPSHOT',
    classLoader:this.class.classLoader.rootLoader)

...

@Grab(group='org.telluriumsource', module='tellurium-website', version='0.7.0-SNAPSHOT')
```

useTelluriumEngine

To make it convenient for users, Tellurium provides a useTelluriumEngine command as follows,

```
void useTelluriumEngine(boolean isUse){
    useCache(isUse);
    useMacroCmd(isUse);
    useTelluriumApi(isUse);
}
```

As you can see, this command actually consists of three commands. In DslContext, Tellurium also provides two handy DSL style methods.

```
void enableTelluriumEngine();
void disableTelluriumEngine();
```

Trace

Tellurium 0.7.0 provides built-in support for the command execution time including
Execution time for each command Total run time Aggregated times for each command

For example, you can see the output as follows,

```
TE: Name: getCurrentCachePolicy, start: 1260496075484, duration: 28ms
TE: Name: useDiscardNewCachePolicy, start: 1260496075514, duration: 51ms
TE: Name: getCurrentCachePolicy, start: 1260496075566, duration: 74ms
TE: Name: useDiscardOldCachePolicy, start: 1260496075642, duration: 35ms
TE: Name: getCurrentCachePolicy, start: 1260496075678, duration: 42ms
TE: Start Time: 1260496060373
End Time: 1260496075720
Total Runtime: 15347ms
Name: keyPress, count: 24, total: 1277ms, average: 53ms
Name: getCurrentCachePolicy, count: 5, total: 222ms, average: 44ms
Name: useDiscardOldCachePolicy, count: 1, total: 35ms, average: 35ms
Name: useDiscardInvalidCachePolicy, count: 1, total: 33ms, average: 33ms
Name: enableCache, count: 2, total: 151ms, average: 75ms
Name: click, count: 3, total: 194ms, average: 64ms
Name: isElementPresent, count: 2, total: 100ms, average: 50ms
Name: useDiscardLeastUsedCachePolicy, count: 1, total: 39ms, average: 39ms
Name: type, count: 1, total: 81ms, average: 81ms
Name: typeKey, count: 3, total: 124ms, average: 41ms
```

We added the following settings to TelluriumConfig.groovy

```
test{
    execution{
        //whether to trace the execution timing
    }
}
```

```
        trace = false
    }
```

You can use the follow methods in `DslContext` to turn on or off the trace, and get the trace data.

```
public void enableTrace();
public void disableTrace();
public void showTrace();
```

Methods Accessible in Test Cases

There are many Tellurium APIs that used to be available only in `DslContext`. That is to say, you have to extend `DslContext` to use them. For example, you often see code in a test case likes this,

```
GoogleSearchModule gsm = new GoogleSearchModule();
gsm.defineUi();
gsm.usejQuerySelector();
gsm.registerNamespace("te", te_ns);
```

Now, many of them, which are not really tied to a specific UI module, are made available in Tellurium test cases. For example, the above code can be changed as follows,

```
GoogleSearchModule gsm = new GoogleSearchModule();
gsm.defineUi();
useCssSelector();
registerNamespace("te", te_ns);
```

New `TelluriumGroovyTestCase` provides the following list of new APIs for your convenience.

```
public void useCssSelector(boolean isUse);

public void useCache(boolean isUse);

public void cleanCache();

public boolean isUsingCache();

public void setCacheMaxSize(int size);

public int getCacheSize();

public int getCacheMaxSize();

public Map<String, Long> getCacheUsage();

public void useCachePolicy(CachePolicy policy);

public String getCurrentCachePolicy();

public void useDefaultXPathLibrary();

public void useJavascriptXPathLibrary();

public void useAjaxsltXPathLibrary();

public void registerNamespace(String prefix, String namespace);

public String getNamespace(String prefix);
```

```
public void pause(int milliseconds);

public void useMacroCmd(boolean isUse);

public void setMaxMacroCmd(int max);

public int getMaxMacroCmd();

public boolean isUseTelluriumApi();

public void useTelluriumApi(boolean isUse);

public void useTrace(boolean isUse);

public void showTrace();

public void setEnvironment(String name, Object value);

public Object getEnvironment(String name);

public void allowNativeXpath(boolean allow);

public void addScript(String scriptContent, String scriptTagId);

public void removeScript(String scriptTagId);

public void EngineState getEngineState();

public void useEngineLog(boolean isUse);

public void useTelluriumEngine(boolean isUse);

public void dumpEnvironment();
```

Tellurium Java test cases provide the same APIs and the only difference is that the APIs in Tellurium Java test cases are static.

Environment

We added an Environment class to Tellurium Core so that you can change the runtime environment. The Environment class is defined as follows,

```
public class Environment {
    def envVariables = [:];
    public boolean isUseCssSelector();

    public boolean isUseCache();

    public boolean isUseBundle();

    public boolean isUseScreenshot();

    public boolean isUseTrace();

    public boolean isUseTelluriumApi();

    public boolean isUseEngineLog();

    public void useCssSelector(boolean isUse);

    public void useCache(boolean isUse);

    public void useBundle(boolean isUse);

    public void useScreenshot(boolean isUse);

    public void useTrace(boolean isUse);

    public void useTelluriumApi(boolean isUse);
```



```
public void useEngineLog(boolean isUse);

public useMaxMacroCmd(int max);

public int myMaxMacroCmd();

public String myLocale();

public void setCustomEnvironment(String name, Object value);

public Object getCustomEnvironment(String name);
}
```

where `setCustomEnvironment` and `getCustomEnvironment` can be used to pass user defined environment variables.

Tellurium provides the following method for users to dump out all the current environment variables.

```
public void dumpEnvironment();
```

The output is like the following.

```
Environment Variables:
useEngineCache: true
useClosestMatch: false
useMacroCommand: true
maxMacroCmd: 5
useTelluriumApi: true
locatorWithCache: true
useCSSSelector: true
useTrace: true
logEngine: false
locale: en_US
```

If any exception is thrown at the dispatcher tier, the environment variable will also be printed out.

Get UIs by Tag Name

As requested by users, Tellurium 0.7.0 added the following two methods.

```
UiByTagResponse getUiByTag(String tag, Map filters);
void removeMarkedUids(String tag);
```

The first method passes in the tag name and the attributes as filters and get back the UI elements associated with the tag. The response object is defined as

```
class UiByTagResponse {
    //tag name
    String tag;

    Map filters

    //temporally assigned IDs
    String[] tids;
}
```

where the tids are assigned by Tellurium Engine. Under the hood, Tellurium core creates one AllPurposeObject for each tid in the tids array and registers it to Core so that users can use the tid as the UID to act on the UI object.

The second method cleans up all the temporally assigned IDs by the Tellurium Engine.

Example:

In UI module JettyLogonModule, we define the following method:

```
public String[] getInputBox(){
    def attrs = ["type" : "text"];
    UiByTagResponse resp = getUiByTag("input", attrs);

    return resp.tids;
}
```

The test case is as follows

```
@Test
public void testGetUiByTag(){
    useEngineLog(true);
    useTelluriumApi(true);
    useCache(true);
    String[] teuids = jlm.getInputBox();
    assertNotNull(teuids);
    for(String teuid: teuids){
        jlm.keyType(teuid, "Tellurium Source");
    }
    jlm.removeMarkedUids("input");
}
```

Misc

The default locator in Tellurium Engine is CSS selector by default. That is to say, we set

```
exploitCssSelector = true
```

in the Environment class and you can use

```
disableCssSelector()
```

or

```
useCssSelector(false)
```

to switch back to xpath if you like to.

Exposed the following Selenium APIs to DslContext.

- `void addScript(String scriptContent, String scriptTagId)`
- `void removeScript(String scriptTagId)`
- `void captureEntirePageScreenshot(String filename, String kwargs)`
- `String captureScreenshotToString()`
- `String captureEntirePageScreenshotToString(String kwargs)`
- `String retrieveLastRemoteControlLogs()`

The following methods in `DslContext` have been renamed

- `String jsonify(String uid)` is renamed to `String toString(String uid)`
- `String generateHtml(String uid)` is renamed to `String toHTML(String uid)`

Chapter 6. Tellurium Core APIs

Tellurium APIs include all methods defined in `DslContext`. This chapter provides tables showing the various API methods and the action/result of each method when used:

- DSL Methods
- Data Access Methods
- UI Module APIs
- Test Support DSLs

DSL Methods

This section contains a list of all available Tellurium methods that can be used as DSLs in `DslContext` and their corresponding DSL syntax.

Note the id here refers to the UiID in the format of "issueSearch.issueType" and the time units are all in milliseconds, if not specified.

Be aware, the user can only apply the methods to the `Ui` Object if it has these methods previously defined.

- `mouseOver id`: Simulates a user hovering a mouse over the specified element.
- `mouseOut id`: Simulates a user moving the mouse pointer away from the specified element.
- `mouseDown id`: Simulates a user pressing the mouse button (without releasing it yet) on the specified element.
- `click id`: 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`.
- `doubleClick id`: 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`.
- `clickAt id, coordination::`: 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`.
- `check id`: 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`.
- `uncheck id`: Uncheck a toggle-button (checkbox/radio).
- `keyType id, value`: Simulates keystroke events on the specified element, as though the user typed the value key-by-key.
- `type id, input`: Sets the value of an input field, as though typed it in.
- `typeAndReturn id, input`: Sets the value of an input field followed by Return key.
- `clearText id`: Resets input field to an empty value.
- `select id, optionLocator`: Select an option from a drop-down using an option locator.
- `selectByLabel id, optionLocator`: Select an option from a drop-down using an option label.

- `selectByValue id, optionLocator`: Select an option from a drop-down using an option value.
- `addSelectionByLabel id, optionLocator`: Add a selection to the set of selected options in a multi-select element using an option label.
- `addSelectionByValue id, optionLocator`: Add a selection to the set of selected options in a multi-select element using an option value.
- `removeSelectionByLabel id, optionLocator`: Remove a selection from the set of selected options in a multi-select element using an option label.
- `removeSelectionByValue id, optionLocator`: Remove a selection from the set of selected options in a multi-select element using an option value.
- `removeAllSelections id`: Unselects all of the selected options in a multi-select element.
- `pause time`: Suspend the current thread for a specified milliseconds.
- `submit id, attribute`: Submit the specified form. This is particularly useful for forms without submit buttons. For example: single-input "Search" forms.
- `openWindow UID, url`: Opens a popup window. (If a window with that specific ID is not already open). After opening the window, select it using the `selectWindow` command.
- `selectWindow UID`: Selects a popup window; once a popup window has been selected, all commands go to that window. To select the main window again, use null as the target.
- `closeWindow UID`: Close the popup window.
- `selectMainWindows`: Select the original window. For example: the Main window.
- `selectFrame frameName`: Selects a frame within the current window.
- `selectParentFrameFrom frameName`: Select the parent frame from the frame identified by the "frameName".
- `selectTopFrameFrom`: Select the main frame from the frame identified by the "frameName".
- `waitForPopUp UID, timeout`: Waits for a popup window to appear and load up.
- `waitForPageToLoad timeout`: Waits for a new page to load.
- `waitForFrameToLoad uid, timeout`: Waits for a new frame to load. Be aware that Selenium uses the name attribute to locate a Frame.
- `runScript 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 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 are not managed by Selenium, so the user should wrap the script in try/catch blocks if there is any chance that the script will throw an exception.

- `captureScreenshot filename`: Captures a PNG screenshot to the specified file.
- `chooseCancelOnNextConfirmation`: By default, Selenium's overridden `window.confirm()` function returns true, as if the user had manually clicked OK.

After running this command, the next call to `confirm()` returns false, as if the user had clicked Cancel. Selenium then resumes using the default behavior for future confirmations, automatically returning true (OK) unless/until the user explicitly calls this command for each confirmation.

- `chooseOkOnNextConfirmation()`: Undo the effect of calling `chooseCancelOnNextConfirmation()`.

Note: Selenium's overridden `window.confirm()` function normally automatically returns true, as if the user had manually clicked OK. The user does not need to use this command unless for some reason there is a need to change prior to the next confirmation.

After any confirmation, Selenium resumes using the default behavior for future confirmations, automatically returning true (OK) unless/until the user explicitly calls `chooseCancelOnNextConfirmation()` for each confirmation.

- `answerOnNextPrompt(String answer)`: Instructs Selenium to return the specified answer string in response to the next JavaScript prompt `window.prompt()`
- `goBack()`: Simulates the user clicking the "back" button on their browser.
- `refresh()`: Simulates the user clicking the "Refresh" button on their browser.
- `dragAndDrop(WebElement source, WebElement target)`: Drags an element a certain distance and then drops it.
- `dragAndDropTo(WebElement source, WebElement target)`: Drags an element and drops it on another element.

Data Access Methods

In addition to the DSL methods, Tellurium provides Selenium-compatible data access methods so that a user can get data or the status of UIs from the web.

- `String getConsoleInput()`: Gets input from Console.
- `String[] getSelectOptions(String id)`: Gets all option labels in the specified select drop-down.
- `String[] getSelectedLabels(String id)`: Gets all selected labels in the specified select drop-down.
- `String getSelectedLabel(String id)`: Gets a single selected label in the specified select drop-down.
- `String[] getSelectedValues(String id)`: Gets all selected values in the specified select drop-down.
- `String getSelectedValue(String id)`: Gets a single selected value in the specified select drop-down.
- `String[] getSelectedIndexes(String id)`: Gets all selected indexes in the specified select drop-down.
- `String getSelectedIndex(String id)`: Gets a single selected index in the specified select drop-down.
- `String[] getSelectedIds(String id)`: Gets option element ID for selected option in the specified select element.
- `String getSelectedId(String id)`: Gets a single element ID for selected option in the specified select element.
- `boolean isSomethingSelected(String id)`: Determines whether some option in a drop-down menu is selected.

- `String waitForText(id, timeout)`: Waits for a text event.
- `int getTableHeaderColumnNum(id)`: Gets the column header count of a table.
- `int getTableMaxRowNum(id)`: Gets the maximum row count of a table.
- `int getTableMaxColumnNum(id)`: Gets the maximum column count of a table.
- `int getTableFootColumnNum(id)`: Gets the maximum foot column count of a standard table.
- `int getTableMaxTbodyNum(id)`: Gets the maximum tbody count of a standard table.
- `int getTableMaxRowNumForTbody(id, index)`: Gets the maximum row number of the index-th tbody of a standard table.
- `int getTableMaxColumnNumForTbody(id, index)`: Gets the maximum column number of the index-th tbody of a standard table.
- `int getListSize(id)`: Gets the item count of a list.
- `getUiElement(id)`: Gets the UIObject of an element.
- `boolean isElementPresent(id)`: Verifies that the specified element is somewhere on the page.
- `boolean isVisible(id)`: 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 of its ancestors. This method will fail if the element is not present.

- `boolean isChecked(id)`: Gets whether a toggle-button (checkbox/radio) is checked. Fails if the specified element doesn't exist or isn't a toggle-button.
- `boolean isDisabled(id)`: Determines if an element is disabled or not.
- `boolean isEnabled(id)`: Determines if an element is enabled or not.
- `boolean waitForElementPresent(id, timeout)`: Wait for the Ui object to be present.
- `boolean waitForElementPresent(id, timeout, step)`: Wait for the Ui object to be present and check the status by step.
- `boolean waitForCondition(script, 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.
- `String getText(id)`: 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.

- `String getValue(id)`: 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.

- `String getLink(id)`: Get the href of an element.

- `String getImageSource(id)`: Get the image source element.
- `String getImageAlt(id)`: Get the image alternative text of an element.
- `String getImageTitle(id)`: Get the image title of an element.
- `getAttribute(id, attribute)`: Get an attribute of an element.
- `getParentAttribute(id, attribute)`: Get an attribute of the parent of an element.
- `String getBodyText()`: Gets the entire text of the page.
- `boolean isTextPresent(pattern)`: Verifies that the specified text pattern appears somewhere on the rendered page shown to the user.
- `boolean isEditable(id)`: 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.
- `String getHtmlSource()`: Returns the entire HTML source between the opening and closing "html" tags.
- `String getExpression(expression)`: Returns the specified expression.
- `getXpathCount(xpath)`: Returns the number of nodes that match the specified xpath, eg. "//table" would give the number of tables.
- `String getCookie()`: Return all cookies of the current page under test.
- `boolean isAlertPresent()`: Has an alert occurred?
- `boolean isPromptPresent()`: Has a prompt occurred?
- `boolean isConfirmationPresent()`: Has confirm() been called?
- `String getAlert()`: Retrieves the message of a JavaScript alert generated during the previous action, or fail if there were no alerts.
- `String getConfirmation()`: Retrieves the message of a JavaScript confirmation dialog generated during the previous action.
- `String getPrompt()`: Retrieves the message of a JavaScript question prompt dialog generated during the previous action.
- `String getLocation()`: Gets the absolute URL of the current page.
- `String getTitle()`: Gets the title of the current page.
- `String[] getAllButtons()`: Returns the IDs of all buttons on the page.
- `String[] getAllLinks()`: Returns the IDs of all links on the page.
- `String[] getAllFields()`: Returns the IDs of all input fields on the page.
- `String[] getAllWindowIds()`: Returns the IDs of all windows that the browser knows about.
- `String[] getAllWindowNames()`: Returns the names of all windows that the browser knows about.
- `String[] getAllWindowTitles()`: Returns the titles of all windows that the browser knows about.

- `allowNativeXPath(boolean 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.

UI Module APIs

UI Module represents a group of nested UI elements or a UI widget and it is the heart of Tellurium Automated Testing Framework (Tellurium). You may not be aware that Tellurium 0.7.0 provides a set of UI module level APIs. Here we go over them one by one.

Example

First of all, we like to use Google Search module as an example so that everyone can run the test code in this article.

For Google Search Module, we define the UI module class as

```
package org.telluriumsource.module

import org.telluriumsource.dsl.DslContext

public class GoogleSearchModule extends DslContext {

    public void defineUi() {
        ui.Container(uid: "Google", clocator: [tag: "table"]) {
            InputBox(uid: "Input", clocator: [tag: "input", title: "Google Search", name: "q"])
            SubmitButton(uid: "Search", clocator: [tag: "input", type: "submit",
                value: "Google Search", name: "btnG"])
            SubmitButton(uid: "ImFeelingLucky", clocator: [tag: "input", type: "submit",
                value: "I'm Feeling Lucky", name: "btnI"])
        }

        ui.Container(uid: "ProblematicGoogle", clocator: [tag: "table"]) {
            InputBox(uid: "Input", clocator: [tag: "input", title: "Google Search", name: "p"])
            SubmitButton(uid: "Search", clocator: [tag: "input", type: "submit",
                value: "Google Search", name: "btns"])
            SubmitButton(uid: "ImFeelingLucky", clocator: [tag: "input", type: "submit",
                value: "I'm Feeling Lucky", name: "btnf"])
        }
    }

    public void doProblematicGoogleSearch(String input) {
        keyType "ProblematicGoogle.Input", input
        pause 500
        click "ProblematicGoogle.Search"
        waitForPageToLoad 30000
    }
}
```

where UI Module "Google" is a correct UI Module definition for Google Search and "ProblematicGoogle" is a not-so-correct UI module to demonstrate the power of UI module partial matching in Tellurium 0.7.0.

dump

The *dump* method prints out the generated runtime locators by Tellurium core.

API

```
void dump(String uid);
```

where *uid* is the UI module name, i.e., the root element UID of a UI module.

Test case

```
@Test
public void testDump(){
    useCssSelector(false);
    gsm.dump("Google");
    useCssSelector(true);
    gsm.dump("Google");
}
```

Note that here we ask Tellurium core to generate xpath and CSS selector locators, respectively.

Results

```
Dump locator information for Google
-----
Google: //descendant-or-self::table
Google.Input: //descendant-or-self::table/descendant-or-self::
    input[@title="Google Search" and @name="q"]
Google.Search: //descendant-or-self::table/descendant-or-self::
    input[@type="submit" and @value="Google Search" and @name="btnG"]
Google.ImFeelingLucky: //descendant-or-self::table/descendant-or-self::
    input[@type="submit" and @value="I'm Feeling Lucky" and @name="btnI"]
-----

Dump locator information for Google
-----
Google: jquery=table
Google.Input: jquery=table input[title=Google Search][name=q]
Google.Search: jquery=table input[type=submit][value=Google Search]
    [name=btnG]
Google.ImFeelingLucky: jquery=table input[type=submit]
    [value=$m Feeling Lucky][name=btnI]
-----
```

toString

The *toString* method converts a UI module to a JSON presentation. Tellurium Core actually provides two methods for your convenience.

API

```
public String toString(String uid);
public JSONArray toJSONArray(String uid);
```

The *toString* method calls the *toJSONArray* method first under the hood and then prints out the JSON array as a string.

Test case

```
@Test
public void testToString(){
```

```
String json = gsm.toString("Google");
System.out.println(json);
}
```

Results

```
[{"obj":{"uid":"Google","locator":{"tag":"table"},"uiType":"Container"},
 "key":"Google"}, {"obj":{"uid":"Input","locator":{"tag":"input",
 "attributes":{"title":"Google Search","name":"q"},"uiType":
 "InputBox"},"key":"Google.Input"}, {"obj":{"uid":"Search","locator":
 {"tag":"input","attributes":{"name":"btnG","value":"Google Search",
 "type":"submit"},"uiType":"SubmitButton"},"key":"Google.Search"},
 {"obj":{"uid":"ImFeelingLucky","locator":{"tag":"input","attributes":
 {"name":"btnI","value":"I'm Feeling Lucky","type":"submit"},"uiType":
 "SubmitButton"},"key":"Google.ImFeelingLucky"}]}
```

toHTML

The *toHTML* method converts the UI module to a HTML source by reverse engineering [<http://code.google.com/p/aost/wiki/GenerateHtmlFromUIModule>]. This API is extremely useful to work with Tellurium mock http server [<http://code.google.com/p/aost/wiki/TelluriumMockHttpServer>] if we want to diagnose problems in other people's UI module definitions while we have no access to their html sources.

API

```
public String toHTML(String uid);
public String toHTML();
```

The first method generates the HTML source for a UI module and the second one generates the HTML source for all UI modules defined in the current UI module class.

Test Case

```
@Test
public void testToHTML(){
    String html = gsm.toHTML("Google");
    System.out.println(html);
}
```

Result

```
<table>
  <input title="Google Search" name="q"/>
  <input type="submit" value="Google Search" name="btnG"/>
  <input type="submit" value="I'm Feeling Lucky" name="btnI"/>
</table>
```

getHTMLSource

The *getHTMLSource* method returns the runtime HTML source for a UI module.

API

```
java.util.List getHTMLSourceResponse(String uid);
void getHTMLSource(String uid);
```

The first method gets back the HTML source as a list of *key* and *val* pair and the second one prints out them to console.

```
class KeyValuePair {
    public static final String KEY = "key";
    String key;

    public static final String VAL = "val";
    String val;
}
```

Test Case

```
@Test
public void testGetHTMLSource(){
    gsm.getHTMLSource("Google");
}
```

Result

```
TE: Name: getHTMLSource, start: 1266260182744, duration: 67ms
TE: Found exact match for UI Module 'Google': {"id":"Google","relaxDetails":
    [], "matches":1, "relaxed":false, "score":100.0, "found":true}
Google:
```

```
<table cellpadding="0" cellspacing="0"><tbody><tr valign="top"><td width="25%">
    &nbsp;</td><td nowrap="nowrap" align="center"><input name="hl" value="en"
    type="hidden"><input name="source" value="hp" type="hidden"><input
    autocomplete="off" onblur="google&amp;&amp;google.fade&amp;&amp;google.fade()"
    maxlength="2048" name="q" size="55" class="lst" title="Google Search"
    value=""><br><input name="btnG" value="Google Search" class="lsb"
    onclick="this.checked=1" type="submit"><input name="btnI"
    value="I'm Feeling Lucky" class="lsb" onclick="this.checked=1"
    type="submit"></td><td id="sbl" nowrap="nowrap" width="25%" align="left">
    <font size="-2">&nbsp;&nbsp;<a href="/advanced_search?hl=en">Advanced Search
    </a><br>&nbsp;&nbsp;<a href="/language_tools?hl=en">Language Tools</a>
    </font></td></tr></tbody></table>
```

Google.Input:

```
<input autocomplete="off" onblur="google&amp;&amp;google.fade&amp;&amp;
    google.fade()" maxlength="2048" name="q" size="55" class="lst"
    title="Google Search" value="">
```

Google.Search:

```
<input name="btnG" value="Google Search" class="lsb" onclick=
    "this.checked=1" type="submit">
```

Google.ImFeelingLucky:

```
<input name="btnI" value="I'm Feeling Lucky" class="lsb"
    onclick="this.checked=1" type="submit">
```

show

The *show* method outlines the UI module on the web page under testing and shows some visual effects [http://code.google.com/p/aost/wiki/TelluriumUiModuleVisualEffect] when a user mouses over it.

API

```
void show(String uid, int delay);  
void startShow(String uid);  
void endShow(String uid);
```

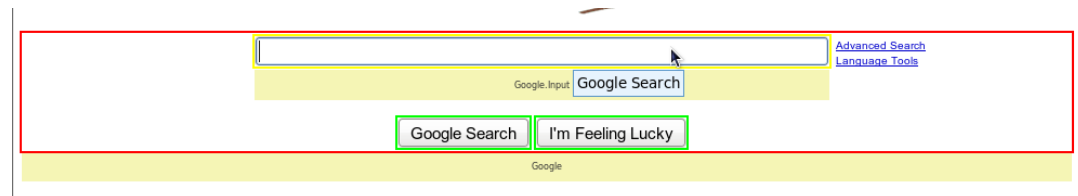
The first method shows the UI module visual effect for some time (delay in milliseconds). The other two methods are used to manually start and end the visual effect.

Test Case

```
@Test  
public void testShow(){  
    gsm.show("Google", 10000);  
    // gsm.startShow("Form");  
    // gsm.endShow("Form");  
}
```

Result

The visual effect is illustrated in the following graph.



validate

The *validate* method is used to validate if the UI module is correct and returns the mismatches at runtime.

API

```
UiModuleValidationResponse getUiModuleValidationResult(String uid);  
void validate(String uid)
```

The first method returns the validation result as a `UiModuleValidationResponse` object,

```
public class UiModuleValidationResponse {  
  
    //ID for the UI module  
    public static String ID = "id";  
    private String id = null;  
  
    //Successfully found or not
```

```

    public static String FOUND = "found";
    private boolean found = false;

    //whether this the UI module used closest Match or not
    public static String RELAXED = "relaxed";
    private boolean relaxed = false;

    //match count
    public static String MATCHCOUNT = "matches";
    private int matches = 0;

    //scaled match score (0-100)
    public static String SCORE = "score";
    private float score = 0.0;

    public static String RELAXDETAIL = "relaxDetail";
    //details for the relax, i.e., closest match
    public static String RELAXDETAILS = "relaxDetails";
    private List<RelaxDetail> relaxDetails = null;
    ...
}

```

and the RelaxDetail is defined as follows,

```

public class RelaxDetail {
    //which UID got relaxed, i.e., closest Match
    public static String UID = "uid";
    private String uid = null;

    //the clocator defintion for the UI object corresponding to the UID
    public static String LOCATOR = "locator";
    private CompositeLocator locator = null;

    //The actual html source of the closest match element
    public static String HTML = "html";
    private String html = null;
    ...
}

```

The second one simply prints out the result to console.

Test Case

```

@Test
public void testValidate(){
    gsm.validate("Google");
    gsm.validate("ProblematicGoogle");
}

```

Here we validate the correct UI module "Google" and the not-so-correct UI module "ProblematicGoogle".

Result

```
TE: Name: getValidateUiModule, start: 1266260203639, duration: 68ms
```

```
UI Module Validation Result for Google
```

```

-----
Found Exact Match: true

```

```
Found Closest Match: false

Match Count: 1

Match Score: 100
```

```
-----

TE: Name: getValidateUiModule, start: 1266260203774, duration: 41ms

UI Module Validation Result for ProblematicGoogle

-----
```

```
Found Exact Match: false

Found Closest Match: true

Match Count: 1

Match Score: 32.292

Closest Match Details:

--- Element ProblematicGoogle.Input -->

Composite Locator: <input title="Google Search" name="p"/>

Closest Matched Element: <input autocomplete="off" onblur=
  "google&amp;&amp;google.fade&amp;&amp;google.fade()"
  maxlength="2048" name="q" size="55" class="lst"
  title="Google Search" value="">

--- Element ProblematicGoogle.Search -->

Composite Locator: <input name="btns" value="Google Search"
  type="submit"/>

Closest Matched Element: <input name="btnG" value=
  "Google Search" class="lsb" onclick="this.checked=1"
  type="submit">

--- Element ProblematicGoogle.ImFeelingLucky -->

Composite Locator: <input name="btnf" value="I'm Feeling Lucky"
  type="submit"/>

Closest Matched Element: <input name="btnI" value=
  "I'm Feeling Lucky" class="lsb" onclick="this.checked=1"
  type="submit">

-----
```

As we can see, the correct UI module returns a score as 100 and the problematic UI module returns a score less than 100.

Closest Match

The UI module closest match, i.e., partial match, is extremely important to keep your test code robust to changes to some degree. By partial match, Tellurium uses the Santa algorithm [<http://code.google.com/p/aost/wiki/SantaUiModuleGroupLocatingAlgorithm>] to locate the whole UI module by using only partial information inside the UI module.

API

```
enableClosestMatch();
disableClosestMatch();
useClosestMatch(boolean isUse);
```

The first two methods are used in `DslContext` to enable and disable the closest match feature. The last one is used on Groovy or Java test case for the same purpose. Be aware that this feature only works with UI module caching, i.e., you should call either

```
useTelluriumEngine(true);
```

or

```
useCache(true);
```

before use this feature.

Test Case

Here we run our test code based on the problematic UI module "ProblematicGoogle".

```
@Test
public void testClosestMatch(){
    useClosestMatch(true);
    gsm.doProblematicGoogleSearch("Tellurium Source");
    useClosestMatch(false);
}
```

Result

The test case passed successfully. What happens if you disable the closest match feature by calling

```
useClosestMatch(true);
```

The above test case will break and you will get a red bar in IDE.

Test Support DSLs

Tellurium defines a set of DSLs to support Tellurium tests. The most often used ones include:

`connectSeleniumServer()`: Connect to the selenium server.

`disconnectSeleniumServer()`: Disconnect the connection to the selenium server.

`openUrl(String url)`: establish a new connection to Selenium server for the given url. The DSL format is:

```
openUrl url
```


Example:

```
openUrl "http://code.google.com/p/aost/"
```

`connectUrl(String url)`: use existing connect for the given url. The DSL format is:

```
connectUrl url
```

Example:

```
connectUrl "http://code.google.com/p/aost/"
```

`openUrlWithBrowserParameters()` is used to change browser settings for different test cases in the same test class. There are three methods:

```
public static void openUrlWithBrowserParameters(String url, String serverHost,
    int serverPort, String baseUrl, String browser, String browserOptions)

public static void openUrlWithBrowserParameters(String url, String serverHost,
    int serverPort, String browser, String browserOptions)

public static void openUrlWithBrowserParameters(String url, String serverHost,
    int serverPort, String browser)
```

For example,

```
public class GoogleStartPageTestNGTestCase extends TelluriumTestNGTestCase {
    protected static NewGoogleStartPage ngsp;

    @BeforeClass
    public static void initUi() {
        ngsp = new NewGoogleStartPage();
        ngsp.defineUi();
    }

    @DataProvider(name = "browser-provider")
    public Object[][] browserParameters() {
        return new Object[][]{
            new Object[] {"localhost", 4444, "*chrome"},
            new Object[] {"localhost", 4444, "*firefox"}};
    }

    @Test(dataProvider = "browser-provider")
    @Parameters({"serverHost", "serverPort", "browser"})
    public void testGoogleSearch(String serverHost, int serverPort,
        String browser){
        openUrlWithBrowserParameters("http://www.google.com", serverHost,
            serverPort, browser);
        ngsp.doGoogleSearch("tellurium selenium Groovy Test");
        disconnectSeleniumServer();
    }

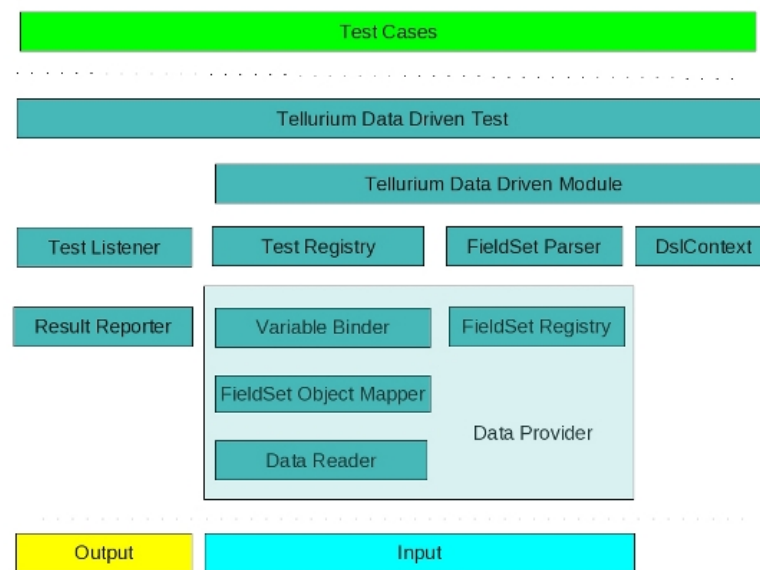
    @Test(dataProvider = "browser-provider")
    @Parameters({"serverHost", "serverPort", "browser"})
    public void testGoogleSearchFeelingLucky(String serverHost, int serverPort,
        String browser){
```

```
        openUrlWithBrowserParameters("http://www.google.com", serverHost,
            serverPort, browser);
        ngsp.doImFeelingLucky("tellurium selenium DSL Testing");
        disconnectSeleniumServer();
    }
}
```

Chapter 7. Tellurium Core Advanced Topics

Data Driven Testing

Data Driven Testing is a different way to write tests. For example, separate test data from the test scripts and the test flow is not controlled by the test scripts, but by the input file instead. In the input file, users can specify which tests to run, what are input parameters, and what are expected results. Data driven testing in Tellurium is illustrated in Figure 2-5 with the following system diagram:



The Tellurium Data Driven Test consists of three main parts:

1. Data Provider
2. TelluriumDataDrivenModule
3. TelluriumDataDrivenTest

Data Provider

The Data Provider is responsible for reading data from input stream and converting data to Java variables.

Tellurium includes the following Data Provider methods:

1. loadData file_name, load input data from a file
2. useData String_name, load input data from a String in the test script
3. bind(field_name), bind a variable to a field in a field set
4. closeData, close the input data stream and report the test results
5. cacheVariable(name, variable), put variable into cache

6. `getCachedVariable(name, variable)`, get variable from cache where the `file_name` includes the file path. For example:

```
loadData "src/test/example/test/ddt/GoogleBookListCodeHostInput.txt"
```

Tellurium supports pipe format and CSV format input file. To change the file reader for different formats, change the following settings in the configuration file `TelluriumConfig.groovy`:

```
datadriven{
    dataprovider{
        //specify which data reader you like the data provider to use
        //the valid options include:
        // "PipeFileReader", "CSVFileReader", "ExcelFileReader"
        //at this point
        reader = "PipeFileReader"
    }
}
```

Tellurium's `useData` is designed to specify test data in the test scripts directly. It loads input from a String. The String is usually defined in Groovy style using triple quota, for example:

```
protected String data = """
google_search | true | 865-692-6000 | tellurium
google_search | false | 865-123-4444 | tellurium selenium test
google_search | true | 755-452-4444 | tellurium groovy
google_search | false | 666-784-1233 | tellurium user group
google_search | true | 865-123-5555 | tellurium data driven
"""
...

useData data
```

`bind` is the command used to bind a variable to an input Field Set field at runtime. `FieldSet` is the format of a line of data. For example:

```
def row = bind("GCHLabel.row")
```

is used to bind the `row` variable to the "row" field in the `FieldSet` "GCHLabel". Tellurium does not explicitly differentiate input parameters from the expected results in the input data. To bind variables to the input data then use any of them as the expected results for result comparison.

`cacheVariable` and `getCachedVariable` are used to pass intermediate variables among tests.

- `cacheVariable` is used to put a variable into a cache
- `getCachedVariable` is used to get back the variable

For example:

```
int headernum = getTableHeaderNum()
cacheVariable("headernum", headernum)

...
```

```
int headernum = getCachedVariable("headernum")
...
```

When testing is completed, use "closeData" to close the input data stream. In the meantime, the result reporter outputs the test results in the format specified in the configuration file.

For example: the XML file as shown in the TelluriumConfig.groovy file:

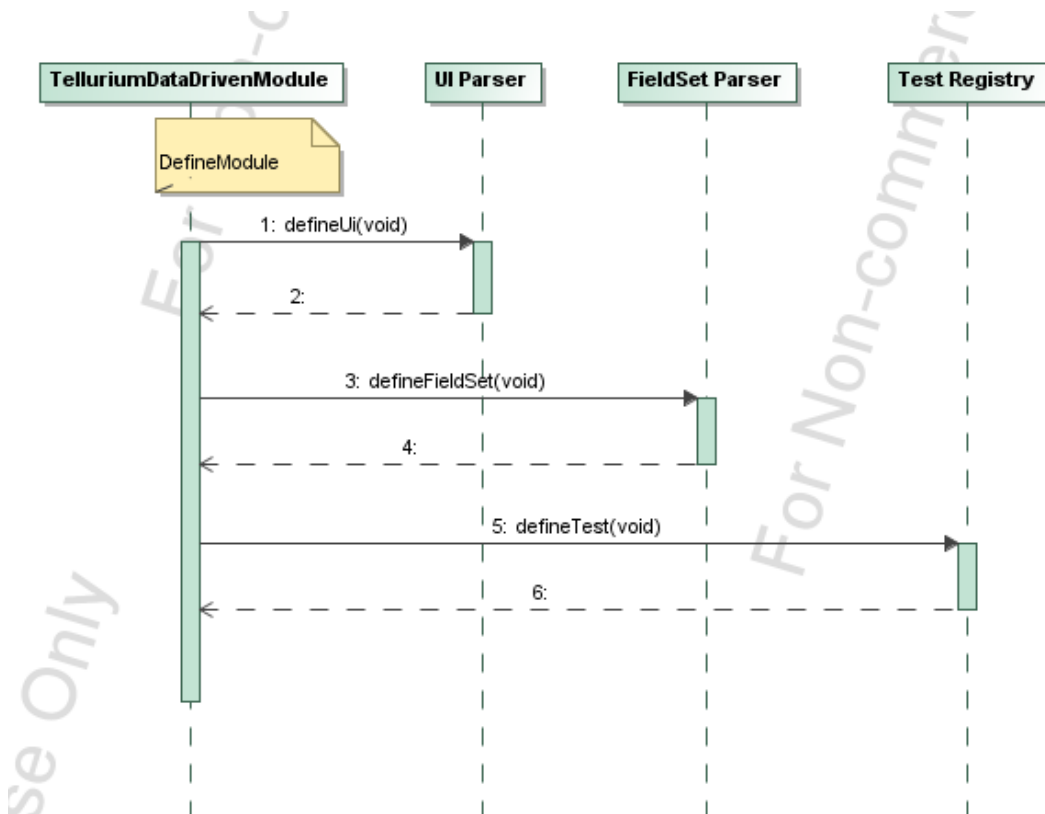
```
test{
  result{
    //specify what result reporter used for the test result
    //valid options include "SimpleResultReporter", "XMLResultReporter",
    //and "StreamXMLResultReporter"
    reporter = "XMLResultReporter"

    //the output of the result
    //valid options include "Console", "File" at this point
    //if the option is "File", you need to specify the file name,
    //other wise it will use the default
    //file name "TestResults.output"
    output = "Console"

    //test result output file name
    filename = "TestResult.output"
  }
}
```

TelluriumDataDrivenModule

TelluriumDataDrivenModule is used to define modules, where users can define UI Modules, FieldSets, and tests as shown in the following Figure sequence diagram. Users should extend this class to define their own test modules.



TelluriumDataDrivenModule provides one method "defineModule" for users to implement. Since it extends the DslContext class, users define UI modules as in regular Tellurium UI Modules. For example:

```
ui.Table(uid: "labels_table", clocator: [:], group: "true"){
  TextBox(uid: "{row: 1, column: 1} as Label", clocator: [tag: "div",
    text: "Example project labels:"])
  Table(uid: "{row: 2, column: 1}", clocator: [header: "/div[@id=\"popular\"]"]){
    UrlLink(uid: "{row: all, column: all}", locator: "/a")
  }
}
```

FieldSet defines the format of one line of input data. FieldSet consists of fields such as columns, in the input data. There is a special field "test", where users can specify what tests this line of data applies to. For example:

```
fs.FieldSet(name: "GCHStatus", description: "Google Code Hosting input") {
  Test(value: "getGCHStatus")
  Field(name: "label")
  Field(name: "rowNum", type: "int")
  Field(name: "columnNum", type: "int")
}
```

FieldSet defines the input data format for testing Google code hosting web page.

Note: The Test field must be the first column of the input data.

The default name of the test field is "test" and does not need to be specified. If the value attribute of the test field is not specified, it implies this same format. For example, FieldSet is used for different tests.

A regular field includes the following attributes:

```
class Field {
  //Field name
  private String name

  //Field type, default is String
  private String type = "String"

  //optional description of the Field
  private String description

  //If the value can be null, default is true
  private boolean nullable = true

  //optional null value if the value is null or not specified
  private String nullValue

  //If the length is not specified, it is -1
  private int length = -1

  //optional String pattern for the value
  //if specified, use it for String validation
  private String pattern
}
```

Tellurium can automatically handle Java primitive types.

Another flexibility Tellurium provides is allowing users to define their own custom type handlers to deal with more complicated data types by using "typeHandler". For example:

```
//define custom data type and its type handler

typeHandler "phoneNumber", "org.tellurium.test.PhoneNumberTypeHandler"

//define file data format
fs.FieldSet(name: "fs4googlesearch", description: "example field set for google search"){
    Field(name: "regularSearch", type: "boolean",
        description: "whether we should use regular search or use I'm feeling lucky")
    Field(name: "phoneNumber", type: "phoneNumber", description: "Phone number")
    Field(name: "input", description: "input variable")
}
```

The above script defines a custom type "PhoneNumber" and the Tellurium automatically calls this type handler to convert the input data to the "PhoneNumber" Java type.

The "defineTest" method is used to define a test in the TelluriumDataDrivenModule. For example, the following script defines the "clickGCHLabel" test:

```
defineTest("clickGCHLabel"){
    def row = bind("GCHLabel.row")
    def column = bind("GCHLabel.column")

    openUrl("http://code.google.com/hosting/")
    click "labels_table[2][1].[${row}][${column}]"

    waitForPageToLoad 30000
}
```

Note: The bind command binds variables row, column to the fields "row" and "column" in the FieldSet "GCHLabel".

Tellurium also provides the command "compareResult" for users to compare the actual result with the expected result. For example, the following script compares the expected label, row number, and column number with the actual ones at runtime:

```
defineTest("getGCHStatus"){
    def expectedLabel = bind("GCHStatus.label")
    def expectedRowNum = bind("GCHStatus.rowNum")
    def expectedColumnNum = bind("GCHStatus.columnNum")

    openUrl("http://code.google.com/hosting/")
    def label = getText("labels_table[1][1]")
    def rownum = getTableMaxRowNum("labels_table[2][1]")
    def columnnum = getTableMaxColumnNum("labels_table[2][1]")

    compareResult(expectedLabel, label)
    compareResult(expectedRowNum, rownum)
    compareResult(expectedColumnNum, columnnum)
    pause 1000
}
```

Sometimes users may require custom "compareResult" to handle more complicated situations. For example, when users compare two lists, users can override the default "compareResult" behaviour by specifying custom code in the closure:

```
compareResult(list1, list2){
    assertTrue(list1.size() == list2.size())
    for(int i=0; i<list1.size();i++){
        //put your custom comparison code here
    }
}
```

```
}
```

If users want to check a variable in the test, the "checkResult" method is used coming with a closure where users define the actual assertions inside:

```
checkResult(issueTypeLabel) {
    assertTrue(issueTypeLabel != null)
}
```

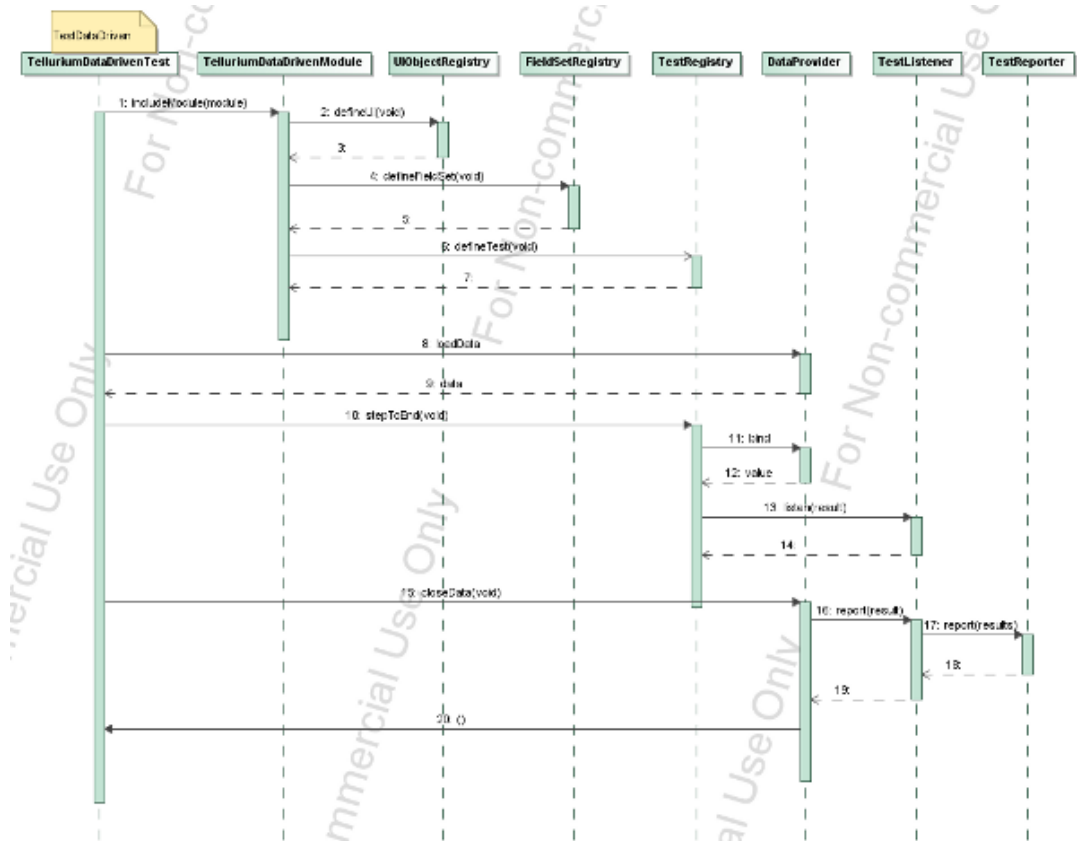
Like "compareResult", "checkResult" captures all assertion errors. The test resumes even when the assertions fail. The result is reported in the output.

In addition, the "logMessage" is used by users to log any messages in the output.

```
logMessage "Found ${actual.size()} ${issueTypeLabel} for owner " + issueOwner
```

Tellurium Data Driven Test

TelluriumDataDrivenTest is the class users should extend to run the actual data driven testing. It is more like a data driven testing engine. There is only one method, "testDataDriven", which users implement. The sequence diagram for the testing process is shown in the following Figure:



Complete the following steps to use TelluriumDataDrivenTest:

1. Use "includeModule" to load defined Modules
2. Use "loadData" or "useData" to load input data stream

3. Use "stepToEnd" to read the input data line by line and pick up the specified test and run it, until reaches the end of the data stream
4. Use "closeData" to close the data stream and output the test results

What the "includeModule" does is to merge in all Ui modules, FieldSets, and tests defined in that module file to the global registry.

"stepToEnd" looks at each input line, first find the test name and pass in all input parameters to it, and then run the test. The whole process is illustrated in the following example:

```
class GoogleBookListCodeHostTest extends TelluriumDataDrivenTest{

    public void testDataDriven() {
        includeModule example.google.GoogleBookListModule.class
        includeModule example.google.GoogleCodeHostingModule.class
        //load file
        loadData "src/test/example/test/ddt/GoogleBookListCodeHostInput.txt"

        //read each line and run the test script until the end of the file
        stepToEnd()

        //close file
        closeData()
    }
}
```

The input data for this example are as follows:

```
##TEST should be always be the first column

##Data for test "checkBookList"
##TEST | CATEGORY | SIZE
checkBookList|Fiction|8
checkBookList|Fiction|3

##Data for test "getGCHStatus"
##TEST | LABEL | Row Number | Column Number
getGCHStatus |Example project labels:| 3 | 6
getGCHStatus |Example project| 3 | 6

##Data for test "clickGCHLabel"
##TEST | row | column
clickGCHLabel | 1 | 1
clickGCHLabel | 2 | 2
clickGCHLabel | 3 | 3
```

Note: The line starting with "##" is the comment line and the empty line is ignored.

If users want to control the testing execution flow by themselves, Tellurium also provides this capability even though its use is **not recommended**.

Tellurium provides two additional commands, "step" and "stepOver".

- "step" is used to read one line of input data and run it.
- "stepOver" is used to skip one line of input data.

In this meanwhile, Tellurium also allows the user to specify additional test scripts using closure. For example:

```
step{
    //bind variables
```

```

boolean regularSearch = bind("regularSearch")
def phoneNumber = bind("fs4googlesearch.phoneNumber")
String input = bind("input")
openUrl "http://www.google.com"
type "google_start_page.searchbox", input
pause 500
click "google_start_page.googlesearch"
waitForPageToLoad 30000
}

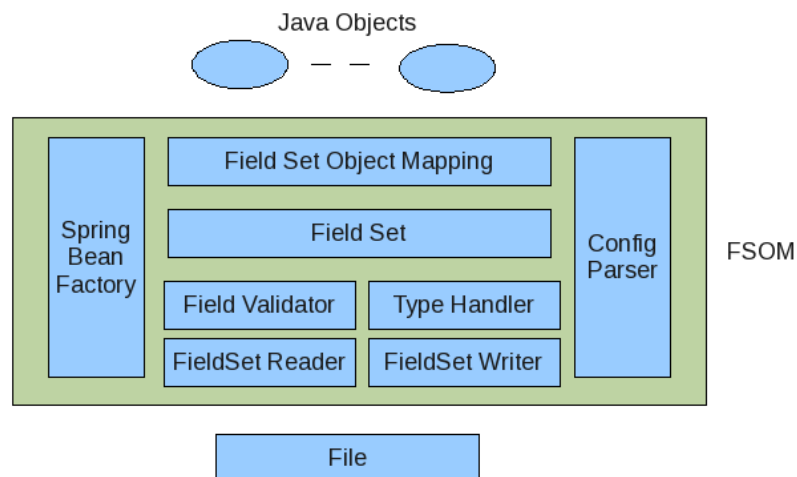
```

This usually implies that the input data format is unique or the test script knows about what format the current input data are using.

Implementations

Historically, the Tellurium Data Driven evolved from the Field Set Object Mapping Framework (FSOM) [<http://code.google.com/p/jianwikis/wiki/FieldSetObjectMappingFramework>] Jian worked for a batch processor. The FSOM framework acts as a converter layer, i.e., to convert Java Objects to and from field sets. What are field sets? A field set is a set of data to represent the format of a flat file or other types of files.

The Architecture of the FSOM framework is shown in the following diagram,



As you can see from the above architecture, Tellurium inherited a lot of concepts from the FSOM framework. The major move from the FSOM framework to Tellurium Data Driven Testing is the adoption of the Groovy BuilderSupport class, which removed the need for the Spring bean factory and XML configuration parser. The Groovy BuilderSupport class makes it very easy to write the Field Set in an expressive way. For example,

```

//define custom data type and its type handler

typeHandler "phoneNumber", "org.tellurium.test.PhoneNumberTypeHandler"

//define file data format
fs.FieldSet(name: "fs4googlesearch", description: "example field set for google search"){
    Field(name: "regularSearch", type: "boolean",
        description: "whether we should use regular search or use I'm feeling lucky")
    Field(name: "phoneNumber", type: "phoneNumber", description: "Phone number")
    Field(name: "input", description: "input variable")
}

```

Note that the above script defines a custom type "PhoneNumber" and Tellurium automatically calls this type handler to convert the input data to the "PhoneNumber" Java type.

Under the hood, Tellurium uses the Groovy BuilderSupport class to parse the nested objects and here is the code for that purpose.

```
class FieldSetParser extends BuilderSupport{
    protected final static String FIELD_SET = "FieldSet"
    protected final static String FIELD = "Field"
    protected final static String IDENTIFIER = "Identifier"
    protected final static String TEST = "Test"

    private FieldSetRegistry registry

    public FieldSetParser(FieldSetRegistry registry){
        this.registry = registry
    }

    private FieldBuilder fb = new FieldBuilder()
    private FieldSetBuilder fsb = new FieldSetBuilder()
    private IdentifierFieldBuilder fsi = new IdentifierFieldBuilder()
    private TestFieldBuilder afb = new TestFieldBuilder()

    protected void setParent(Object parent, Object child) {
        if (parent instanceof FieldSet) {
            FieldSet fs = (FieldSet)parent
            fs.addField(child)
        }
    }

    protected Object createNode(Object name) {
        if(FIELD_SET.equalsIgnoreCase(name))
            return new FieldSet()
        if(FIELD.equalsIgnoreCase(name))
            return new Field()
        if(IDENTIFIER.equalsIgnoreCase(name))
            return new IdentifierField()
        if(TEST.equalsIgnoreCase(name))
            return new TestField()

        return null
    }

    protected Object createNode(Object name, Object value) {
        return null
    }

    protected Object createNode(Object name, Map map) {
        if(FIELD_SET.equalsIgnoreCase(name))
            return fsb.build(map)
        if(FIELD.equalsIgnoreCase(name))
            return fb.build(map)
        if(IDENTIFIER.equalsIgnoreCase(name))
            return fsi.build(map)
        if(TEST.equalsIgnoreCase(name))
            return afb.build(map)

        return null
    }

    protected Object createNode(Object name, Map map, Object value) {
        if(FIELD_SET.equalsIgnoreCase(name))
            return fsb.build(map, (Closure)value)

        return null
    }

    protected void nodeCompleted(Object parent, Object node) {
        //when the node is completed and it is a FieldSet, put it into the registry
        if (node instanceof FieldSet) {

            FieldSet fs = (FieldSet)node

            //need to check if the identifier is presented
            fs.checkFields()

            //only put the top level nodes into the registry
            registry.addFieldSet(fs)
        }
    }
}
```

```

    }
  }
}

```

Selenium Grid Support

Selenium Grid transparently distributes tests on multiple machines so that the tests are run in parallel. Recently support for the Selenium Grid has been added to Tellurium. Now Tellurium tests can be run against different browsers using Selenium Grid. Tellurium core is updated.

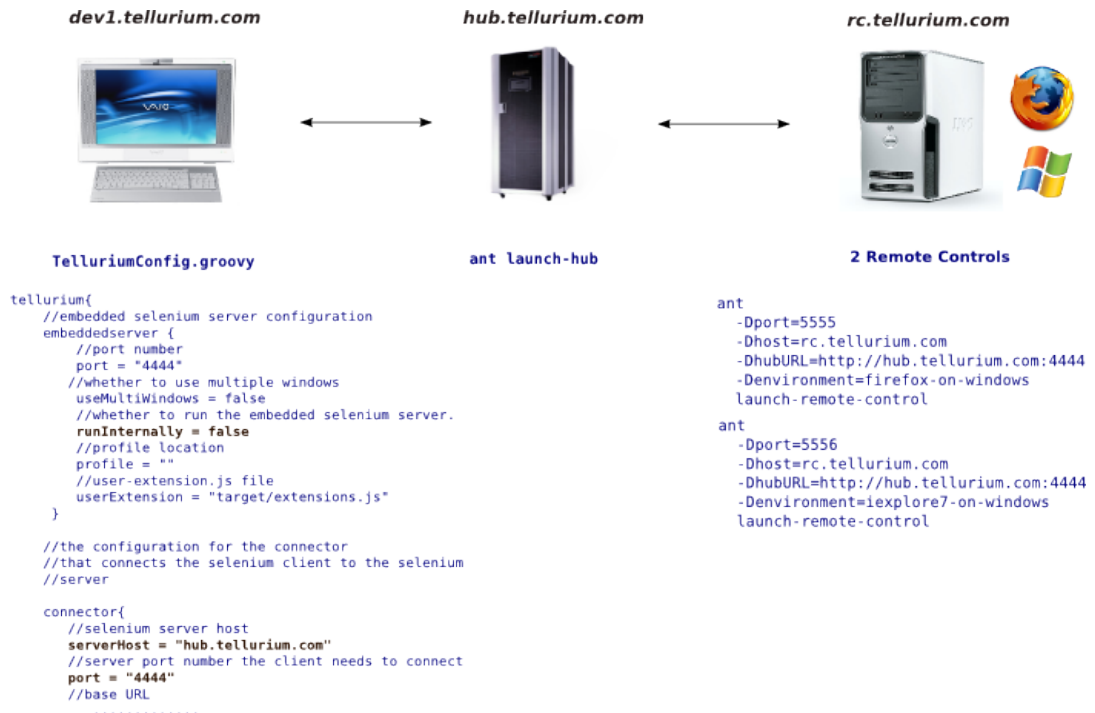
For example, assume 3 machines are set up to run Tellurium tests on the Selenium Grid. All the steps can be completed on the user's local box. To do this locally, remove the machine names with `localhost`. Each machine in this set up has a defined role as described below:

1. **dev1.tellurium.com** Tellurium test development machine.
2. **hub.tellurium.com** Selenium Grid hub machine that drives the tests.
3. **rc.tellurium.com** Multiple Selenium RC server running and registered to the Selenium Grid HUB.

The actual test execution is completed on this machine. Register as many Selenium RC servers as required. However, be realistic about the hardware specification.

Download the Selenium Grid from the following URL and extract the contents of the folder on each of these machines.

Tellurium uses Selenium Grid 1.0.3, the current released version. <http://selenium-grid.seleniumhq.org/download.html> [http://selenium-grid.seleniumhq.org/download.html]. Figure 2-8 shows an illustration of the environment.



The first step would be to launch the selenium grid hub on the hub machine. Open up a terminal on the HUB machine `hub.tellurium.com` and go to the download directory of Selenium Grid.

```
> cd /Tools/selenium-grid-1.0.3
> ant launch-hub
```

Result: The Selenium HUB is launched on the machine with different browsers. To ensure that the HUB is working properly go to the following location.

Navigate to the following URL location to ensure that the HUB is working properly: <http://hub.tellurium.com:4444/console> [<http://hub.tellurium.com:4444/console>]

View the web page with 3 distinct columns:

1. a Configured Environments
2. Available Remote Controls
3. Active Remote Controls

Have a list of browsers configured by default to run the tests while the list for Available Remote Controls and Active Remote Controls is empty.

Launch the Selenium RC servers and register them with the selenium HUB. Open up a terminal on rc.tellurium.com and go to the selenium grid download directory.

```
> cd /Tools/selenium-grid-1.0.3
> ant -Dport=5555 -Dhost=rc.tellurium.com -DhubURL=http://hub.tellurium.com:4444 \
    -Denvironment="Firefox on Windows" launch-remote-control
```

Result: The command starts a Selenium RC server on this machine.

Register the Selenium RC server with the Selenium Grid hub machine as specified by the hubURL.

Note: To register another Selenium RC server on this machine for internet explorer repeat the step on a different port.

```
> cd /Tools/selenium-grid-1.0.3
> ant -Dport=5556 -Dhost=rc.tellurium.com -DhubURL=http://hub.tellurium.com:4444 \
    -Denvironment="IE on Windows" launch-remote-control
```

1. *port* the remote control is listening to. Must be unique on the machine the remote control runs from.
2. *hostname* Hostname or IP address of the machine the remote control runs on. Must be visible from the Hub machine.
3. *hub url* Which hub the remote control should register/unregister to. As the hub is running on hostname hub.tellurium.com, the URL is <http://hub.tellurium.com:4444> [<http://hub.tellurium.com:4444>]

Point your browser to the Hub console Once you are successful in replicating a setup similar to the one described above, (<http://hub.tellurium.com:4444/console> [<http://hub.tellurium.com:4444/console>]).

Verify that all the remote controls registered correctly. Available remote controls list should be updated and have the 2 selenium servers available to run the tests.

Run the Tellurium tests against different browsers once the Selenium Hub and the Selenium RC servers on the Grid environment have started.

Go to the Tellurium test development machine, the **dev1.tellurium.com**.

Open up the `TelluriumConfig.groovy`. Change the values of the Selenium server and port to ensure the Tellurium requests for the new sessions from the Selenium HUB are received. Verify that the Selenium HUB points to Tellurium tests run on `rc.tellurium.com` based on the browser of choice.

Change the values for the following properties:

1. *runInternally* : ensures that the Selenium Server on the local machine is not launched.
2. *serverHost* : the selenium grid hub machine that has the information about the available selenium rc servers.
3. *port* : port that Selenium HUB is running on. By default, this port is 4444. This can be changed in the `grid_configuraton.yml` file if this port is not available on your HUB machine.
4. *browser* : the browser that comes under the configured environments list on the selenium HUB machine. These values can be changed to a user's choice in the `grid_configuration.yml` file.

```
tellurium{

    //embedded selenium server configuration
    embeddedserver {

        //port number
        port = "4444"

        //whether to use multiple windows
        useMultiWindows = false

        //whether to run the embedded selenium server.
        //If false, you need to manually set up a selenium server
        runInternally = false

        //profile location
        profile = ""

        //user-extension.js file
        userExtension = "target/classes/extension/user-extensions.js"
    }

    //event handler
    eventhandler{

        //whether we should check if the UI element is presented
        checkElement = false

        //wether we add additional events like "mouse over"
        extraEvent = true
    }

    //data accessor
    accessor{
        //whether we should check if the UI element is presented
        checkElement = true
    }

    //the configuration for the connector that connects the selenium client
    //to the selenium server
    connector{
        //selenium server host
        //please change the host if you run the Selenium server remotely
        serverHost = "hub.tellurium.com"

        //server port number the client needs to connect
        port = "4444"

        //base URL
        baseUrl = "http://localhost:8080"
    }
}
```

```
//Browser setting, valid options are
// *firefox [absolute path]
// *iexplore [absolute path]
// *chrome
// *iehta
browser = "Firefox on Windows"

//user's class to hold custom selenium methods associated with user-extensions.js
//should in full class name, for instance, "com.mycom.CustomSelenium"
customClass = "org.tellurium.test.MyCommand"
}
```

The set up is now complete. Run the tests as usual using either the Maven command or the IDE. Notice that the tests are running on rc.tellurium.com and the list for Active Remote Controls is also updated on the hub URL (<http://hub.tellurium.com:4444/console> [<http://hub.tellurium.com:4444/console>]) during the test execution.

Mock Http Server

This feature only exists in Tellurium Core 0.7.0. The MockHttpServer is an embedded http server leveraging the Java 6 http server and it is very convenient method of testing HTML sources directly without running a web server.

Tellurium defines two classes:

1. MockHttpHandler
2. MockHttpServer

Mock Http Handler Class

The MockHttpHandler class processes the http request:

```
public class MockHttpHandler implements HttpHandler {

    private Map<String, String> contents = new HashMap<String, String>();

    private String contentType = "text/html";

    public void handle(HttpExchange exchange) {
        .....
    }
}
```

The MockHttpHandler method is handle (HttpExchange exchange) and its actions are:

- Reads the request URI
- Finds the corresponding response HTML source from the hash map contents
- Sends the response back to the http client

By default, the response is treated as an HTML source. The user can change this by using the following setter:

```
public void setContentType(String contentType)
```

MockHttpHandler includes two methods to add URI and its HTML source to the hash map contents:

1. public void registerBody(String url, String body)
2. public void registerHtml(String url, String html)

The MockHttpHandler comes with a default HTML template as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Mock HTTP Server</title>
  </head>

  <body>
    BODY_HTML_SOURCE
  </body>
</html>
```

If registerBody(String url, String body) is used, the MockHttpHandler uses the above HTML template to wrap the HTML body. Overwrite the default HTML template by calling registerHtml(String url, String html) directly, which uses the whole HTML source provided in the variable "html".

Usually, the MockHttpHandler is encapsulated by the MockHttpServer and the user does not need to work on it directly.

The MockHttpServer includes an embedded http server, a http handler, and a http port:

```
public class MockHttpServer {

    //default port
    private int port = 8080;

    private HttpServer server = null;
    private MockHttpHandler handler;

    public MockHttpServer() {
        this.handler = new MockHttpHandler();
        this.server = HttpServer.create();
    }

    public MockHttpServer(int port) {
        this.handler = new MockHttpHandler();
        this.port = port;
        this.server = HttpServer.create();
    }

    public MockHttpServer(int port, HttpHandler handler) {
        this.port = port;
        this.handler = handler;
        this.server = HttpServer.create();
    }

    .....
}
```

Mock Http Server

The MockHttpServer provides three different constructors so the user can overwrite the default values. The MockHttpServer encapsulates the MockHttpHandler by providing the following methods:

1. public void setContentTypes(String contentType)
2. public void registerHtmlBody(String url, String body)
3. public void registerHtml(String url, String html)

The user can stop and start the server with the following methods:

1. public void start()
2. public void stop()

Use a modified version of a HTML source provided by one Tellurium user as an example and create the UI module Groovy class as follows:

```
public class ListModule extends DslContext {

    public static String LIST_BODY = ""
    <div class="thumbnails">
        <ul>
            <li class="thumbnail">
                
            </li>
            <li class="thumbnail">
                
            </li>
            <li class="thumbnail">
                
            </li>
            <li class="thumbnail">
            </li>
            <li class="thumbnail active">
                
            </li>
            <li class="thumbnail potd">
                <div class="potd-icon png-fix"/>
                
            </li>
        </ul>
    </div>
    ""

    public void defineUi() {
        ui.Container(uid: "rotator", clocator: [tag: "div", class: "thumbnails"]) {
            List(uid: "tnails", clocator: [tag: "ul"], separator: "li") {
                UrlLink(uid: "{all}", clocator: [:])
            }
        }
    }
}
```

The reason the HTML source in a Groovy file is included is that the "" quote in Groovy is very easy to present complicated HTML source as a String variable. In Java, the user must concatenate each line of the HTML Source to make it a String variable.

The defineUi() defines the UI module for the given HTML source. The major part of the UI module is a List, which uses UI templates to represent a list of links. Tellurium makes it easy and concise to use UI templates to represent UI elements. Based on the ListModule UI module, define a Tellurium JUnit test case as follows:

```
public class ListTestCase extends TelluriumJavaTestCase {
    private static MockHttpServer server;

    @BeforeClass
    public static void setUp(){
        server = new MockHttpServer(8080);
        server.registerHtmlBody("/list.html", ListModule.LIST_BODY);
        server.start();
    }

    @Test
    public void testGetSeparatorAttribute(){
        ListModule lm = new ListModule();
        lm.defineUi();

        connectUrl("http://localhost:8080/list.html");

        attr = (String)lm.getParentAttribute("rotator.tnails[6]", "class");
        assertEquals("thumbnail potd", attr);
    }

    @AfterClass
    public static void tearDown(){
        server.stop();
    }
}
```

Generate Html Source From UI Modules

Very often, some Tellurium users asked us to help them to track problems in their Tellurium test code. Due to some company policy, they cannot provide us the HTML source directly, but the UI module instead. Without the HTML source, there is no way for us to debug their test code because we do not have access to their web applications.

However, if we can do reverse engineering to generate the HTML source from the given UI module, we can use the mock http server [<http://code.google.com/p/aost/wiki/TelluriumMockHttpServer>] to test the generated HTML Source without the need to access their web applications.

Driven by this motivation, we provided the following new method in `DslContext` for users to generate HTML source from UI modules:

```
public String generateHtml(String uid)
```

The `generateHtml(uid)` method is really helpful if you want to help other people to track the problem in their Tellurium test code but you have not access to their web applications and HTML sources. Once the HTML source is generated, you can use the mock http server [<http://code.google.com/p/aost/wiki/TelluriumMockHttpServer>] to test the generated HTML Source].

Implementation

The key is to generate the HTML source for each individual UI object from the composite locator, denoted by `clocator`. As a result, we added two methods to the `CompositeLocator` class:

```
class CompositeLocator {
    String header
    String tag
    String text
    String trailer
    Map<String, String> attributes = [:]
```

```

    public String generateHtml(boolean closeTag){
        .....
    }

    public String generateCloseTag(){
        .....
    }
}

```

where `generateHtml(boolean closeTag)` returns the generated HTML source from the composite locator and the boolean variable *closeTag* indicates whether to generate the closing tag for the HTML source. For Container type UI objects, most likely, you will not generate the closing tag directly, but use the other method `generateCloseTag()` to generate the closing tag separately so that we can include its child elements in between.

Then on the base class `UiObject`, we add the `generateHtml()` method as follows,

```

abstract class UiObject implements Cloneable{
    String uid
    String namespace = null

    def locator

    //reference back to its parent
    def Container parent

    public String generateHtml(){
        if(this.locator != null){
            return getIndent() + this.locator.generateHtml(true) + "\n";
        }

        return "\n";
    }

    public String getIndent(){
        if(parent != null){
            return parent.getIndent() + "    ";
        }else{
            return "";
        }
    }
}

```

To make pretty print, we add a `getIndent()` method in the `UiObject` to get the indentation for the current UI object.

Once we added the `generateHtml()` method, all the concrete UI objects such as `Button`, `InputBox`, and `UrlLink` inherit this method to generate HTML source. However, for a `Container` type, the implementation is different because we need to include its child UI objects in the HTML source. As a result, we overwrite the `generateHtml()` method in the `UiObject`.

```

class Container extends UiObject {
    def components = [:]

    @Override
    public String generateHtml(){
        StringBuffer sb = new StringBuffer(64);
        String indent = getIndent();

        if(this.components.size() > 0){
            if(this.locator != null)
                sb.append(indent + this.locator.generateHtml(false)).append("\n");
            this.components.each {String uid, UiObject obj ->
                sb.append(obj.generateHtml());
            }
        }
    }
}

```

```

        if(this.locator != null)
            sb.append(indent + this.locator.generateCloseTag()).append("\n");
        }else{
            if(this.locator != null){
                sb.append(this.locator.generateHtml(true)).append("\n")
            }
        }
        return sb.toString();
    }
}

```

UI templates in Tellurium objects such as List and Table make things more complicated. The basic idea is to elaborate all UI templates and key is to get the appropriate List size and Table size. We use an algorithm to determine the sizes and we don't want to go over the details here.

Finally, we add the `generateHtml(String uid)` method to the `DslContext` class

```

public String generateHtml(String uid){
    WorkflowContext context = WorkflowContext.getContextByEnvironment(
        this.exploreJQuerySelector, this.exploreSelectorCache)
    def obj = walkToWithException(context, uid)
    return obj.generateHtml()
}

```

Another method `generateHtml()` is used to generate the HTML source for all UI modules defined in a UI module class file.

```

public String generateHtml(){
    StringBuffer sb = new StringBuffer(128)
    ui.registry.each {String key, UiObject val ->
        sb.append(val.generateHtml())
    }
    return sb.toString()
}

```

Usage

We used the following UI module

```

ui.Form(uid: "accountEdit", clocator: [tag: "form", id: "editPage", method: "post"]) {
    InputBox(uid: "accountName", clocator: [tag: "input", type: "text", name: "acc2",
        id: "acc2"])
    InputBox(uid: "accountSite", clocator: [tag: "input", type: "text", name: "acc23",
        id: "acc23"])
    InputBox(uid: "accountRevenue", clocator: [tag: "input", type: "text", name: "acc8",
        id: "acc8"])
    TextBox(uid: "heading", clocator: [tag: "h2", text: "**Account Edit "])
    SubmitButton(uid: "save", clocator: [tag: "input", class: "btn", type: "submit",
        title: "Save", name: "save"])
}

```

Call the `generateHtml()` method

```

generateHtml("accountEdit");

```

and it generates the HTML source as follows,

```
<form id="editPage" method="post">
  <input type="text" name="acc2" id="acc2"/>
  <input type="text" name="acc23" id="acc23"/>
  <input type="text" name="acc8" id="acc8"/>
  <h2>Account Edit </h2>
  <input class="btn" type="submit" title="Save" name="save"/>
</form>
```

Tellurium Powerful Utility: Diagnose

Usually, the main problem that users have in Tellurium is that their UI modules are not defined correctly. As a result, the generated runtime locator is either not unique or cannot be found. Very often, users ask our developers to trace or debug their test code. However, it is a difficult task for our Tellurium developers, too because usually the web application and their full test code are not available to us. It would be more important to provide users some utilities for them to trace/debug their code by themselves instead of relying on our Tellurium developers.

The utility method *diagnose* is designed for this purpose, which is available in the `DslContext` class and the method signature is as follows,

```
public void diagnose(String uid)
```

What it actually does is to dump the following information to console,

1. The number of the matching UI element for the runtime locator corresponding to the *uid*.
2. The html source for the parent UI object of the UI object *uid*.
3. The closest matching UI elements in the DOM for the generated locator.
4. The html source for the entire page.

Most of the above are optional, and thus, Tellurium provides you three more methods for your convenience.

```
public DiagnosisResponse getDiagnosisResult(String uid)
public void diagnose(String uid, DiagnosisOption options)
public DiagnosisResponse getDiagnosisResult(String uid,
    DiagnosisOption options)
```

where `DiagnosisResponse` is defined as

```
public class DiagnosisResponse {
  private String uid;

  private int count;

  private ArrayList<String> matches;
```

```

private ArrayList<String> parents;

private ArrayList<String> closest;

private String html;
}

```

so that you can process the result programmatically. `DiagnosisOption` is used to configure the return result,

```

public class DiagnosisOption {

    boolean retMatch = true;

    boolean retHtml = true;

    boolean retParent = true;

    boolean retClosest = true;
}

```

Implementation

Under the hood, Tellurium core first creates a request for the diagnose call,

```

public class DiagnosisRequest {
    //uid for the UI object
    private String uid;

    //parent UI object's locator
    private String pLocator;

    //UI objects attributes obtaining from the composite locator
    private Map<String, String> attributes;

    //options for the return results
    private boolean retMatch;

    private boolean retHtml;

    private boolean retParent;

    private boolean retClosest;
}

```

The request is then converted into a JSON string so that we can pass the request to Selenium as a custom method,

```

class CustomSelenium extends DefaultSelenium {
    .....

    public String diagnose(String locator, String request){
        String[] arr = {locator, request};
        String st = commandProcessor.doCommand("getDiagnosisResponse", arr);
        return st;
    }
}

```

The custom Selenium server includes our jQuery selector [<http://code.google.com/p/aost/wiki/TelluriumjQuerySelector>] support. We add the following new Selenium method,

```
Selenium.prototype.getDiagnosisResponse = function(locator, req){
.....
}
```

I wouldn't go over the implementation details for this method and you can read the source code on Tellurium Engine project if you are really interested.

Usage

Assume we have the following Tellurium UI module defined

```
public class ProgramModule extends DslContext {

    public static String HTML_BODY = ""
    <div id="ext-gen437" class="x-form-item" tabindex="-1">
        <label class="x-form-item-label" style="width: 125px;" for="ext-comp-1043">
            <a class="help-tip-link" onclick="openTip('Program','program');return false;"
                title="click for more info" href="http://localhost:8080">Program</a>
        </label>

        <div id="x-form-el-ext-comp-1043" class="x-form-element" style="padding-left: 130px;">
            <div id="ext-gen438" class="x-form-field-wrap" style="width: 360px;">
                <input id="programId" type="hidden" name="programId" value="" />
                <input id="ext-comp-1043" class="x-form-text x-form-field x-combo-noedit"
                    type="text" autocomplete="off"
                    size="24" readonly="true" style="width: 343px;" />
                
            </div>
        </div>
        <div class="x-form-clear-left" />
    </div>
    ""

    public void defineUi() {
        ui.Container(uid: "Program", clocator: [tag: "div"], group: "true") {
            Div(uid: "label", clocator: [tag: "a", text: "Program"])
            Container(uid: "triggerBox", clocator: [tag: "div"], group: "true") {
                InputBox(uid: "inputBox", clocator: [tag: "input", type: "text", readonly: "true"],
                    respond: ["click"])
                Image(uid: "trigger", clocator: [tag: "img", src: "**images/s.gif"], respond: ["click"])
            }
        }
    }
}
```

We create a Tellurium test case using the `MockHttpServer` [<http://code.google.com/p/aost/wiki/TelluriumMockHttpServer>] without running an actual web application.

```
public class ProgramModuleTestCase extends TelluriumJavaTestCase{
    private static MockHttpServer server;

    @BeforeClass
    public static void setUp(){
        server = new MockHttpServer(8080);
        server.registerHtmlBody("/program.html", ProgramModule.HTML_BODY);
        server.start();
    }

    @Test
    public void testGetSeparatorAttribute(){
        ProgramModule pm = new ProgramModule();
        pm.defineUi();
        pm.useJQuerySelector();
    }
}
```

```

        connectUrl("http://localhost:8080/program.html");
        pm.diagnose("Program.triggerBox.trigger");
        pm.click("Program.triggerBox.trigger");
    }

    @AfterClass
    public static void tearDown(){
        server.stop();
    }
}

```

Note that we want to diagnose the Image UI object "Program.triggerBox.trigger",

```
pm.diagnose("Program.triggerBox.trigger");
```

Run the test and you will see the return result as follows,

```

Diagnosis Result for Program.triggerBox.trigger
-----

    Matching count: 1

    Match elements:

    --- Element 1 ---



    Parents:

    --- Parent 1---

<div id="x-form-el-ext-comp-1043" class="x-form-element" style="padding-left: 130px;">
  <div id="ext-gen438" class="x-form-field-wrap" style="width: 360px;">
    <input id="programId" name="programId" value="" type="hidden">
    <input id="ext-comp-1043" class="x-form-text x-form-field x-combo-noedit"
      autocomplete="off" size="24" readonly="true" style="width: 343px;"
      type="text">
    
  </div>
</div>

    --- Parent 2---

<div id="ext-gen438" class="x-form-field-wrap" style="width: 360px;">
  <input id="programId" name="programId" value="" type="hidden">
  <input id="ext-comp-1043" class="x-form-text x-form-field x-combo-noedit"
    autocomplete="off" size="24" readonly="true" style="width: 343px;"
    type="text">
  
</div>

    Closest:

    --- closest element 1---


HTML Source:

<head>
  <title>Mock HTTP Server</title>
</head>
<body>
  <div id="ext-gen437" class="x-form-item" tabIndex="-1">

```



```
<label class="x-form-item-label" style="width: 125px;" for="ext-comp-1043">
  <a class="help-tip-link" onclick="openTip('Program','program');return false;"
    title="click for more info" href="http://localhost:8080">Program</a>
</label>

<div id="x-form-el-ext-comp-1043" class="x-form-element" style="padding-left: 130px;">
  <div id="ext-gen438" class="x-form-field-wrap" style="width: 360px;">
    <input id="programId" name="programId" value="" type="hidden">
    <input id="ext-comp-1043" class="x-form-text x-form-field x-combo-noedit"
      autocomplete="off" size="24" readonly="true" style="width: 343px;" type="text">
    
  </div>
</div>
<div class="x-form-clear-left">
</div>
</div>
</body>
```

This is really the happy path and runtime locator is found and is unique. What if the UI module definition is a bit wrong about the Image object?

```
Image(uid: "trigger", clocator: [tag: "img", src: "image/s.gif"], respond:["click"])
```

That is to say, the *src* attribute is not correct.

Run the same test code and the result is as follows,

Diagnosis Result for Program.triggerBox.trigger

```
-----

Matching count: 0

Parents:

--- Parent 1---

<div id="x-form-el-ext-comp-1043" class="x-form-element" style="padding-left: 130px;">
  <div id="ext-gen438" class="x-form-field-wrap" style="width: 360px;">
    <input id="programId" name="programId" value="" type="hidden">
    <input id="ext-comp-1043" class="x-form-text x-form-field x-combo-noedit"
      autocomplete="off" size="24" readonly="true" style="width: 343px;"
      type="text">
    
  </div>
</div>

--- Parent 2---

<div id="ext-gen438" class="x-form-field-wrap" style="width: 360px;">
  <input id="programId" name="programId" value="" type="hidden">
  <input id="ext-comp-1043" class="x-form-text x-form-field x-combo-noedit"
    autocomplete="off" size="24" readonly="true" style="width: 343px;"
    type="text">
  
</div>

Closest:

--- closest element 1---


```

HTML Source:

```
<head>
  <title>Mock HTTP Server</title>
</head>
<body>
  <div id="ext-gen437" class="x-form-item" tabindex="-1">
    <label class="x-form-item-label" style="width: 125px;" for="ext-comp-1043">
      <a class="help-tip-link" onclick="openTip('Program','program');return false;"
        title="click for more info" href="http://localhost:8080">Program</a>
    </label>

    <div id="x-form-el-ext-comp-1043" class="x-form-element" style="padding-left: 130px;">
      <div id="ext-gen438" class="x-form-field-wrap" style="width: 360px;">
        <input id="programId" name="programId" value="" type="hidden">
        <input id="ext-comp-1043" class="x-form-text x-form-field x-combo-noedit"
          autocomplete="off" size="24" readonly="true" style="width: 343px;"
          type="text">
        
      </div>
    </div>
    <div class="x-form-clear-left">
    </div>
  </div>
</body>
```

You can see that there is no matching elements for the runtime locator. But the good thing is that the diagnose method provides you the closest UI elements it can find from the DOM,

```
--- closest element 1---


```

By looking at this above lines, we could realize that the *src* attribute is wrong in our UI module.

Some careful readers may want to ask "why you add a partial matching symbol * to the *src* attribute in the UI module. The reason is that in jQuery, seems the *src* attribute in an Image has to be a full URL such as <http://code.google.com/p/aost/>. One workaround is to put the partial matching symbol * before the URL.

In some case, the return matching count is larger than 1 and you can figure out how to update your UI module definition by looking at all the return elements and their parents.

Internationalization support in Tellurium

Tellurium now provides support for internationalization of strings and exception messages. Any software system should have support for regional language settings and options to be effective. Internationalization and localization provides this support. Locales define the language and region. Locales can define how region specific data is presented to users. Every locale will have a language code followed by a region code. Ex: `fr_FR` represents french language in the region of France. Internationalized strings for each locale is provided through a `MessageBundle` engineered for a specific locale which is of the format `<MessageBundleName>_<language-code>_<country code>.properties`

The Internationalization support in Tellurium is provided through the `InternationalizationManager` class. The default bundle used in Tellurium is the `DefaultMessagesBundle.properties`. All strings and exception messages used in the tellurium core classes are read in from the `DefaultMessageBundle.properties` file.

In order to configure regional messages, This class has a `getMessage` function that provides Internationalization support. This function can also take an optional `Locale` argument to accept function level locale translations.

For plain strings

```
getMessage ( "<key>" )
```

For Strings with parameters

```
getMessage ("<key>" , { [ item1 , item2 , ... , item n] } )
```

For double numeric value

```
getNumber(<doubleValue>)
```

For currency data

```
getCurrency(<doubleValue>)
```

For Dates

```
getDate(<dateValue>)
```

For time

```
getTime(<timeValue>)
```

The `getMessage (<key>)` method signature internationalizes a simple string. The `getMessage (<key> , { [item1 , item2 , ... , item n] })` method definition allows parameterization of an internationalized string to allow external strings/arguments as parameter to the string. This also takes locale as an argument, so we have `getMessage (<key> , locale)` to allow translation of the string to the locale passed in as an argument, provided the key value pair exists in the respective locale property file

The localization can be defined by setting the locale on your system preferences / settings. (ex: regional settings in Windows machine).

Note: By default using `getMessage()` without any locale argument causes the system to use the default locale as defined in ur regional settings. In order to use a locale different from the regional settings, you will have to pass in the locale as an argument to `getMessage`.

Internationalization support has been extended to test cases, so any user defined test case can use

```
getI18nBundle()
```

to utilize the `getMessage` function support in their own test code. Internationalized strings can be added to user defined `MessageBundles` defined in the `src/main/resources` folder of user defined projects. The general steps to provide internationalization in your project are as follows:

1. Create a user defined `MessageBundle.properties`, a default locale message bundle, as well as one for each region you want to provide support for in your project, ex: `MessageBundle_fr_FR.properties` will have strings translated into french.
2. Add the user defined resource bundle using the `getI18nManager` function, like so:
`getI18nBundle().addResourceBundle("MessageBundle")`. This can take an optional locale to add the resource bundle in that locale.
3. Now use the `getMessage` function to internationalize strings

Here is a simple example of code from a `GoogleBooksListGroovyTestCase`. I assume that user has already defined a `MessagesBundle.properties`, located at `src/main/resources`, as follows

`MessagesBundle.properties`

```
GoogleBooksListGroovyTestCase.SetUpModule=Setting up google book list
GoogleBooksListGroovyTestCase.Category=Category is {0}
GoogleBooksListGroovyTestCase.ConnectSeleniumServer=Connection to selenium server
```

Now defining the same properties file in French

`MessageBundle_fr_FR.properties`

```
GoogleBooksListGroovyTestCase.SetUpModule=Liste de livre de google d'établissement
GoogleBooksListGroovyTestCase.Category=La catégorie est {0}
GoogleBooksListGroovyTestCase.ConnectSeleniumServer=Se relier au serveur de sélénium
```

Here is the definition of a `testCase` that uses the Internationalization support

```
class SampleGroovyTestCase extends TelluriumGroovyTestCase {

    public void initUi() {
    }

    public void setUp(){
        setUpForClass()
        //adding the local resource bundle, make sure it's not titled
        //"DefaultMessagesBundle" since this will overwrite the default
        //one we use in Tellurium core and cause exceptions
        getI18nBundle().addResourceBundle("MessagesBundle")

        //getI18nBundle() can also be replaced by
        //IResourceBundle bundle = new ResourceBundle(),

    }

    public void tearDown(){
        tearDownForClass()
    }

    public void testTranslateWithEnglishLocale()
    {
        //translating of strings
    }
}
```

```
String message = getI18nBundle().getMessage("i18nManager.testString")
assertEquals("This is a testString in English", message)

//translation of number data types
Double amount = new Double(345987.246);
String translatedValue = getI18nBundle().getNumber(amount)
assertEquals("345,987.246" , translatedValue)

//translation of currency data types
amount = new Double(9876543.21);
translatedValue = getI18nBundle().getCurrency(amount)
assertEquals("\$9,876,543.21" , translatedValue)

//translation of dates - date is 2009, Jan 1
Date date = new Date(109 , 0 , 1)
translatedValue = getI18nBundle().getDate(date)
assertEquals("Jan 1, 2009" , translatedValue)
    }
}
```

Better Reporting With ReportNG

ReportNG is a simple HTML reporting plug-in for the TestNG framework. It is intended as a replacement for the default TestNG HTML report. The default report is comprehensive but is not so easy to understand at-a-glance. ReportNG provides a simple, colour-coded view of the test results. You can find more information about ReportNG on the following URL. ReportNG [<https://reportng.dev.java.net/>]

TestNG reference project supports generating reports using ReportNG out of the box. To use the reporting plug-in, set the listeners attribute of the testng element in your suite file. ReportNG provides following TestNG listeners.

```
org.uncommons.reportng.HTMLReporter
org.uncommons.reportng.JUnitXMLReporter
```

We are going to use HTMLReporter in the following example. We have a test suite file as shown below.

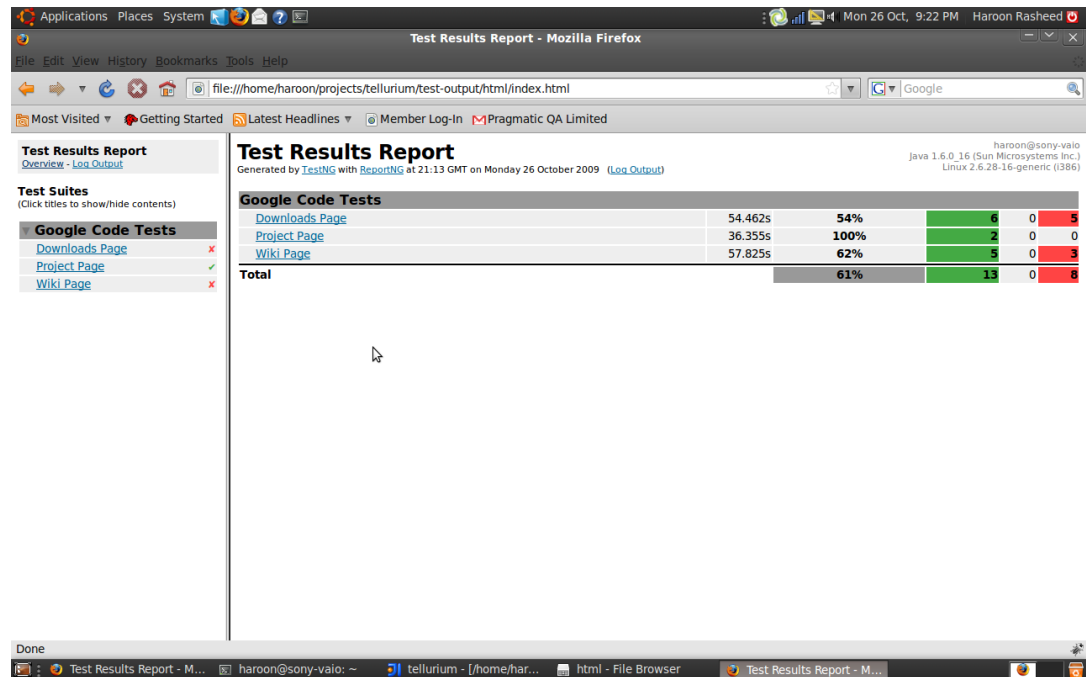
```
<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd" >
<suite name="Google Code Tests">
  <listeners>
    <listener class-name="org.uncommons.reportng.HTMLReporter"/>
  </listeners>

  <test name="Downloads Page">
    <classes>
      <class name="org.tellurium.test.TelluriumDownloadsPageTestNGTestCase"/>
    </classes>
  </test>

  <test name="Project Page">
    <classes>
      <class name="org.tellurium.test.TelluriumProjectPageTestNGTestCase"/>
    </classes>
  </test>

  <test name="Wiki Page">
    <classes>
      <class name="org.tellurium.test.TelluriumWikiPageTestNGTestCase"/>
    </classes>
  </test>
</suite>
```

Now you can run the tests either by Maven, ANT or any IDE using this TestNG suite file. When the test run is complete, you will get a report as shown below.

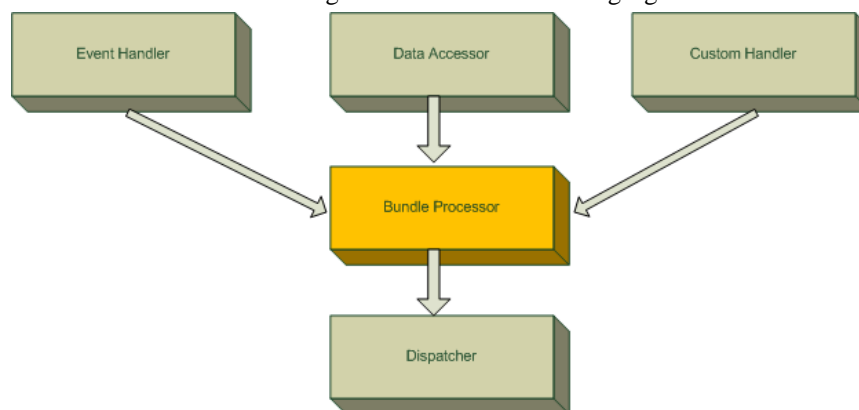


You can see the test report is more readable and comprehensive.

Macro Command

Macro Command is a set of Selenium commands that are bundled together and sent to Selenium Core in one call. This will reduce the round trip latency from Tellurium Core to Engine and thus, improve the speed performance. Another advantage for Macro Command is that Tellurium Engine can reuse the locator because many times the commands in the same bundle act on the same UI element or same sub-tree in the DOM.

To implement Macro Command, we added one more tier to Tellurium Core to automatically handle the Macro Command bundling as shown in the following figure.



To use Macro Command, we add the following settings to configuration file TelluriumConfig.groovy:

```
//the bundling tier
bundle{
    maxMacroCmd = 5
    useMacroCommand = true
}
```

and the following methods to `DslContext` to change the Macro command settings at runtime

```
public void useMacroCmd();
public void disableMacroCmd();
public useMaxMacroCmd(int max);
public int getMaxMacroCmd();
```

If you look at the server log and you will see what happened under the hood as follows.

```
14:57:49.584 INFO - Command request: getBundleResponse[["uid":"ProblematicForm.Username.Input",
  "args":["jquery=table tr input[type=text][name=j]", "t"], "name":"keyDown", "sequ":44},
  {"uid":"ProblematicForm.Username.Input", "args":["jquery=table tr input[type=text][name=j]",
  "t"], "name":"keyPress", "sequ":45}, {"uid":"ProblematicForm.Username.Input", "args":
  ["jquery=table tr input[type=text][name=j]", "t"], "name":"keyUp", "sequ":46}, {"uid":
  "ProblematicForm.Password.Input", "args":["jquery=table tr input[type=password][name=j]",
  "t"], "name":"keyDown", "sequ":47}, {"uid":"ProblematicForm.Password.Input", "args":
  ["jquery=table tr input[type=password][name=j]", "t"], "name":"keyPress", "sequ":48}],
  ] on session 9165cd68806a42fdbdef9f87e804a251
14:57:49.617 INFO - Got result: OK,[] on session 9165cd68806a42fdbdef9f87e804a251
```

In the above example, the command bundle includes the following commands:

```
keyDown "ProblematicForm.Username.Input", "t"
keyPress "ProblematicForm.Username.Input", "t"
keyUp "ProblematicForm.Username.Input", "t"
keyDown "ProblematicForm.Username.Input", "t"
keyPress "ProblematicForm.Username.Input", "t"
```

and they are combined as a single API call to the Tellurium Engine.

Groovy Features Used in Tellurium

In Tellurium, we used many Groovy features. We like to list some of them as follows.

BuildSupport

`BuildSupport` is very powerful for parsing nested definitions, such as XML markup. In Tellurium, the `buildsupport` is used as UI object definition parser and it is one of the bases for our internal DSL.

```
UiDslParser extends BuilderSupport{

    protected void setParent(Object parent, Object child) {
        if(parent instanceof Container){
            parent.add(child)
            child.parent = parent
        }
    }

    protected Object createNode(Object name) {
        def builder = builderRegistry.getBuilder(name)

        if(builder != null){
            def obj = builder.build(null, null)

            return obj
        }
    }
}
```

```

        return null
    }

    //should not come here for Our DSL
    protected Object createNode(Object name, Object value) {

        return null
    }

    protected Object createNode(Object name, Map map) {
        def builder = builderRegistry.getBuilder(name)

        if(builder != null){
            def obj = builder.build(map, null)
            return obj
        }

        return null
    }

    protected Object createNode(Object name, Map map, Object value) {
        def builder = builderRegistry.getBuilder(name)

        if(builder != null){
            def obj = builder.build(map, (Closure)value)

            return obj
        }

        return null
    }

    protected void nodeCompleted(Object parent, Object node) {
        //when the node is completed and its parent is null, it means this
        //node is at the top level
        if(parent == null){
            UiObject uo = (UiObject)node
            //only put the top level nodes into the registry
            registry.put(uo.uid, node)
        }
    }
}

```

Dynamic Scripting

In Groovy, you can define Groovy code as a String and then run them at run time, i.e, code generates code, which makes pure DSL tests possible. Our DslScriptExecutor is a good example and can demonstrate the power of the dynamic scripting.

```

class DslScriptExecutor {

    static void main(String[] args){
        if(args != null && args.length == 1){
            def dsl = new File(args[0]).text
            def script = ""
            import aost.dsl.DslScriptEngine

            class DslTest extends DslScriptEngine{
                def test(){
                    init()
                    ${dsl}
                    shutDown()
                }
            }

            DslTest instance = new DslTest()
            instance.test()
        }
    }
}

```



```
        new GroovyShell().evaluate(script)
    }else{
        println("Usage: DslScriptExecutor dsl_file")
    }
}
}
```

GroovyInterceptable

GroovyInterceptable can intercept all method calls so that you can do some processing before or after the invocation. Or you can delegate the method calls to another class. For example, The Tellurium dispatcher will delegate all method calls it received to Selenium Client as follows.

```
class Dispatcher implements GroovyInterceptable{

    private SeleniumClient sc = new SeleniumClient()

    def invokeMethod(String name, args)
    {
        return sc.client.metaClass.invokeMethod(sc.client, name, args)
    }
}
```

methodMissing

In Groovy, you can use "methodMissing" to intercept and delegate undefined method calls. For example, in DslScriptEngine, the "methodMissing" method will catch and delegate all methods to DslAostSeleneseTestCase except the "init", "openUrl", and "shutDown" methods:

```
class DslScriptEngine extends DslContext{

    protected def methodMissing(String name, args) {
        if(name == "init")
            return init()
        if(name == "openUrl")
            return openUrl(args)
        if(name == "shutDown")
            return shutDown()

        if(DslAostSeleneseTestCase.metaClass.respondsTo(aost, name, args)){
            return aost.invokeMethod(name, args)
        }

        throw new MissingMethodException(name, DslScriptEngine.class, args)
    }
}
```

Singleton

You can Use Groovy MetaClass to define a singleton, which is the right Groovy way to define a singleton. For example, we have the EventHandler class

```
class EventHandler{
}
```

Then, we can define its `MetaClass` as follows.

```
class EventHandlerMetaClass extends MetaClassImpl{
    private final static INSTANCE = new EventHandler()
    EventHandlerMetaClass() { super(EventHandler) }
    def invokeConstructor(Object[] arguments) { return INSTANCE }
}
```

After that, we register the metaClass:

```
def registry = GroovySystem.metaClassRegistry
registry.setMetaClass(EventHandler, new EventHandlerMetaClass())
```

In this way, all

```
EventHandler handler = new EventHandler()
```

will return the same Instance, i.e., it is a singleton. Our Tellurium framework heavily depends on this pattern so that we do not need to use another framework to wiring different object together.

Groovy 1.7.0 supports the `@Singleton` annotation to make the singleton definition much easier. For example, we define the `BundleProcessor` class as follows.

```
@Singleton
public class BundleProcessor implements Configurable {
    ...
}
```

GString

GString can be used to embed variable in a String and the value will be resolved at run time. This makes it very convenient to write XPath template. For example, In the Selector object, we have

```
class Selector extends UiObject {
    def selectByLabel(String target, Closure c){
        c(locator, "label=${target}")
    }
    def selectByValue(String target, Closure c){
        c(locator, "value=${target}")()
    }
}
```

Optional Type

In Groovy you can specify a variable type if you know the type and you want to do Type check at compile time. If you do not care about the type or want the method to be more flexible, you do not

need to define the type and just use "def" to define a variable without specifying its type. For example, in the composite locator class we specified the Type of all variables except the position. Because type of UI component position might be a single type.

```
class CompositeLocator {
    String header
    String tag
    String text
    String trailer
    def position
    Map<String, String> attributes = [:]
}
```

Closure

Closure is also used in Tellurium frequently. For example, in the DslContext class we define the closure as

```
def selectByLabel(String uid, String target){
    WorkflowContext context = WorkflowContext.getDefaultContext()
    ui.walkTo(context, uid)?.selectByLabel(target){ loc, optloc ->
        String locator = locatorMapping(context, loc)
        eventHandler.select(locator, optloc)
    }
}
```

In the Selector object, it will run the closure:

```
class Selector extends UiObject {
    def selectByLabel(String target, Closure c){
        c(locator, "label=${target}")
    }
}
```

i.e, it will pass in UI specific parameters to the closure and run it. What the closure actually does is to create the runtime locator and call the eventhandler to fire the select event to selenium server. Be aware, because the scope of closure, the eventhandler and the WorkflowContext defined in the DslContext class are still available to the closure even the closure is run inside the Selector object. Amazing, right?

Groovy Syntax

In Groovy, the syntax is very expressive, for example, the DslContext class defines a lot of methods such as:

```
def doubleClick(String uid){}
```

and it can be written as

```
doubleClick uid
```

and the above is one of the basic DSLs for Tellurium. The Map definition is also very expressive and useful. Tellurium locators use map to define xpath or UI component attributes. For example:

```
ui.Container(uid: "google_start_page", clocator: [tag: "td", group: "true"]){
  InputBox(uid: "searchbox", clocator: [title: "Google Search"])
  SubmitButton(uid: "googlesearch", clocator: [name: "btnG", value: "Google Search"])
  SubmitButton(uid: "Imfeelinglucky", clocator: [value: "I'm Feeling Lucky"])
}
```

Varargs

Groovy followed a convention if you wanted a method to have varargs: the last argument should be declared as `Object[]`. For example, in `DslContext`, we have the `onWidget` method:

```
def onWidget(String uid, String method, Object[] args){
  .....
}
```

The `onWidget` method can be expressed in DSL syntax as follows.

```
onWidget "DOJO_DatePicker", selectYear, 1992
```

propertyMissing

Groovy also supports `propertyMissing` for dealing with property resolution attempts. For example, in `DslContext`, we define

```
def propertyMissing(String name) {
  println "Warning: property ${name} is missing, treat it as a String. "
  name
}
```

That is to say, for the missing property, we will return it as a `String`. This is very useful for the `"onWidget"` method. Without the `propertyMissing`, we have to use

```
onWidget "DOJO_DatePicker", "selectYear", 1992
```

With the `propertyMissing`, we can use the method name directly:

```
onWidget "DOJO_DatePicker", selectYear, 1992
```

Delegate

Groovy 1.6 provides a set of Java annotations for meta programming, the `@Delegate` is one of them. With the `@Delegate` transformation, a class field or property can be annotated and become an object to which method calls are delegated. Tellurium widget class used this feature.

```
abstract class Widget extends UiObject {  
  
    @Delegate  
    private WidgetDslContext dsl = new WidgetDslContext();  
  
    ...  
}
```

That is to say, all DSL calls will be delegated to the `WidgetDslContext` class.

Grape

Grape is the infrastructure enabling the `grab()` calls in Groovy, a set of classes leveraging Ivy to allow for a repository driven module system for Groovy. Grape will, at runtime, download as needed and link the named libraries and all dependencies forming a transitive closure when the script is run from existing repositories such as Ibiblio, Codehaus, and java.net.

Tellurium `rundsl.groovy` used Grape to download dependencies for DSL scripts.

```
import groovy.grape.Grape;  
  
Grape.grab(group:'org.telluriumsource', module:'tellurium-core', version:'0.7.0', classLoader:this.class.classLoader)  
...  
  
import org.telluriumsource.dsl.DslScriptExecutor  
  
def runDsl(String[] args) {  
    def cli = new CliBuilder(usage: 'rundsl.groovy -[hf] [scriptname]')  
    cli.with {  
        h longOpt: 'help', 'Show usage information'  
        f longOpt: 'scriptname', 'DSL script name'  
    }  
  
    ...  
}  
  
println "Running DSL test script, press Ctrl+C to stop."  
  
runDsl(args)
```

Chapter 8. Tellurium UID Description Language

Introduction

Tellurium UID Description Language (UDL) is the Language to definite the UID field in Tellurium UI objects. In Tellurium, the UI object is referred to by its UID, i.e., the UI object identifier.

For nested UI objects, the UID of the UI Object is a concatenated UI objects' uids along its path to the UI Object.

For example, in the following nested UI Module shown below, the TextBox is referred to as the "parent_ui.child_ui.grand_child.textbox1".

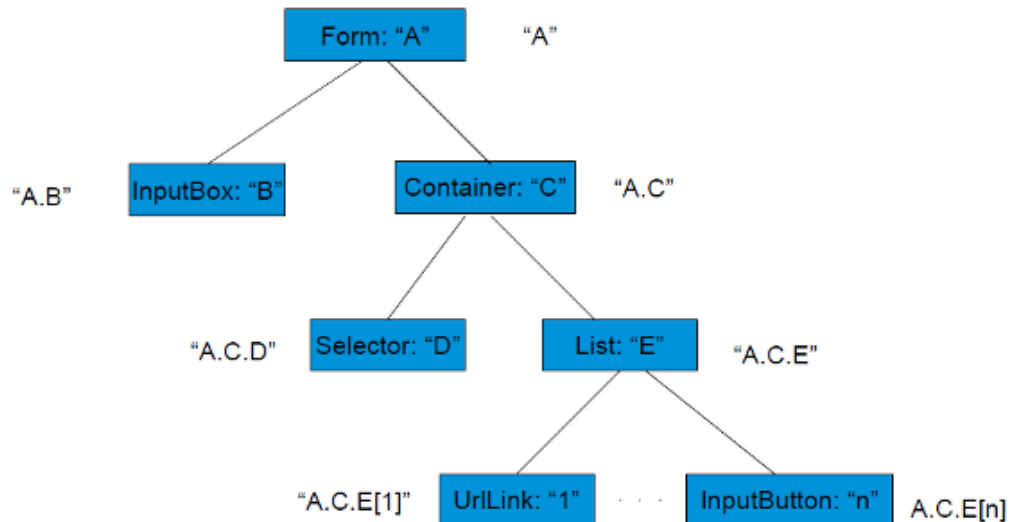
```
ui.Container(uid: "parent_ui"){
  InputBox(uid: "inputbox1", locator: "...")
  Button(uid: "button1", locator: "...")
  Container(uid: "child_ui"){
    Selector(uid: "selector1", locator: "...")
    ...
    Container(uid: "grand_child"){
      TextBox(uid: "textbox1", locator: "...")
      ...
    }
  }
}
```

The exceptions are tables and lists, which use [x][y] or [x] to reference the elements inside. For example, labels_table[2][1] and GoogleBooksList.subcategory[2]. The Table header can be referred in the format of issueResult.header[2].

More general cases are illustrated by the following UI module:

```
ui.Form(uid: "A", clocator: [:]){
  InputBox(uid: "B", clocator: [:])
  Container(uid: "C", clocator: [tag: "div"]){
    Selector(uid: "D", clocator: [:])
    List(uid: "E", clocator: [tag: "ul", separator: "li"]){
      UrlLink(uid: "{1} as Link", clocator: [:])
      InputBox(uid: "{all}", clocator: [:])
    }
  }
}
```

The UID name convention can be illustrated by the following graph.



For example, the UID of the List *E* in the above diagram is *A . C . E* and the *IconButton* in the List *E* is referred by its index *n*. For example: *A.C.En*.

As we said above, UID is used to identify and describe a UI object in Tellurium, but why do we need a language to describe the name of a UI object in Tellurium?

The answer is that UID is not just the name of a UI object, it is also used to describe the dynamic factors in a Tellurium UI template [http://code.google.com/p/aost/wiki/UserGuide070TelluriumBasics#UI_Templates].

Tellurium UI templates have two purposes:

1. When there are many identical UI elements, use one template to represent them all.
2. When there are variable/dynamic sizes of UI elements at runtime, the patterns are known, but not the size.

More specifically, *Table*, *StandardTable*, and *List* are the three Tellurium objects that define UI templates. The *Table* object is a special case of the *StandardTable* object.

1. **Table** and **StandardTable** define two dimensional UI templates.
2. **List** defines one dimensional UI templates.

As a result, the Tellurium UID Description Language (UDL) is designed to

1. address the dynamic factors in Tellurium UI templates
2. increase the flexibility of Tellurium UI templates.

Tellurium UID Description Language (UDL) is implemented with the Antlr 3 parser generator [<http://wwwantlr.org/>]. We will first cover the grammars of UDL and then the implementation details.

Tellurium UID Description Language

We like to introduce the UDL grammars and the use of UDL.

UDL Grammars

The UDL grammars are defined as follows,

```

grammar Udl;

uid
    :      baseUid
    |      listUid
    |      tableUid
    ;

baseUid
    :      ID
    ;

listUid
    :      '{ ' INDEX ' }'
    |      '{ ' INDEX ' }' 'as' ID
    ;

tableUid
    :      tableHeaderUid
    |      tableFooterUid
    |      tableBodyUid
    ;

tableHeaderUid
    :      '{ ' 'header' ':' INDEX ' }'
    |      '{ ' 'header' ':' INDEX ' }' 'as' ID
    ;

tableFooterUid
    :      '{ ' 'footer' ':' INDEX ' }'
    |      '{ ' 'footer' ':' INDEX ' }' 'as' ID
    ;

tableBodyUid
    :      '{ ' 'row' ':' INDEX ',' 'column' ':' INDEX ' }'
    |      '{ ' 'row' ':' INDEX ',' 'column' ':' INDEX ' }' 'as' ID
    |      '{ ' 'row' '->' ID ',' 'column' ':' INDEX ' }'
    |      '{ ' 'row' '->' ID ',' 'column' ':' INDEX ' }' 'as' ID
    |      '{ ' 'row' ':' INDEX ',' 'column' '->' ID ' }'
    |      '{ ' 'row' ':' INDEX ',' 'column' '->' ID ' }' 'as' ID
    |      '{ ' 'row' '->' ID ',' 'column' '->' ID ' }'
    |      '{ ' 'row' '->' ID ',' 'column' '->' ID ' }' 'as' ID
    |      '{ ' 'tbody' ':' INDEX ',' 'row' ':' INDEX ',' 'column' ':' INDEX ' }'
    |      '{ ' 'tbody' ':' INDEX ',' 'row' ':' INDEX ',' 'column' ':' INDEX ' }' 'as' ID
    |      '{ ' 'tbody' ':' INDEX ',' 'row' '->' ID ',' 'column' ':' INDEX ' }'
    |      '{ ' 'tbody' ':' INDEX ',' 'row' '->' ID ',' 'column' ':' INDEX ' }' 'as' ID
    |      '{ ' 'tbody' ':' INDEX ',' 'row' ':' INDEX ',' 'column' '->' ID ' }'
    |      '{ ' 'tbody' ':' INDEX ',' 'row' ':' INDEX ',' 'column' '->' ID ' }' 'as' ID
    |      '{ ' 'tbody' ':' INDEX ',' 'row' '->' ID ',' 'column' '->' ID ' }'
    |      '{ ' 'tbody' ':' INDEX ',' 'row' '->' ID ',' 'column' '->' ID ' }' 'as' ID
    ;

fragment LETTER : ('a'..'z' | 'A'..'Z') ;
fragment DIGIT : '0'..'9';
INDEX : (DIGIT+ | 'all' | 'odd' | 'even' | 'any' | 'first' | 'last' );
ID : LETTER (LETTER | DIGIT | '_' )*;
WS : (' ' | '\t' | '\n' | '\r' | '\f' )+ {$channel = HIDDEN;};

```

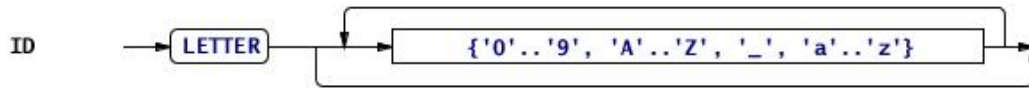
The grammars defined the UIDs for the following three categories of Tellurium UI Objects [<http://code.google.com/p/aost/wiki/UserGuide070UIObjects>].

1. Regular UI objects without UI templates such as Input Box [http://code.google.com/p/aost/wiki/UserGuide070UIObjects#Input_Box] and Container [<http://code.google.com/p/aost/wiki/UserGuide070UIObjects#Container>]
2. List type UI object, i.e., List [<http://code.google.com/p/aost/wiki/UserGuide070UIObjects#List>]
3. Table type UI objects including Table [<http://code.google.com/p/aost/wiki/UserGuide070UIObjects#Table>] and StandardTable [http://code.google.com/p/aost/wiki/UserGuide070UIObjects#Standard_Table].

We like to go over the grammars in details.

ID

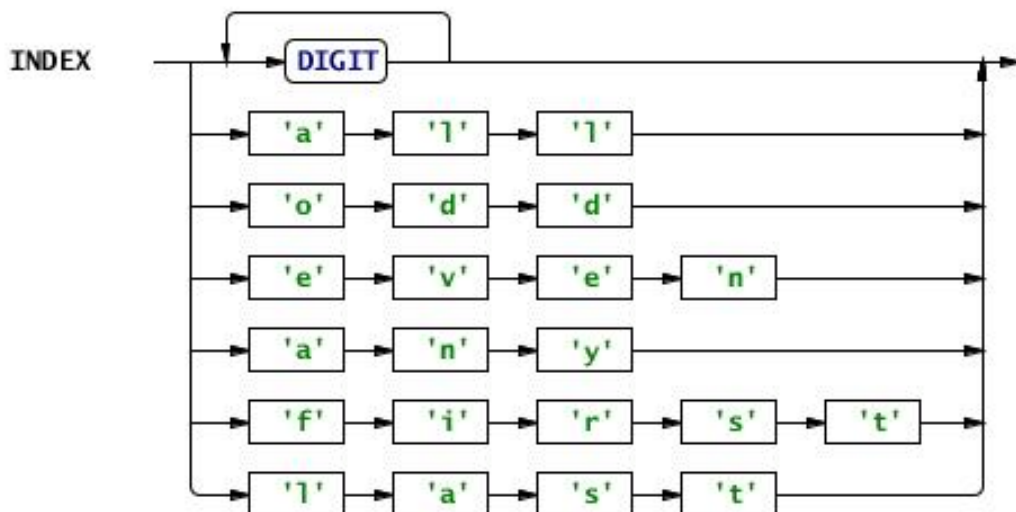
ID is the name of the UI object. The ID in the UDL starts with a letter and is followed by digits, letters, and "_" as follows.



Index

Index defines the position of the UI object. The index in the UDL can be any of the following values:

- Number, for example, "5".
- "first", the first element.
- "last", the last element.
- "any", any position, usually the position is dynamic at runtime.
- "odd", the odd elements.
- "even", the even elements
- "all", wild match if not exact matches found.



Regular UI Objects

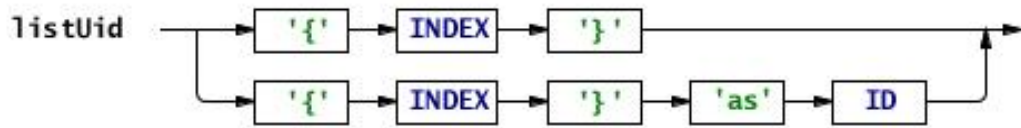
For most Tellurium UI objects [<http://code.google.com/p/aost/wiki/UserGuide070UIObjects>] without UI templates, the UID is an ID, i.e., a name. They don't need any index description since the name and the UI object is one to one mapping.



For example, the UID of the container is "GoogleSearchModule" and "Search" is the name of one SubmitButton in the previous Google search module.

List

The List object [<http://code.google.com/p/aost/wiki/UserGuide070UIObjects#List>] defines an array of UI objects and its UID consists of an index and an optional name.



For example, the following List "A", defines the following Ui Objects:

- InputBox, position at "1" and name "Input".
- Selector, position at "2" and name "Select".
- TextBox, represents the rest objects not at position "1" and "2".

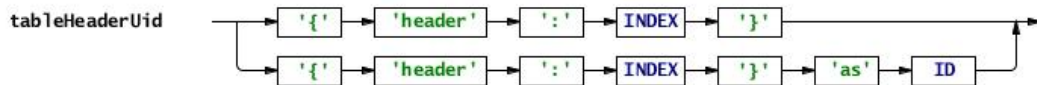
```
ui.List(uid: "A", clocator: [tag: "table"], separator: "tr") {  
  InputBox(uid: "{1} as Input", clocator: [:])  
  Selector(uid: "{2} as Select", clocator: [:])  
  TextBox(uid: "{all}", clocator: [tag: "div"])  
}
```

We can use "A[1]" or "A.Input" to reference the InputBox object. "A[3]" and "A[6]" are mapped to the TextBox object.

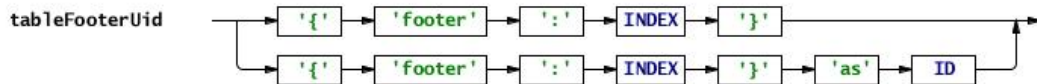
Table

As we said, the Table object [<http://code.google.com/p/aost/wiki/UserGuide070UIObjects#Table>] is a special case of the more general object StandardTable [http://code.google.com/p/aost/wiki/UserGuide070UIObjects#Standard_Table], which includes a header, a footer, and one or multiple body sections.

Table header is very much similar to the List, but its index starts with the "header" indicator.



The footer is similar to header and its index starts with the "footer" indicator.



The body is more complicated since it is represented by the triple [tbody, row, column].

Routing

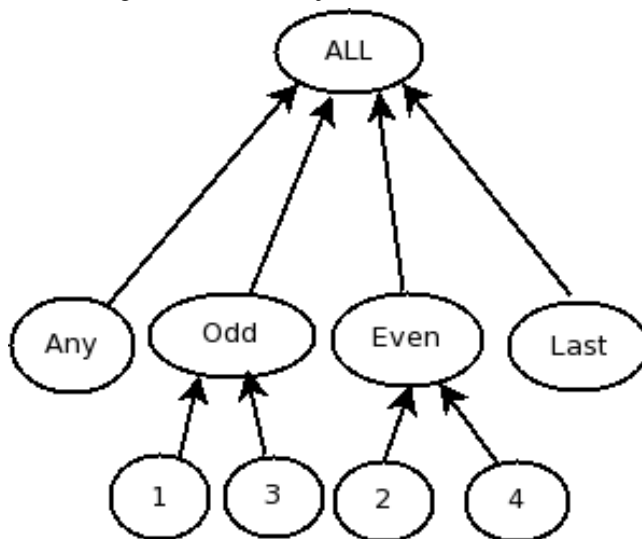
The routing mechanism maps the runtime UID reference to the appropriate UI template. We will cover the routing mechanism for UI templates in the List and Table objects.

RTree

For a List object, the index could be any of the following:

- *digits*, such as "1", "3", and "5".
- *first*, which is converted to "1".
- *last*, the last element.
- *any*, any position.
- *odd*, odd elements.
- *even*, even elements.
- *all*, match all and default UI element.

The routing tree for a List object is called a **RTree** as follows.



The root is the "all" node and the digits, "any", and "last" are leaf nodes. "odd" and "even" nodes are parents of the digit nodes. The routing algorithm always to match the runtime uid to one of the leaf node, if not found, go up to match its parent node until it reaches the "all" node, which presents a default UI object.

For example, we have the following List defined:

```
ui.List(uid: "Example", clocator: [tag:"div"], separator: "p"){
  InputBox(uid: "{first} as Input", clocator: [:])
  Button(uid: "{odd}", clocator: [:])
  Selector(uid: "{4} as Select", clocator: [:])
  SubmitButton(uid: "{last} as Submit", clocator: [:])
  TextBox(uid: "{all}", clocator: [:])
}
```

By the RTree routing algorithm, the runtime uid mappings are shown as follows.

```
//Runtime uid          mapped UI object

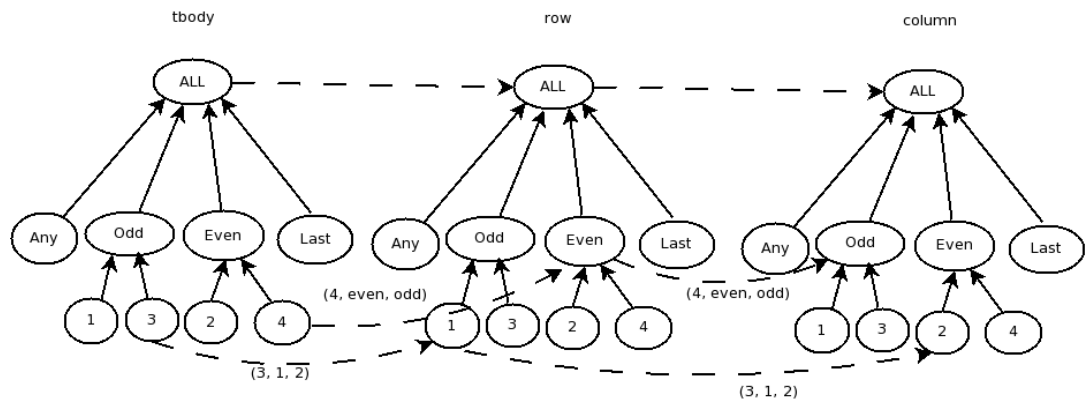
//List Referenced by ID
"Example.Input"  ---> InputBox
"Example.Select" ---> Selector
"Example.Submit" ---> SubmitButton

//List Referenced by Index
"Example[first]" ---> InputBox
"Example[1]"     ---> InputBox
"Example[2]"     ---> TextBox
"Example[3]"     ---> Button
"Example[4]"     ---> Selector
"Example[last]"  ---> SubmitButton
```

RGraph

The Table type of objects usually include one header, one or multiple tbodies, and one footers. That is to say, the Table object is represented by triple[tbody, row, column]. Each tuple is represented by a RTree and the three RTrees form a RGraph. The reason it is called a graph is that each tuple is not separated. If multiple nodes in the RGraph form a UI template, we can draw a dash line to connect together. For example, the node "4" in tbody, the "even" node in row, and the "odd" node in the column form a UI template such as

```
UrlLink(uid: "tbody: 4, row: even, column: odd", clocator: [:])
```



In this way, the three RTrees actually form a graph. As a result, the routing problem becomes "how to find a UI template in the RGraph that is the closest one to the runtime UID?".

To do this, we assign a fitness, i.e., weight, to tbody, row, and column respectively. Usually, we select the weight as follows:

```
tbody > row > column
```

That is to say, we always try to match the tbody first, then the row and the column.

The routing algorithm can be illustrated by the following code snippet.

```
UiObject route(String key) {
//check the ID reference
UiObject object = this.indices.get(key);

//this is a index reference
if(object == null){
//normalize the index
String[] parts= key.replaceFirst('_', '').split('_');
```

```
String[] ids = parts;
if(parts.length < 3){
    ids = ["1", parts].flatten();
}
String x = ids[0];
if("first".equalsIgnoreCase(x)){
    x = "1";
}
String y = ids[1];
if("first".equalsIgnoreCase(y)){
    y = "1";
}
String z = ids[2];
if("first".equalsIgnoreCase(z)){
    z = "1";
}

//Find match nodes separately for tbody, row, and column
String[] list = this.generatePath(x);
Path path = new Path(list);
RNode nx = this.walkTo(this.t, x, path);
list = this.generatePath(y);
path = new Path(list);
RNode ny = this.walkTo(this.r, y, path);
list = this.generatePath(z);
path = new Path(list);
RNode nz = this.walkTo(this.c, z, path);

//Use the fitness to select the closest match
int smallestFitness = 100 * 4;
RNode xp = nx;
while (xp != null) {
    RNode yp = ny;
    while (yp != null) {
        RNode zp = nz;
        while(zp != null){
            //internal representation of the index
            String iid = this.getInternalId(xp.getKey(), yp.getKey(), zp.getKey());
            //If they form a UI template
            if(xp.templates.contains(iid) && yp.templates.contains(iid)
                && zp.templates.contains(iid)){
                int fitness = (nx.getLevel() - xp.getLevel()) * 100 +
                    (ny.getLevel() - yp.getLevel()) * 10 +
                    (nz.getLevel() - zp.getLevel());
                if(fitness < smallestFitness){
                    object = this.templates.get(iid);
                    smallestFitness = fitness;
                }
            }
            //walk up the RGraph
            zp = zp.parent;
        }
        //walk up the RGraph
        yp = yp.parent;
    }
    //walk up the RGraph
    xp = xp.parent;
}

//return the closest match
return object;
}
```

Take the previous Tellurium Issue Result UI module [<http://code.google.com/p/aost/wiki/TelluriumUIDDescriptionLanguage#Example>] as an example, we have the following runtime UID to UI object mapping.

//Runtime UID	mapped UI object
"issueResult[1][Extra]"	-->TextBox
"issueResult[1][ID]"	--> UrlLink
"issueResult[2][Type]"	-->UrlLink

Be aware that the "Extra", "ID", and "Type" are index references to the header columns of the header "Extra", "ID", and "Type" respectively and they will be replaced by the actual column number of the corresponding header at runtime.

Implementation

Antlr 3

UDL exploited Antlr [<http://wwwantlr.org/>] parser generator. To use Antlr 3, we can define the Maven dependencies as follows.

```
<dependency>
  <groupId>org.antlr</groupId>
  <artifactId>antlr</artifactId>
  <version>3.1.3</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.antlr</groupId>
  <artifactId>gunit</artifactId>
  <version>3.1.3</version>
  <scope>test</scope>
</dependency>
```

We also need the Antlr3 Maven Plugin [<http://mojo.codehaus.org/antlr3-maven-plugin/>] to automatically generate the Java parser code from Antlr 3 grammar.

```
<build>
  <resources>
    <resource>
      <directory>src/main/java</directory>
    </resource>
    <resource>
      <directory>src/main/resources</directory>
    </resource>
    <resource>
      <directory>src/main/gen</directory>
    </resource>
  </resources>
  <testResources>
    <testResource>
      <directory>src/test/java</directory>
    </testResource>
    <testResource>
      <directory>src/test/resources</directory>
    </testResource>
  </testResources>

  <plugins>
    <plugin>
      <groupId>org.antlr</groupId>
      <artifactId>antlr3-maven-plugin</artifactId>
      <version>3.1.3-1</version>
      <executions>
        <execution>
          <goals>
            <goal>antlr</goal>
          </goals>
          <configuration>
            <conversionTimeout>10000</conversionTimeout>
            <debug>false</debug>
            <dfa>false</dfa>
            <nfa>false</nfa>
            <libDirectory>src/main/antlr3/imports</libDirectory>
            <messageFormat>antlr</messageFormat>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```
        <outputDirectory>src/main/gen</outputDirectory>
        <printGrammar>false</printGrammar>
        <profile>false</profile>
        <report>false</report>
        <sourceDirectory>src/main/antlr3</sourceDirectory>
        <trace>false</trace>
        <verbose>true</verbose>
    </configuration>
</execution>
</executions>

</plugin>
<plugin>
    <artifactId>maven-clean-plugin</artifactId>
    <configuration>
        <filesets>
            <fileset>
                <directory>src/main/gen</directory>
                <includes>
                    <include>*/*</include>
                </includes>
                <followSymlinks>false</followSymlinks>
            </fileset>
        </filesets>
    </configuration>
</plugin>
</plugins>
</build>
```

ANTLRWorks [<http://www.antlr.org/works/index.html>] is a novel grammar development environment for ANTLR v3 grammars. ANTLRWorks has an IntelliJ IDEA Plugin [<http://plugins.intellij.net/plugin/?id=953>]. There are also other IDE plugins [<http://www.antlr.org/wiki/display/ANTLR3/Integration+with+Development+Environments>] available.

Data Structure

To create the parser, we need couple Java objects to map to the Antlr 3 grammars. For example, we define the following classes.

```
public enum IndexType {
    VAL,
    REF
}

public class Index {
    IndexType type;
    String value;
}

public class MetaData {
    protected String id;
}

public class ListMetaData extends MetaData {
    protected Index index;
}

public class TableHeaderMetaData extends ListMetaData{
}

public class TableFooterMetaData extends ListMetaData {
}

public class TableBodyMetaData extends MetaData {
    protected Index tbody;
    protected Index row;
}
```



```
        protected Index column;
    }
```

Embedded Java Code

To get data from the parser, we can embed Java code inside the Antlr 3 grammars as follows.

```
grammar Udl;

options {
    language = Java;
}

@header {
    package org.telluriumsource.udl;

    import org.telluriumsource.udl.code.IndexType;
    import org.telluriumsource.udl.Index;
    import org.telluriumsource.udl.Metadata;
    import org.telluriumsource.udl.ListMetadata;
    import org.telluriumsource.udl.TableHeaderMetadata;
    import org.telluriumsource.udl.TableFooterMetadata;
    import org.telluriumsource.udl.TableBodyMetadata;
}

@lexer::header {
    package org.telluriumsource.udl;

    import org.telluriumsource.udl.code.IndexType;
    import org.telluriumsource.udl.Index;
    import org.telluriumsource.udl.Metadata;
    import org.telluriumsource.udl.ListMetadata;
    import org.telluriumsource.udl.TableHeaderMetadata;
    import org.telluriumsource.udl.TableFooterMetadata;
    import org.telluriumsource.udl.TableBodyMetadata;
}

@members{
}

uid      returns [Metadata metadata]
:        bu=baseUid {metadata=bu;}
|        lu=listUid {metadata=lu;}
|        tu=tableUid {metadata=tu;}
;

baseUid returns [Metadata metadata]
:        ID {metadata = new Metadata($ID.text);}
;

listUid returns [ListMetadata metadata]
:        '{ ' INDEX ' }' {metadata = new ListMetadata('_', $INDEX.text, $INDEX.text);}
|        '{ ' INDEX ' }' 'as' ID {metadata = new ListMetadata($ID.text, $INDEX.text);}
;

tableUid returns [Metadata metadata]
:        thu=tableHeaderUid {metadata = thu;}
|        tfu=tableFooterUid {metadata = tfu;}
|        tbu=tableBodyUid {metadata = tbu;}
;

tableHeaderUid returns [TableHeaderMetadata metadata]
:        '{ ' 'header' ':' INDEX ' }'
    {metadata = new TableHeaderMetadata('_', $INDEX.text, $INDEX.text);}
|        '{ ' 'header' ':' INDEX ' }' 'as' ID
    {metadata = new TableHeaderMetadata($ID.text, $INDEX.text);}
;

tableFooterUid returns [TableFooterMetadata metadata]
:        '{ ' 'footer' ':' INDEX ' }'
    {metadata = new TableFooterMetadata('_', $INDEX.text, $INDEX.text);}
|        '{ ' 'footer' ':' INDEX ' }' 'as' ID
    {metadata = new TableFooterMetadata($ID.text, $INDEX.text);}
;

tableBodyUid returns [TableBodyMetadata metadata]
:        '{ ' 'row' ':' inx1=INDEX ',' 'column' ':' inx2=INDEX ' }'
    {metadata = new TableBodyMetadata("_1_" + inx1.getText() + '_' + inx2.getText());
    metadata.setTbody(new Index("1"));
    metadata.setRow(new Index(inx1.getText()));
    metadata.setColumn(new Index(inx2.getText()));}
|        '{ ' 'row' ':' inx1=INDEX ',' 'column' ':' inx2=INDEX ' }' 'as' ID
    {metadata = new TableBodyMetadata($ID.text);}
```

Tellurium UID Description Language

```

        metadata.setTbody(new Index("1"));
        metadata.setRow(new Index(inx1.getText()));
        metadata.setColumn(new Index(inx2.getText())); }
    |
    | ' 'row' '->' id1=ID ' 'column' ':' INDEX ' '
    | {metadata = new TableBodyMetaData("_1_" + id1.getText() + '_' + $INDEX.text);
    | metadata.setTbody(new Index("1"));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index($INDEX.text));}
    |
    | ' 'row' '->' id1=ID ' 'column' ':' INDEX ' 'as' id2=ID
    | {metadata = new TableBodyMetaData(id2.getText());
    | metadata.setTbody(new Index("1"));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index($INDEX.text));}
    |
    | ' 'row' ':' INDEX ' 'column' '->' id1=ID ' '
    | {metadata = new TableBodyMetaData("_1_" + $INDEX.text + '_' + id1.getText());
    | metadata.setTbody(new Index("1"));
    | metadata.setRow(new Index($INDEX.text));
    | metadata.setColumn(new Index(IndexType.REF, id1.getText()));}
    |
    | ' 'row' ':' INDEX ' 'column' '->' id1=ID ' 'as' id2=ID
    | {metadata = new TableBodyMetaData(id2.getText());
    | metadata.setTbody(new Index("1"));
    | metadata.setRow(new Index($INDEX.text));
    | metadata.setColumn(new Index(IndexType.REF, id1.getText()));}
    |
    | ' 'row' '->' id1=ID ' 'column' '->' id2=ID ' '
    | {metadata = new TableBodyMetaData("_1_" + id1.getText() + '_' + id2.getText());
    | metadata.setTbody(new Index("1"));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index(IndexType.REF, id2.getText()));}
    |
    | ' 'row' '->' id1=ID ' 'column' '->' id2=ID ' 'as' id3=ID
    | {metadata = new TableBodyMetaData(id3.getText());
    | metadata.setTbody(new Index("1"));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index(IndexType.REF, id2.getText()));}
    |
    | ' 'tbody' ':' inx1=INDEX ' 'row' ':' inx2=INDEX ' 'column' ':' inx3=INDEX ' '
    | {metadata = new TableBodyMetaData('_' + inx1.getText() + '_' +
    | inx2.getText() + '_' + inx3.getText());
    | metadata.setTbody(new Index(inx1.getText()));
    | metadata.setRow(new Index(inx2.getText()));
    | metadata.setColumn(new Index(inx3.getText()));}
    |
    | ' 'tbody' ':' inx1=INDEX ' 'row' ':' inx2=INDEX ' 'column' ':' inx3=INDEX ' 'as' ID
    | {metadata = new TableBodyMetaData($ID.text);
    | metadata.setTbody(new Index(inx1.getText()));
    | metadata.setRow(new Index(inx2.getText()));
    | metadata.setColumn(new Index(inx3.getText()));}
    |
    | ' 'tbody' ':' inx1=INDEX ' 'row' '->' id1=ID ' 'column' ':' inx2=INDEX ' '
    | {metadata = new TableBodyMetaData('_' + inx1.getText() + '_' + id1.getText() + '_' + inx2.getText());
    | metadata.setTbody(new Index(inx1.getText()));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index(inx2.getText()));}
    |
    | ' 'tbody' ':' inx1=INDEX ' 'row' '->' id1=ID ' 'column' ':' inx2=INDEX ' 'as' id2=ID
    | {metadata = new TableBodyMetaData(id2.getText());
    | metadata.setTbody(new Index(inx1.getText()));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index(inx2.getText()));}
    |
    | ' 'tbody' ':' inx1=INDEX ' 'row' ':' inx2=INDEX ' 'column' '->' id1=ID ' '
    | {metadata = new TableBodyMetaData('_' + inx1.getText() + '_' + inx2.getText() + '_' + id1.getText());
    | metadata.setTbody(new Index(inx1.getText()));
    | metadata.setRow(new Index(inx2.getText()));
    | metadata.setColumn(new Index(IndexType.REF, id1.getText()));}
    |
    | ' 'tbody' ':' inx1=INDEX ' 'row' ':' inx2=INDEX ' 'column' '->' id1=ID ' 'as' id2=ID
    | {metadata = new TableBodyMetaData(id2.getText());
    | metadata.setTbody(new Index(inx1.getText()));
    | metadata.setRow(new Index(inx2.getText()));
    | metadata.setColumn(new Index(IndexType.REF, id1.getText()));}
    |
    | ' 'tbody' ':' INDEX ' 'row' '->' id1=ID ' 'column' '->' id2=ID ' '
    | {metadata = new TableBodyMetaData('_' + $INDEX.text + '_' + id1.getText() + '_' + id2.getText());
    | metadata.setTbody(new Index($INDEX.text));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index(IndexType.REF, id2.getText()));}
    |
    | ' 'tbody' ':' INDEX ' 'row' '->' id1=ID ' 'column' '->' id2=ID ' 'as' id3=ID
    | {metadata = new TableBodyMetaData(id3.getText());
    | metadata.setTbody(new Index($INDEX.text));
    | metadata.setRow(new Index(IndexType.REF, id1.getText()));
    | metadata.setColumn(new Index(IndexType.REF, id2.getText()));}
    |
    ;

fragment LETTER : ('a'..'z' | 'A'..'Z') ;
fragment DIGIT : ('0'..'9') ;
INDEX : (DIGIT+ | 'all' | 'odd' | 'even' | 'any' | 'first' | 'last') ;
ID : LETTER (LETTER | DIGIT | '_' ) * ;
WS : (' ' | '\t' | '\n' | '\r' | '\f' ) + { $channel = HIDDEN ; } ;

```

Parser

The above Antlr 3 Maven plugin generates a lexer, `UdlLexer`, and a parser, `UdlParser`. To wrap things up, we define a parser class as follows.

```
public class UidParser {

    public static Metadata parse(String uid) throws RecognitionException {
        if (uid == null || uid.trim().length() == 0)
            return null;

        CharStream stream = new ANTLRStringStream(uid);
        UdlLexer lexer = new UdlLexer(stream);
        TokenStream tokenStream = new CommonTokenStream(lexer);
        UdlParser parser = new UdlParser(tokenStream);
        return parser.uid();
    }
}
```

Unit Test

To test the generated parser, `UidParser`, we can use JUnit 4 to test it.

```
public class UidParser_UT {

    @Test
    public void testBaseUid(){
        try{
            Metadata data = UidParser.parse("Tellurium");
            assertNotNull(data);
            assertEquals("Tellurium", data.getId());
        }catch(RecognitionException e){
            fail(e.getMessage());
        }
    }

    @Test
    public void testTableHeaderUid(){
        try{
            Metadata data = UidParser.parse("{header: 3} as A");
            assertNotNull(data);
            assertTrue(data instanceof TableHeaderMetadata);
            TableHeaderMetadata th = (TableHeaderMetadata)data;
            assertEquals("A", th.getId());
            assertEquals("3", th.getIndex().getValue());
            assertEquals(IndexType.VAL, th.getIndex().getType());
        }catch(RecognitionException e){
            fail(e.getMessage());
        }
    }

    @Test
    public void testTableBodyRefUid(){
        try{
            Metadata data =
                UidParser.parse("{tbody : 1, row -> good, column -> bad} as Search");
            assertNotNull(data);
            assertEquals("Search", data.getId());
            assertTrue(data instanceof TableBodyMetadata);
            TableBodyMetadata tbmd = (TableBodyMetadata)data;
            assertEquals("1", tbmd.getTbody().getValue());
            assertEquals(IndexType.VAL, tbmd.getTbody().getType());
            assertEquals("good", tbmd.getRow().getValue());
            assertEquals(IndexType.REF, tbmd.getRow().getType());
            assertEquals("bad", tbmd.getColumn().getValue());
            assertEquals(IndexType.REF, tbmd.getColumn().getType());
        }
    }
}
```

```
        } catch (RecognitionException e) {  
            fail(e.getMessage());  
        }  
    }  
}
```

Chapter 9. Tellurium Widgets

Introduction

Tellurium team is refactoring the trunk code base into multiple sub-projects. One purpose is to separate tellurium core framework from its extensions. Tellurium widget is a good example of tellurium extensions. Tellurium provides you the capability to composite UI objects into a widget object and then you can use the widget directly just like using a tellurium UI object. The advantage is that you do not need to deal with the UI at the link or button level for the widget, you just work on the high level methods. Another advantage is that this widget is reusable.

Widget Implementation

Usually, Java script frameworks provide a lot of widgets. Take the Dojo framework as an example. We use the widget DatePicker? to prototype the tellurium widget. The steps to define and use the widget are listed as follows,

- **Extend the widget object and define the DojoWidget? object with its name space as "DOJO"**

For widgets, it is important to include name space to avoid name collision between different widget modules. For example, what is Dojo and ExtJs? both have the widget Date Picker? After add the name space, the widget will be like "DOJO_DatePicker".

- **Define widget by combining tellurium UI Objects**

For example, the DatePicker? widget is defined as follows,

```
class DatePicker extends DojoWidget{

    public void defineWidget() {
        ui.Container(uid: "DatePicker", locator: "/div[@class='datePickerContainer' and
            child::table[@class='calendarContainer']]"){
            Container(uid: "Title", locator: "/table[@class='calendarContainer']
                /thead/tr/td[@class='monthWrapper']/table[@class='monthContainer']
                /tbody/tr/td[@class='monthLabelContainer']"){
                Icon(uid: "increaseWeek", locator: "/span[@dojoattachpoint='increaseWeekNode']")
                Icon(uid: "increaseMonth", locator: "/span[@dojoattachpoint='increaseMonthNode']")
                Icon(uid: "decreaseWeek", locator: "/span[@dojoattachpoint='decreaseWeekNode']")
                Icon(uid: "decreaseMonth", locator: "/span[@dojoattachpoint='decreaseMonthNode']")
                TextBox(uid: "monthLabel", locator: "/span[@dojoattachpoint='monthLabelNode']")
            }
            StandardTable(uid: "calendar", locator: "/table[@class='calendarContainer']/tbody
                /tr/td/table[@class='calendarBodyContainer']"){
                TextBox(uid: "header: all", locator: "")
                ClickableUi(uid: "all", locator: "")
            }
            Container(uid: "year", locator: "/table[@class='calendarContainer']/tfoot/tr
                /td/table[@class='yearContainer']/tbody/tr/td/h3[@class='yearLabel']"){
                Span(uid: "prevYear", locator: "/span[@class='previousYear' and
                    @dojoattachpoint='previousYearLabelNode']")
                TextBox(uid: "currentYear", locator: "/span[@class='selectedYear' and
                    @dojoattachpoint='currentYearLabelNode']")
                Span(uid: "nextYear", locator: "/span[@class='nextYear' and
                    @dojoattachpoint='nextYearLabelNode']")
            }
        }
    }
}
```

Since widget is a fixed block of UIs, it is fine to use the XPath directly instead of the composite locator.

- **Define widget builder**

For example, we defined the following methods:

```
public String getFullYear(){
    return getText("DatePicker.year.currentYear")
}

public void selectPrevYear(){
    click "DatePicker.year.prevYear"
}
```

- **Define widget builder**

The widget is treated as a tellurium UI object and the builder is the same as regular tellurium objects

```
class DatePickerBuilder extends UiObjectBuilder{

    public build(Map map, Closure c) {
        //add default parameters so that the builder can use them if not specified
        def df = [:]
        DatePicker datePicker = this.internBuild(new DatePicker(), map, df)
        datePicker.defineWidget()

        return datePicker
    }
}
```

- **Hook the widget into the Tellurium Core framework**

Each widget module will be compiled as a separate jar file and it should define a bootstrap class to register all the widgets inside the module. By default, the full class name of the bootstrap class is `org.tellurium.widget.XXXX.Init`, where the class `Init` should implement the `WidgetBootstrap` interface to register widgets and `XXXX` stands for the widget module name. It is `DOJO` in our case

```
class Init implements WidgetBootstrap{
    public void loadWidget(UiObjectBuilderRegistry uiObjectBuilderRegistry) {
        if(uiObjectBuilderRegistry != null){
            uiObjectBuilderRegistry.registerBuilder(getFullName("DatePicker"),
                new DatePickerBuilder())
        }
    }
}
```

Then in the tellurium configuration file `TelluriumConfig.groovy`, you should include your module name there,

```
widget{
    module{
        //define your widget modules here, for example Dojo or ExtJs
        included="dojo"
    }
}
```

If you use your own package name for the bootstrap class, for example, `com.mycompany.widget.Boot`, then you should specify the full name there like

```
widget{
    module{
        //define your widget modules here, for example Dojo or ExtJs
        included="com.mycompany.widget.Boot"
    }
}
```

Note, you can load multiple widget modules into the Tellurium Core framework by defining

```
included="dojo, com.mycompay.widget.Boot"
```

- **Use widgets**

The Widget is defined as a regular tellurium UI object. For example,

```
class DatePickerDemo extends DslContext{

    public void defineUi() {
        ui.Form(uid: "dropdown", clocator: [], group: "true"){
            TextBox(uid: "label", clocator: [tag: "h4", text: "Dropdown:"])
            InputBox(uid: "input", clocator: [dojoattachpoint: "valueInputNode"])
            Image(uid: "selectDate", clocator: [title: "select a date",
                dojoattachpoint: "containerDropdownNode", alt: "date"])
            DOJO_DatePicker(uid: "datePicker", clocator: [tag: "div",
                dojoattachpoint: "subWidgetContainerNode"])
        }
    }
}
```

Then on the module file DatePickerDemo?, you can call the widget methods instead of dealing with low level links, buttons, and so on. To make the testing more expressive, Tellurium provides an onWidget method.

```
onWidget(String uid, String method, Object[] args)
```

In that way, we can call the widget methods as follows:

```
onWidget "dropdown.datePicker", selectPrevYear
```

This prototype is on the tellurium trunk/extensions/dojo-widget project. Please see the project for details and try out by yourself. Your feedback is always welcome.

Tellurium Widget Archetype

To create a Tellurium UI widget project, we can use Tellurium Widget archetype as follows,

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id
-DarchetypeArtifactId=tellurium-widget-archetype
-DarchetypeGroupId=org.telluriumsource -DarchetypeVersion=0.7.0-SNAPSHOT
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

Chapter 10. Tellurium Engine

Tellurium 0.7.0 include a new Engine embedded in Selenium Core. The main functionalities of the Tellurium Engine include

- CSS Selector support based on jQuery
- UI module group locating
- UI module Caching
- New APIs based on jQuery

Code Structure

The following are Javascript files in the Engine project:

```
[jfang@Mars engine]$ tree src/main/resources/core/scripts/  
src/main/resources/core/scripts/  
|-- firebuglite  
|   |-- errorIcon.png  
|   |-- firebug-lite.css  
|   |-- firebug-lite.js  
|   |-- firebug.gif  
|   |-- firebug_logo.png  
|   |-- infoIcon.png  
|   |-- progress.gif  
|   |-- spacer.gif  
|   |-- tree_close.gif  
|   |-- tree_open.gif  
|   |-- warningIcon.png  
|-- htmlutils.js  
|-- jquery-1.4.2.js  
|-- jquery-cookies-2.1.0.js  
|-- jquery-simpleteip-1.3.1.js  
|-- json2.js  
|-- log4js.js  
|-- selenium-api.js  
|-- selenium-browserbot.js  
|-- selenium-browserdetect.js  
|-- selenium-commandhandlers.js  
|-- selenium-executionloop.js  
|-- selenium-logging.js  
|-- selenium-remoterunner.js  
|-- selenium-testrunner.js  
|-- selenium-version.js  
|-- tellurium-api.js  
|-- tellurium-cache.js  
|-- tellurium-extensions.js  
|-- tellurium-logging.js  
|-- tellurium-selector.js  
|-- tellurium-udl.js  
|-- tellurium-uialg.js  
|-- tellurium-uibasic.js  
|-- tellurium-uiextra.js  
|-- tellurium-uimodule.js  
|-- tellurium-uiobj.js  
|-- tellurium-uisnapshot.js  
|-- tellurium.js  
|-- tooltip  
|   |-- simpleteip.css  
|-- user-extensions.js  
|-- utils.js  
|-- xmlextras.js
```

where

- jquery-1.4.2.js: jQuery is updated to the latest version 1.4.2.
- jquery-cookies-2.1.0.js: jQuery Cookies Plugin to support more cookie related operation
- tellurium.js: Entry point for Tellurium Engine code and it defined the `Tellurium` function.
- tellurium-selector.js: CSS selector builder
- tellurium-udl.js: Tellurium UDL processing
- tellurium-uialg.js: Tellurium UI algorithm
- tellurium-uibasic.js: Tellurium UI basic
- tellurium-uiextra.js: Tellurium extra UI functionalities
- tellurium-uimodule.js: Tellurium UI module definition on Engine side
- tellurium-uiobj.js: Tellurium UI object
- tellurium-uisnapshot.js: Tellurium UI snapshot
- tellurium-cache.js: Tellurium Engine caching for UI modules and locators
- tellurium-extension.js: Extra Tellurium APIs for Selenium
- tellurium-api.js: New Tellurium APIs based on jQuery
- utils.js: Utility functions

CSS Selector Support

Started from version 0.6.0, Tellurium supports a CSS selector to address the problem of poor performance of xpath in Internet Explorer. Auto-generating jQuery instead of xpath has the following advantages:

- Faster performance in IE
- The power of CSS selector to call methods on jQuery collections to retrieve bulk data
- New CSS selector based Engine to replace Selenium Core

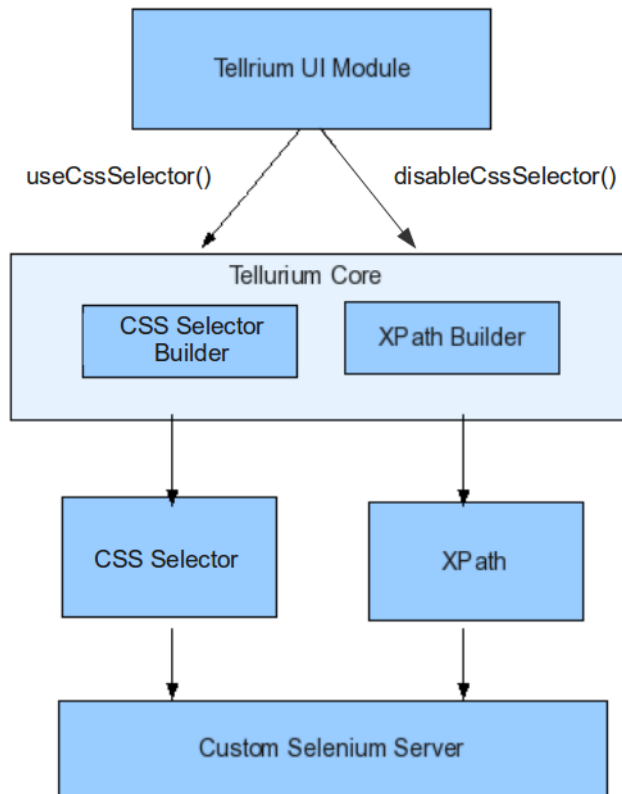
Tellurium Core automatically builds runtime xpath or CSS selector based on a flag in `DslContext`. Tellurium Core uses CSS selector as the default locator. To switch back to XPath from CSS selector, you should call

```
disableCssSelector()
```

and use

```
enableCssSelector()
```

to go back to CSS selector as shown in the following diagram.



Be aware that CSS selector only works for composite locator, i.e., *locator*. If you have base locator, which is pre-generated locator, then the CSS selector will not work for you.

How does the CSS Selector Work? The basic idea is to customize Selenium Core to load the jQuery library at startup time. In other words, we add `jquery.js` in to the `TestRunner.html` and `RemoteRunner.html`.

After that, we register a custom locate strategy "jquery" in Selenium Core. This is done by adding the following lines to the method `BrowserBot.prototype._registerAllLocatorFunctions` in the `selenium-browserbot.js` file. Note that the locate strategy "uimcal" is used by Tellurium Engine internally.

```

this.locationStrategies['jquery'] = function(locator, inDocument, inWindow) {
    return tellurium.locateElementByCSSSelector(locator, inDocument, inWindow);
};

//used internally by Tellurium Engine
this.locationStrategies['uimcal'] = function(locator, inDocument, inWindow) {
    return tellurium.locateElementWithCacheAware(locator, inDocument, inWindow);
};
  
```

This defines new functions for Selenium to locate elements on the page. For example, for the strategy "jquery", if someone runs `click("jquery=div#myid")`, Selenium Core will find the element by CSS selector `div#myid`. Selenium passed three arguments to the location strategy 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.

The actual implementation can be illustrated by the `locateElementByCSSSelector` method.

```
Tellurium.prototype.locateElementByCSSSelector =
    function(locator, inDocument, inWindow){
        var loc = locator;
        var attr = null;
        var isattr = false;
        //check attribute locator
        var inx = locator.lastIndexOf('@');
        if (inx != -1) {
            loc = locator.substring(0, inx);
            attr = locator.substring(inx + 1);
            isattr = true;
        }
        //find element by jQuery CSS selector
        var found = tejQuery(inDocument).find(loc);
        if (found.length == 1) {
            if (isattr) {
                return found[0].getAttributeNode(attr);
            } else {
                return found[0];
            }
        } else if (found.length > 1) {
            if (isattr) {
                return found.get().getAttributeNode(attr);
            } else {
                return found.get();
            }
        } else {
            return null;
        }
    };
};
```

The code is pretty straightforward. When we find one element, return its DOM reference (Note: Selenium does not accept returning an array with only one element) and if we find multiple elements, we use jQuery `get()` method to return an array of DOM references. Otherwise, return null.

As shown in the code, we use the same format of attribute locator as the XPath one, i.e.,

`locator@attribute`

With the adoption of jQuery, we also need some custom jQuery selectors and plugins to meet our needs.

To design jQuery custom selectors, we need to understand the jQuery selector syntax:

```
$.expr[':'].selector_name = function(obj, index, meta, stack){
    .....
}
```

where

- *obj*: a current DOM element
- *index*: the current loop index in stack
- *meta*: meta data about your selector
- *stack*: stack of all elements to loop

The above function returns true to include current element and returns false to exclude current element. A more detailed explanation could be found from jQuery Custom Selectors with Parameters [<http://jquery-howto.blogspot.com/2009/06/jquery-custom-selectors-with-parameters.html>].

To avoid conflicts with user's jQuery library, we yield the "\$" symbol and rename jQuery to tejQuery in Tellurium.

We defined the following Custom jQuery Selectors.

:te_text

The `:te_text` selector is created to select a UI element whose text attribute is a given string. The implementation is simple,

```
tejQuery.extend(tejQuery.expr[':'], {
  te_text: function(a, i, m) {
    return tejQuery.trim(tejQuery(a).text()) === tejQuery.trim(m[3]);
  }
});
```

You may wonder why we use `m[3]` here, the variable `m` includes the following parameters

- `m[0]` : `te_text(argument)` full selector
- `m[1]` : `te_text` selector name
- `m[2]` : ' ' quotes used
- `m[3]` : argument parameters

As a result, the selector picks up the elements whose text attribute, obtained by `text()`, is equal to the passed in parameter `m[3]`.

:group

The `:group` selector is used to implement the group locating [http://code.google.com/p/aost/wiki/UserGuide#Group_Locating] in Tellurium. For example, we want to select a "div" whose children include one "input", one "img", and one "span" tags. How to express this using jQuery?

One way is to use the following selector,

```
tejQuery.expr[':'].group = function(obj){
  var $this = tejQuery(obj);
  return ($this.find("input").length > 0) && ($this.find("img").length > 0)
    && ($this.find("span").length > 0);
};
```

That is to say, only a DOM node satisfying all the three conditions, i.e, whose children include "input", "img", and "span", is selected because the AND conditions. Remember, only the node that returns true for the above function is selected.

However, in real world, we may have many conditions and we cannot use this hard-coded style selector and we need to use the custom selector with parameters instead. Here is our implementation,

```
tejQuery.expr[':'].group = function(obj, index, m){
```

```

var $this = tejQuery(obj);

var splitted = m[3].split(",");
var result = true;

for(var i=0; i<splitted.length; i++){
    result = result && ($this.find(splitted[i]).length > 0);
}

return result;
};

```

If we use firebug to debug the code by running the following jQuery selector

```
tejQuery("div:group(input, img, span)")
```

We can see the variable *m* includes the following parameters

- *m[0]* : `group(input, img, span)` full selector
- *m[1]* : group selector name
- *m[2]* : `' '` quotes used
- *m[3]* : `input, img, span` parameters

:styles

We can see the variable *m* includes the following parameters

```

ui.Container(uid: "Program", clocator: [tag: "div"], group: "true") {
    Div(uid: "label", clocator: [tag: "a", text: "Program"])
    Container(uid: "triggerBox", clocator: [tag: "div"], group: "true") {
        InputBox(uid: "inputBox", clocator: [tag: "input", type: "text",
            readonly: "true", style: "width: 343px;"], respond: ["click"])
        Image(uid: "trigger", clocator: [tag: "img", style: "overflow: auto;
            width: 356px; height: 100px;"], respond: ["click"])
    }
}

```

Unfortunately, the following generated jQuery selector does not work.

```

$('div:has(input[type=text][readonly=true][style="width: 343px;"],
img[style="overflow: auto; width: 356px;height: 100px;"])
img[style="overflow: auto; width: 356px; height: 100px;"]')

```

We have to use a custom jQuery selector to handle the style attribute as follows,

```

tejQuery.expr[':'].styles = function(obj, index, m){
    var $this = tejQuery(obj);

    var splitted = new Array();
    var fs = m[3].split(/:|/);
    for(var i=0; i<fs.length; i++){
        var trimmed = tejQuery.trim(fs[i]);
        if(trimmed.length > 0){

```

```
        splitted.push(trimmed);
    }
}

var result = true;

var l=0;
while(l < splitted.length){
    result = result &&
        (tejQuery.trim($this.css(splitted[l])) == splitted[l+1]);
    l=l+2;
}

return result;
};
```

The main idea is to split the content of the style attribute into multiple single-css classes, then try to match each css class one by one. This approach may not be the optimal one, but it works.

Then, the new runtime jQuery selector becomes,

```
div:group(a:te_text(Program), div) div:group(input:styles(width: 343px;
[type=text][readonly=true], img:styles(overflow: auto; width: 356px;
height: 100px;)) img:styles(overflow: auto; width: 356px; height: 100px;)
```

:nextToLast

One implemented suggested by Kevin is shown as follows,

```
tejQuery.expr[':'].nextToLast = function(obj, index, m){
    var $this = tejQuery(obj);

    if ($this.index() == $this.siblings().length - 1) {
        return true;
    } else {
        return false;
    }
};
```

and he also suggested a more efficient implementation [<http://www.tentonaxe.com/2010/03/custom-jquery-selectors.html>].

```
// this is a selector called nextToLast. its sole purpose is to
// return the next to last element of the array of elements supplied
// to it. The parameters in the function below are as follows;
//
// obj => the current node being checked
// ind => the index of obj in the array of objects being checked
// prop => the properties passed in with the expression
// node => the array of nodes being checked
tejQuery.expr[':'].nextToLast = function(obj, ind, prop, node){

    // if ind is 2 less than the length of the array of nodes, keep it
    if (ind == node.length-2) {
        return true;
    } else {
        // else, remove the node
        return false;
    }
};
```

We also have the following custom jQuery plugin.

outerHTML

When we worked on the diagnose utility [<http://code.google.com/p/aost/wiki/TelluriumPowerUtilityDiagnose>], we were frustrated because we need to get the HTML source of a DOM node, but the `html()` method in jQuery only returns innerHTML. We posted a question to jQuery group [<http://groups.google.com/group/jquery-en>] and got the answer,

```
$('<div>').append( $(jQuery_Selector).clone() ).html()
```

and as suggested by another person, we went further to implement this as a simple jQuery plugin,

```
tejQuery.fn.outerHTML = function() {  
    return tejQuery("<div/>").append( tejQuery(this[0]).clone() ).html();  
};
```

We made two changes here.

1. *outerHTML* is defined as a new property of `jQuery.fn` rather than as a standalone function. This registers the function as a plug-in method.
2. We use the keyword *this* as a replacement for the jQuery selector. Within a plug-in method, *this* refers to the jQuery object that is being acted upon.

UI Module Group Locating

UI Module is the heart of Tellurium Automated Testing Framework. Even UI Module was introduced at the prototype phase, but there was really no algorithm to locate the UI module as a whole. Up to Tellurium 0.6.0, we still need Tellurium core to generate runtime locators based on the UI module definition and then pass Selenium commands to the Selenium core to locate each individual UI element.

The Santa algorithm is the missing half of the Tellurium UI module concept. The algorithm can locate the whole UI module at the runtime DOM. After that, you can just pass in UI element's UID to find it in the cached UI module on Tellurium Engine. That is to say, you don't need Tellurium Core to generate the runtime locators any more. For compatibility reason, Tellurium Core still generates runtime locators, but they are not really needed if you turn on UI module group locating and caching by calling

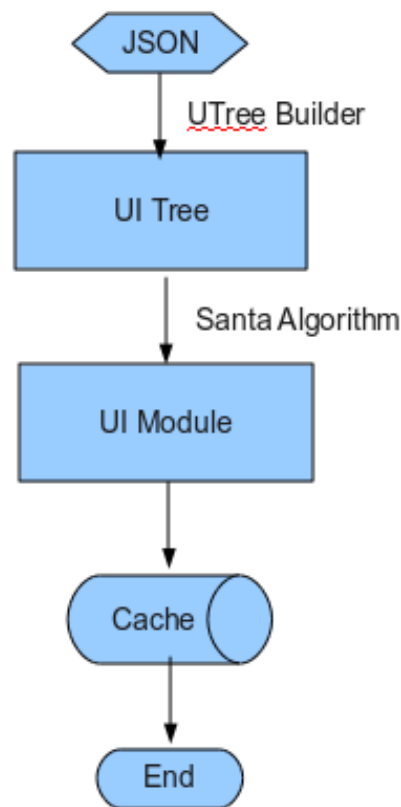
```
useTelluriumEngine(true);
```

Why is the algorithm named **Santa**. This is because I have completed most of the design and coding work during the Christmas season in 2009. It is like a gift for me from Santa Claus.

Basic Flow

Ui Module Group Locating is to locate all elements defined in a UI module by exploiting the relationship among themselves. The problem is to locate the UI module as a whole, not an individual UI element.

The UI module group locating basic flow is illustrated in the following diagram.



First, the Tellurium Engine API accepts a JSON presentation of the UI module. For example,

```

var json = [{ "obj": { "uid": "Form", "locator": { "tag": "form" }, "uiType": "Form", "key": "Form" },
  { "obj": { "uid": "Username", "locator": { "tag": "tr" }, "uiType": "Container", "key": "Form.Username" },
  { "obj": { "uid": "Label", "locator": { "direct": true, "text": "Username:", "tag": "td" },
    "uiType": "TextBox", "key": "Form.Username.Label" },
  { "obj": { "uid": "Input", "locator": { "tag": "input", "attributes": { "name": "j_username" },
    "type": "text" }, "uiType": "InputBox", "key": "Form.Username.Input" },
  { "obj": { "uid": "Password", "locator": { "tag": "tr" }, "uiType": "Container", "key": "Form.Password" },
  { "obj": { "uid": "Label", "locator": { "direct": true, "text": "Password:", "tag": "td" },
    "uiType": "TextBox", "key": "Form.Password.Label" },
  { "obj": { "uid": "Input", "locator": { "tag": "input", "attributes": { "name": "j_password" },
    "type": "password" }, "uiType": "InputBox", "key": "Form.Password.Input" },
  { "obj": { "uid": "Submit", "locator": { "tag": "input", "attributes": { "name": "submit" },
    "value": "Login", "type": "submit" }, "uiType": "SubmitButton", "key": "Form.Submit" } }];
  
```

The UI tree, i.e., UTree, builder in Tellurium Engine builds a UTree based on the JSON input. Then Tellurium Engine calls the Santa algorithm to locate all UI elements in the UI module except the elements that are defined as not **cacheable** by two UI object attributes, i.e., *lazy* and *noCacheForChildren*. Dynamic elements can be located by searching from its parent and use a subset of the Santa algorithm, which will not be covered here.

Once an element in a UI module is located, its DOM reference is stored into the UTree and an index is also created for fast access. After the Santa algorithm is finished, the UI module is stored into a cache.

Data Structures

The UI object definition in Tellurium Engine is very much similar to the one defined in Tellurium Core. The basic UI object is defined as,


```
//base UI object
var UiObject = Class.extend({
  init: function() {
    //UI object identification
    this.uid = null;

    //meta data
    this.metaData = null;

    //its parent UI object
    this.parent = null;

    //namespace, useful for XML, XHTML, XForms
    this.namespace = null;

    this.locator = null;

    //event this object should be respond to
    this.events = null;

    //should we do lazy locating or not, i.e.,
    //wait to the time we actually use this UI object
    //usually this flag is set because the content is dynamic at runtime
    this.lazy = false;

    //If it is contained in its parent or not
    this.self = false;

    this.uiType = null;

    //Tellurium Core generated locator for this UI Object
    this.generated = null;

    //dom reference
    this.domRef = null;

    //UI Module reference, which UI module this UI object belongs to
    this.uim = null;
  },
  ...
}
```

All UI objects extend this basic UI object. For example, the Container object is defined as follows.

```
var UiContainer = UiObject.extend({
  init: function(){
    this._super();
    this.uiType = 'Container';
    this.group = false;
    this.noCacheForChildren = false;
    this.components = new Hashtable();
  },
  ...
})
```

The UI module at Tellurium Engine is defined as follows.

```
function UiModule(){
  //top level UI object
  this.root = null;

  this.valid = false;

  //hold a hash table of the uid to UI objects for fast access
  this.map = new Hashtable();

  //index for uid - dom reference for fast access
```

```
this.indices = null;

//If the UI Module is relaxed, i.e., use closest match
this.relaxed = false;

//the relax details including the UIDs and their corresponding html source
this.relaxDetails = null;

//number of matched snapshots
this.matches = 0;

//scaled score (0-100) for percentage of match
this.score = 0;

//ID Prefix tree, i.e., Trie, for the lookForId operation in group locating
this.idTrie = new Trie();

//Cache hit, i.e., direct get dom reference from the cache
this.cacheHit = 0;

//Cache miss, i.e., have to use walkTo to locate elements
this.cacheMiss = 0;

//the latest time stamp for the cache access
this.timestamp = null;

//UI module dump visitor
this.dumpVisitor = new UiDumpVisitor();

//Snapshot Tree, i.e., STree
this.stree = null;
}
```

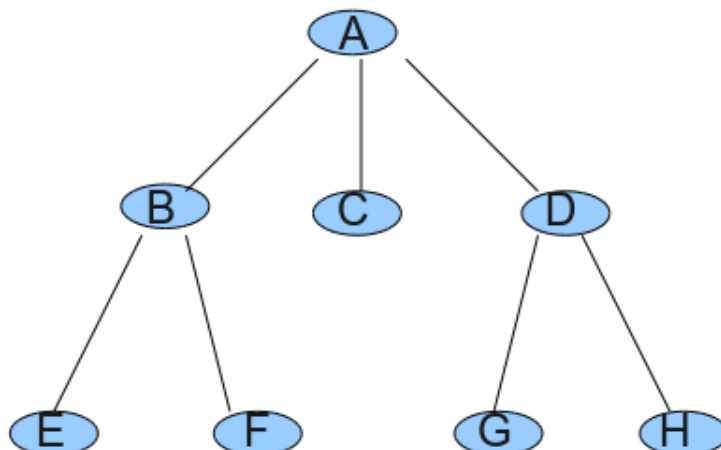
From above, you can see the UI module has two indices for fast access. One is UID to UI object mapping and the other one is the UID to DOM reference mapping.

An ID prefix tree, i.e., Trie, is built from UI module JSON presentation if the UI module includes elements with an ID attribute. The Trie is used by the Santa *lookID* operation. A more detailed Trie build process can be found on the wiki The UI Module Generating Algorithm in Trump [http://code.google.com/p/aost/wiki/UiModuleGeneratingAlgorithm#A_Trie_Based_Dictionary]

The scaled score is used by the *relax* operation for partial matching, i.e., closest match, and the score stands for how close the UI module matches the runtime DOM. 100 is a perfect match and zero is no found. This is very powerful to create robust Tellurium test code. That is to say, the Santa algorithm is adapt to changes on the web page under testing to some degree.

Locate

Assume we have UI module as shown in the following graph.



The group locating procedure is basically a breadth first search algorithm. That is to say, it starts from the root node of the UTree and then its children, its grandchildren, ..., until all node in the UTree has been searched. Santa marks color for already searched node in the UTree and you can see the color changes during the search procedure.

Algorithm

The main flow of group locating can be self-explained by the following greatly simplified code snippet.

```
UiAlg.prototype.santa = function(uimodule, rootdom){
  //start from the root element in the UI module
  if(!uimodule.root.lazy){
    //object Queue
    this.oqueue.push(uimodule.root);

    var ust = new UiSnapshot();
    //Snapshot Queue
    this.squeue.push(ust);
  }
  while(this.oqueue.size() > 0){
    var uiobj = this.oqueue.pop();
    uiobj.locate(this);
  }

  //bind snapshot to the UI Module
  this.bindToUiModule(uimodule, snapshot);

  //unmark marked UID during the locating procedure
  this.unmark();
  ...
}
```

where the locate procedure is defined as follows.

```
UiAlg.prototype.locate = function(uiobj, snapshot){
  //get full UID
  var uid = uiobj.fullUid();
  var clocator = uiobj.locator;

  //get parent's DOM reference
  var pref = snapshot.getUi(puid);

  //Build CSS selector from UI object's attributes
  var csel = this.buildSelector(clocator);
  //Starting from its parent, search for the UI element
  var $found = tejQuery(pref).find(csel);

  //if multiple matches, need to narrow down
  if($found.size() > 1){
    if(uiobj.noCacheForChildren){
      //dynamic elements, use bestEffort operation
      $found = this.bestEffort(uiobj, $found);
    }else{
      //first try lookId operation
      $found = this.lookId(uiobj, $found);
      if($found.size() > 1){
        //then try lookAhead operation
        $found = this.lookAhead(uiobj, $found);
      }
    }
  }

  ...
  if($found.size() == 0){
    if(this.allowRelax){
      //use the relax operation
      var result = this.relax(clocator, pref);
    }
  }
}
```

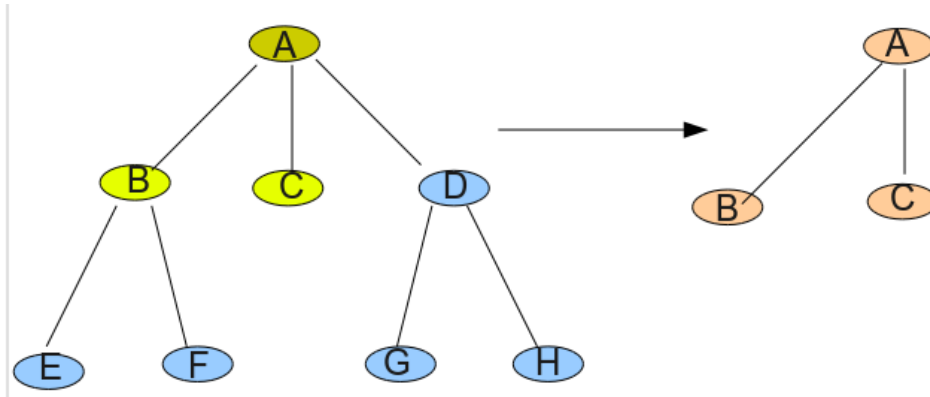
```

}
};

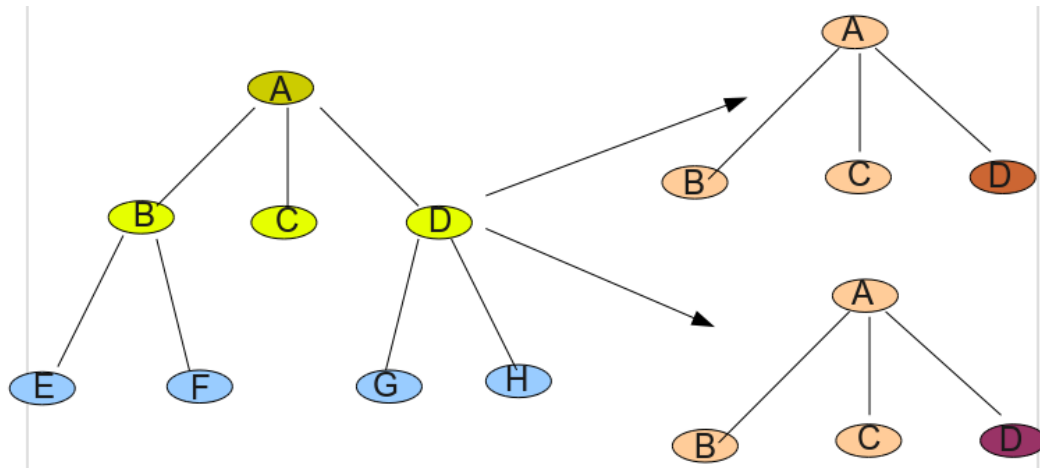
```

Branch and Trim

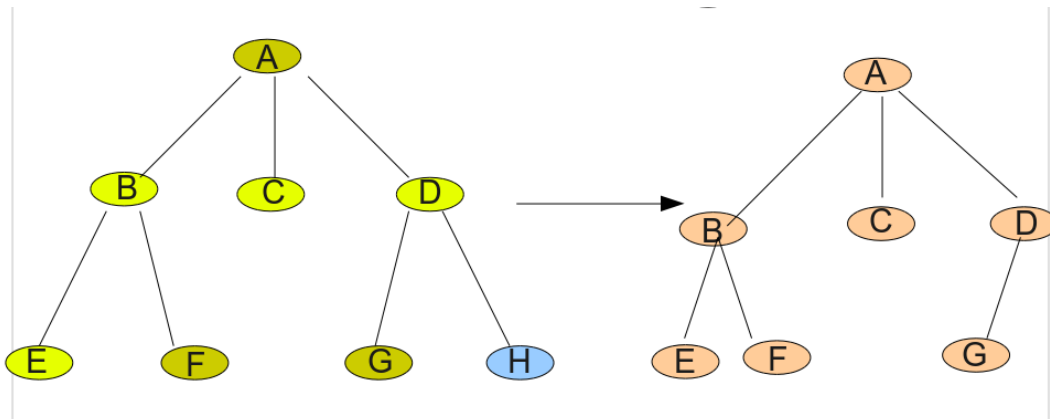
Santa is basically a branch and trim search procedure on the runtime DOM. Assume at some point, the Santa algorithm has located UI elements A, B, and C. A snapshot has been generated as shown in the following diagram.



When Santa locates UI element D, it finds two matches. Santa branches the snapshot tree and create two separate ones with each hold a different D node.



After couple steps, Santa locates the UI element G, it removes one of the snapshot trees because it cannot find G from the removed snapshot. Hence, only one snapshot tree is left.



Of course, the actual locating procedure is much more complicated than what described here. But this should be able to give you some idea on how the branch and trim procedure works.

Multiple-Match Reduction Mechanisms

As you can see from the above procedure, it would be time-consuming if Santa branches too frequently and creates too many snapshot trees because Santa needs to exploit every possible snapshot. As a result, Santa introduced the following multiple-match reduction mechanisms to reduce the number of snapshot trees it needs to search on.

Mark

When Santa locates a node at the DOM, it marks it with its UID.

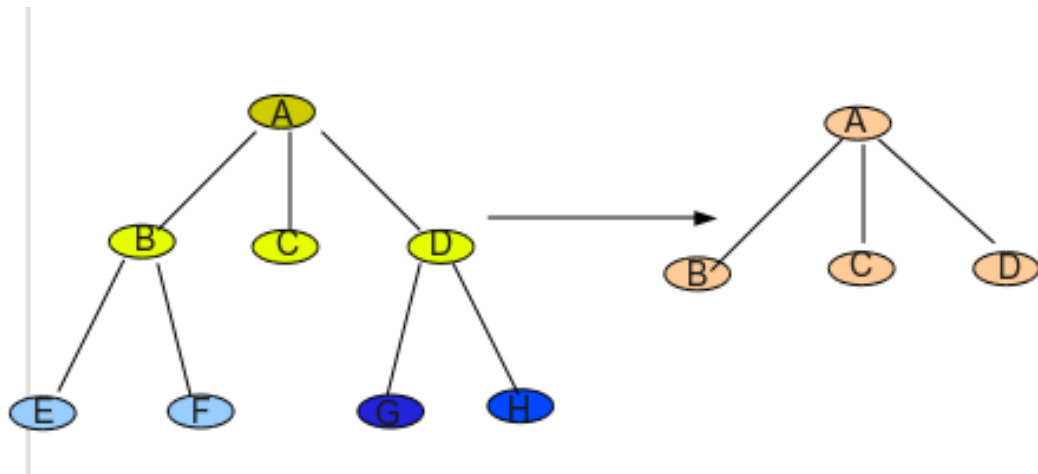
```
$found.eq(0).data("uid", uid);
```

In this way, Santa will skip this DOM node when it tries to locate other UI elements in the UI module.

When Santa finishes the group locating procedure, it unmarks all the uids from the DOM nodes.

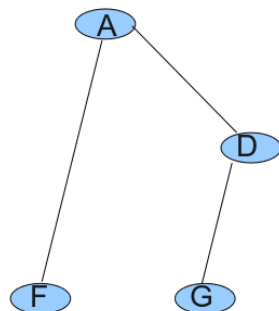
Look Ahead

Look Ahead means to look at not only the current UI element but also its children when Santa locates it. For example, when Santa locates the node D, it also looks at its children G and H. This could decrease snapshot trees at the early search stage and thus reduce the UI module locating time.



Look ID

The ID attribute uniquely defines a UI element on a web page and locating a DOM element by its ID is very fast, thus, Tellurium Engine builds an ID prefix tree, i.e., Trie, when it parses the JSON presentation of the UI module. For example, assume the UI module has four elements, A, D, F, and G, with an ID attribute. The Trie looks as follows.



When Santa locates the UI element A, it can use the IDs for element A and D to reduce multiple matches. If Santa locates element D, only the ID of element G is helpful.

Best Effort

Best effort is similar to the Look Ahead mechanism, but it is for dynamic UI elements defined Tellurium templates. For dynamic elements, Tellurium defines the following two attributes to determine whether it and its children are cacheable.

```
var UiObject = Class.extend({
  init: function(){
    ...
    //should we do lazy locating or not, i.e., wait to the time we actually use
    //this UI object usually this flag is set because the content is dynamic at runtime
    //This flag is correspond to the cacheable attribute in a Tellurium Core UI object
    this.lazy = false;
  }
});

var UiContainer = UiObject.extend({
  init: function(){
    ...
    this.noCacheForChildren = false;
  },
  ...
})
```

For a dynamic UI element defined by a UI template, it may have zero, one, or multiple matches at runtime. Santa defines a **Bonus Point** for dynamic UI elements. The bonus calculation is straightforward as shown in the following *calcBonus* method, where variable *one* is the parent DOM reference and *gsel* is a set of CSS selectors of current node's children defined by Tellurium UI templates [http://code.google.com/p/aost/wiki/UserGuide070TelluriumBasics#UI_Templates].

```
UiAlg.prototype.calcBonus = function(one, gsel){
  var bonus = 0;
  var $me = tejQuery(one);
  for(var i=0; i<gsel.length; i++){
    if($me.find(gsel[i]).size() > 0){
      bonus++;
    }
  }

  return bonus;
};
```

If the DOM matches more attributes defined by a UI template, the candidate DOM reference usually gets a higher bonus point. Santa chooses the candidate with the highest bonus point into the snapshot tree.

Relax

The relax procedure, i.e., closest match, is to match the UI attribute with the DOM node as closely as possible. A **Match Score** is defined to measure how many attributes match the one on the DOM node. The total score is scaled to 0-100 at the end. The snapshot with the highest match score is selected.

The following simplified code snippet should give you some idea of how the relax procedure works.

```
//the tag must be matched
var jqe = tag;
//attrs is the attributes defined by a UI template
var keys = attrs.keySet();
```

```
//number of properties, tag must be included
var np = 1;
//number of matched properties
var nm = 0;

if (keys != null && keys.length > 0) {
    np = np + keys.length;
    for (var m = 0; m < keys.length; m++) {
        var attr = keys[m];
        //build css selector
        var tsel = this.cssbuilder.buildSelector(attr, attrs.get(attr));
        var $mt = tejQuery(pref).find(jqs + tsel);
        if ($mt.size() > 0) {
            jqs = jqs + tsel;
            if(nm == 0){
                nm = 2;
            }else{
                nm++;
            }
        }
    }
}

//calculate match score, scaled to 100 percentage
var score = 100*nm/np;
```

As shown in the above code, the relax must satisfy one requirement, i.e., the tag name must match the one on the DOM node. Otherwise, the relax result returns as "not found".

Usage

For instance, we have the following html snippet to test.

```
<H1>FORM Authentication demo</H1>

<div class="box-inner">
    <a href="js/tellurium-test.js">Tellurium Test Cases</a>
    <input name="submit" type="submit" value="Test">
</div>

<form method="POST" action="j_security_check">
    <table border="0" cellspacing="2" cellpadding="1">
        <tr>
            <td>Username:</td>
            <td><input size="12" value="" name="j_username" maxlength="25" type="text"></td>
        </tr>
        <tr>
            <td>Password:</td>
            <td><input size="12" value="" name="j_password" maxlength="25" type="password"></td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input name="submit" type="submit" value="Login">
            </td>
        </tr>
    </table>
</form>
```

The correct UI module is shown as follows,

```
ui.Container(uid: "Form", clocator: [tag: "table"]){
    Container(uid: "Username", clocator: [tag: "tr"]){
        TextBox(uid: "Label", clocator: [tag: "td", text: "Username:", direct: "true"])
        InputBox(uid: "Input", clocator: [tag: "input", type: "text",
            name: "j_username"])
    }
    Container(uid: "Password", clocator: [tag: "tr"]){
```

```

        TextBox(uid: "Label", clocator: [tag: "td", text: "Password:", direct: "true"])
        InputBox(uid: "Input", clocator: [tag: "input", type: "password",
            name: "j_password"])
    }
    SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit",
        value: "Login", name: "submit"])
}

```

Assume the html was changed recently and you still use the following UI module defined some time ago.

```

ui.Container(uid: "ProblematicForm", clocator: [tag: "table"]){
    Container(uid: "Username", clocator: [tag: "tr"]){
        TextBox(uid: "Label", clocator: [tag: "td", text: "Username:", direct: "true"])
        InputBox(uid: "Input", clocator: [tag: "input", type: "text", name: "j"])
    }
    Container(uid: "Password", clocator: [tag: "tr"]){
        TextBox(uid: "Label", clocator: [tag: "td", text: "Password:", direct: "true"])
        InputBox(uid: "Input", clocator: [tag: "input", type: "password", name: "j"])
    }
    SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit", value: "logon",
        name: "submit"])
}

```

Here are the differences:

```

InputBox(uid: "Input", clocator: [tag: "input", type: "text",
    name: "j_username"])
InputBox(uid: "Input", clocator: [tag: "input", type: "text",
    name: "j"])

InputBox(uid: "Input", clocator: [tag: "input", type: "password",
    name: "j_password"])
InputBox(uid: "Input", clocator: [tag: "input", type: "password",
    name: "j"])

SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit",
    value: "Login", name: "submit"])
SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit",
    value: "logon", name: "submit"])

```

What will happen without using the Santa algorithm? Your tests will be broken because the generated locators will not be correct any more. But if you use the latest Tellurium 0.7.0 snapshot, you will notice that the tests still work if you allow Tellurium to do closest match by calling

```
useClosestMatch(true);
```

The magic is that the new Tellurium Engine will locate the UI module as a whole. It may have some trouble to find some individual UI elements such as "ProblematicForm.Username.Input", but it has no problem to locate the whole UI module structure in the DOM.

Apart from that, Tellurium 0.7.0 also provides a handy method for you to validate your UI module. For example, if you call

```
validateUiModule("ProblematicForm");
```

You will get the detailed validation results including the closest matches.

UI Module Validation Result for ProblematicForm

```
-----  
  
    Found Exact Match: false  
  
    Found Closest Match: true  
  
    Match Count: 1  
  
    Match Score: 85.764  
  
    Closest Match Details:  
  
    --- Element ProblematicForm.Submit -->  
  
    Composite Locator: <input name="submit" value="logon"  
                        type="submit"/>  
  
    Closest Matched Element: <input name="submit" value="Login"  
                             type="submit">  
  
    --- Element ProblematicForm.Username.Input -->  
  
    Composite Locator: <input name="j" type="text"/>  
  
    Closest Matched Element: <input size="12" value="" name="j_username"  
                             maxLength="25" type="text">  
  
    --- Element ProblematicForm.Password.Input -->  
  
    Composite Locator: <input name="j" type="password"/>  
  
    Closest Matched Element: <input size="12" value="" name="j_password"  
                             maxLength="25" type="password">  
  
-----
```

UI Module Caching

From Tellurium 0.6.0, we provides the cache capability for CSS selectors so that we can reuse them without doing re-locating. In 0.7.0, we move a step further to cache the whole UI module on the Engine side. Each UI module cache holds a snapshot of the DOM references for most of the UI elements in the UI module. The exceptions are dynamic web elements defined by Tellurium UI templates. For these dynamic web elements, the Engine will try to get the DOM reference of its parent and then do locating inside this subtree with its parent node as the root, which will improve the locating speed a lot.

On the Engine side the cache is defined as

```
function TelluriumCache(){  
  
    //global flag to decide whether to cache jQuery selectors  
    this.cacheOption = false;  
  
    //cache for UI modules  
    this.sCache = new Hashtable();  
  
    this.maxCacheSize = 50;  
  
    this.cachePolicy = discardOldCachePolicy;  
  
    //Algorithm handler for UI  
    this.uiAlg = new UiAlg();  
}
```

Tellurium Engine provides the following cache eviction policies:

- DiscardNewPolicy: discard new jQuery selector.
- DiscardOldPolicy: discard the oldest jQuery selector measured by the last update time.
- DiscardLeastUsedPolicy: discard the least used jQuery selector.
- DiscardInvalidPolicy: discard the invalid jQuery selector first.

To turn on and off the caching capability, you just simply call the following method in your code.

```
void useCache(boolean isUse);
```

Or use the following methods to do fine control of the cache.

```
public void enableCache();
public boolean disableCache();
public boolean cleanCache();
public boolean getCacheState();
public void setCacheMaxSize(int size);
public int getCacheSize();
public void useDiscardNewCachePolicy();
public void useDiscardOldCachePolicy();
public void useDiscardLeastUsedCachePolicy();
public void useDiscardInvalidCachePolicy();
public String getCurrentCachePolicy();
public Map<String, Long> getCacheUsage();
```

How the cached UI modules are used? Tellurium Core checks if a UI module has been located or not when it calls a method on the UI module. If not, it calls the `useUiModule(json)` API to Engine and the Engine calls the following method to do group locating using the Santa algorithm and then push the UI module into the cache.

```
TelluriumCache.prototype.useUiModule = function(jsonarray){
    var uim = new UiModule();
    uim.parseUiModule(jsonarray);
    var response = new UiModuleLocatingResponse();
    var result = this.uiAlg.santa(uim, null);
    if(result){
        //set the UI Module to be valid after it is located
        uim.valid = true;
        var id = uim.getId();
        var cached = this.getCachedData(id);
        if (cached == null) {
            this.addToCache(id, uim);
        } else {
            this.updateToCache(id, uim);
        }

        response.id = id;
        response.relaxed = uim.relaxed;
        if (!response.relaxed)
            response.found = true;
        response.relaxDetails = uim.relaxDetails;
        response.matches = uim.matches;
        response.score = uim.score;
    }

    return response;
};
```

For subsequent calls, the Engine will check the UI object UID to see if it can find the UI module from the cache. If the UI module has been cached, a UI object traverse is called to get back the runtime dom reference for the corresponding UI object.

```
TelluriumCache.prototype.getCachedUiElement = function(uid){  
  
    var uiid = new Uiid();  
    uiid.convertToUiid(uid);  
  
    if(uiid.size() > 0){  
        var first = uiid.peek();  
        var uim = this.sCache.get(first);  
        if(uim != null){  
            var domref = uim.index(uid);  
            if(domref == null){  
                uim.increaseCacheMiss();  
                //if could not find directly from the UI module indices,  
                //then try to use walkTo to find the element first  
                //and then its domRef  
                var context = new WorkflowContext();  
                context.alg = this.uiAlg;  
                var obj = uim.walkTo(context, uid);  
                if(obj != null){  
                    domref = context.domRef;  
                }  
            }else{  
                uim.increaseCacheHit();  
            }  
            return domref;  
        }  
    }  
    return null;  
};
```

To measure the cache usage, Tellurium Engine defined the following class.

```
function CacheUsage(){  
    this.totalCall = 0;  
    this.cacheHit = 0;  
    //percentage of the cacheHit/totalCall  
    this.usage = 0;  
}
```

The following method can be called from Tellurium Core to get back the actual cache usage.

```
public Map<String, Long> getCacheUsage();
```

New APIs

With the addition of jQuery in the Engine, it is pretty easy to create a set of custom Selenium methods. For example, we can create a `getAllText()` method to get back all texts.

```
TelluriumApi.prototype.getAllText = function(locator) {  
    var element = this.cacheAwareLocate(locator);  
    var out = [];  
    var $e = tejQuery(element);  
    $e.each(function() {  
        out.push(tejQuery(this).text());  
    });  
};
```

```

    return out;
};

```

Tellurium Engine provides another running mode, i.e., the Tellurium core does not need to generate the runtime locator anymore and it can simply pass the object UID to the Engine. For example, Tellurium Engine only needs UID in the following method.

```

TelluriumApi.prototype.getAllTableBodyText = function(uid) {
    var context = new WorkflowContext();
    context.alg = this.cache.uiAlg;

    var obj = this.cache.walkToUiObjectWithException(context, uid);
    if(obj.respondsToWithException("getAllBodyCell")){
        var out = obj.getAllBodyCell(context, this.textWorker);

        return out;
    }

    return null;
};

```

Right now, new Tellurium jQuery-based APIs include

```

TelluriumApi.prototype.blur = function(locator);

TelluriumApi.prototype.click = function(locator);

TelluriumApi.prototype.clickAt = function(locator, coordString);

TelluriumApi.prototype.doubleClick = function(locator);

TelluriumApi.prototype.fireEvent = function(locator, event);

TelluriumApi.prototype.focus = function(locator);

TelluriumApi.prototype.typeKey = function(locator, key);

TelluriumApi.prototype.keyDown = function(locator, key);

TelluriumApi.prototype.keyPress = function(locator, key);

TelluriumApi.prototype.keyUp = function(locator, key);

TelluriumApi.prototype.mouseOver = function(locator);

TelluriumApi.prototype.mouseDown = function(locator);

TelluriumApi.prototype.mouseDownRight = function(locator);

TelluriumApi.prototype.mouseEnter = function(locator);

TelluriumApi.prototype.mouseLeave = function(locator);

TelluriumApi.prototype.mouseOut = function(locator);

TelluriumApi.prototype.submit = function(locator);

TelluriumApi.prototype.check = function(locator);

TelluriumApi.prototype.uncheck = function(locator);

TelluriumApi.prototype.isElementPresent = function(locator);

TelluriumApi.prototype.getAttribute = function(locator, attributeName);

TelluriumApi.prototype.waitForPageToLoad = function(timeout);

TelluriumApi.prototype.type = function(locator, val);

```

```
TelluriumApi.prototype.select = function(locator, optionLocator);
TelluriumApi.prototype.addSelection = function(locator, optionLocator);
TelluriumApi.prototype.removeSelection = function(locator, optionLocator);
TelluriumApi.prototype.removeAllSelections = function(locator);
TelluriumApi.prototype.open = function(url);
TelluriumApi.prototype.getText = function(locator);
TelluriumApi.prototype.isChecked = function(locator);
TelluriumApi.prototype.isVisible = function(locator);
TelluriumApi.prototype.isEditable = function(locator) ;
TelluriumApi.prototype.getXpathCount = function(xpath);
TelluriumApi.prototype.getAllText = function(locator);
TelluriumApi.prototype.getCssSelectorCount = function(locator);
TelluriumApi.prototype.getCSS = function(locator, cssName);
TelluriumApi.prototype.isDisabled = function(locator);
TelluriumApi.prototype.getListSize = function(locator, separators);
TelluriumApi.prototype.getCacheState = function();
TelluriumApi.prototype.enableCache = function();
TelluriumApi.prototype.disableCache = function();
TelluriumApi.prototype.cleanCache = function();
TelluriumApi.prototype.setCacheMaxSize = function(size);
TelluriumApi.prototype.getCacheSize = function();
TelluriumApi.prototype.getCacheMaxSize = function();
TelluriumApi.prototype.getCacheUsage = function();
TelluriumApi.prototype.addNamespace = function(prefix, namespace);
TelluriumApi.prototype.getNamespace = function(prefix);
TelluriumApi.prototype.useDiscardNewPolicy = function();
TelluriumApi.prototype.useDiscardOldPolicy = function();
TelluriumApi.prototype.useDiscardLeastUsedPolicy = function();
TelluriumApi.prototype.useDiscardInvalidPolicy = function();
TelluriumApi.prototype.getCachePolicyName = function();
TelluriumApi.prototype.useUiModule = function(json);
TelluriumApi.prototype.isUiModuleCached = function(id);
TelluriumApi.prototype.getEngineState = function();
TelluriumApi.prototype.useEngineLog = function(isUse);
```

As you can see, most of the new APIs have the same signature as the Selenium ones so that your test code is agnostic to which test driving engine that you use. You can always switch between the Tellurium Engine and Selenium Engine by the following API at Tellurium core.

```
public void useTelluriumApi(boolean isUse);
```

Debug Tellurium Engine

Tellurium Engine is similar to Selenium core to drive browser events to simulate users who physically work on the web application. The basic idea is to write a Javascript test to call Tellurium Engine and embed the test code in the header of the html page that we want to test. Then we can set a breakpoint in the JavaScript code using Firebug. After refresh the web page, we can run the test code, which will stop at the breakpoint. One obstacle is that some parts of Tellurium Engine need to call Selenium core, we work around this problem by mocking Selenium Core.

First, we installed a Jetty server and use the Jetty server to load up the web page that we want to run tellurium Engine tests on.

Take the following html source as an example,

```
<H1>FORM Authentication demo</H1>

<div class="box-inner">
  <a href="js/tellurium-test.js">Tellurium Test Cases</a>
  <input name="submit" type="submit" value="Test"
    onclick="teTestCase.testSuite();" />
</div>

<form method="POST" action="j_security_check">
  <table border="0" cellspacing="2" cellpadding="1">
    <tr>
      <td>Username:</td>
      <td><input size="12" value="" name="j_username" maxlength="25"
        type="text"></td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input size="12" value="" name="j_password" maxlength="25"
        type="password"></td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <input name="submit" type="submit" value="Login">
      </td>
    </tr>
  </table>
</form>
```

First, we added the following headers to the above html source.

```
<head>
  <title>Tellurium Test Page</title>
  <script src="js/selenium-mock.js"> </script>
  <script src="http://localhost:4444/selenium-server/core/scripts/jquery-1.4.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/json2.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/Utils.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-logging.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-api.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-cache.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-extensions.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-selector.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-uibasic.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-uiobj.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-uimodule.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-uisnapshot.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-uialg.js"/>
  <script src="http://localhost:4444/selenium-server/core/scripts/tellurium-uiextra.js"/>
```

```
<script src="http://localhost:4444/selenium-server/core/scripts/tellurium.js"/>
<script src="js/tellurium-test.js"> </script>
<script type="text/javascript">
    tejQuery(document).ready(function(){
        var testcase = new TelluriumTestCase();
        testcase.testLogonUiModule();
    });
</script>
</head>
```

where the tellurium-test.js is the Engine test script. The above header loads up the Tellurium Engine code when the web page is served by Jetty.

Then, we defined a test class as follows,

```
function TelluriumTestCase(){

};

TelluriumTestCase.prototype.testLogonUiModule = function(){
    var json = [{"obj":{"uid":"Form","locator":{"tag":"form"},"uiType":"Form"},
    {"key":"Form"}, {"obj":{"uid":"Username","locator":{"tag":"tr"},
    "uiType":"Container"},"key":"Form.Username"}, {"obj":{"uid":"Label","locator":
    {"direct":true,"text":"Username:","tag":"td"},"uiType":"TextBox"},"key":
    "Form.Username.Label"}, {"obj":{"uid":"Input","locator":{"tag":"input",
    "attributes":{"name":"j_username","type":"text"},"uiType":"InputBox"},
    "key":"Form.Username.Input"}, {"obj":{"uid":"Password","locator":{"tag":"tr"},
    "uiType":"Container"},"key":"Form.Password"}, {"obj":{"uid":"Label","locator":
    {"direct":true,"text":"Password:","tag":"td"},"uiType":"TextBox"},
    "key":"Form.Password.Label"}, {"obj":{"uid":"Input","locator":{"tag":"input",
    "attributes":{"name":"j_password","type":"password"},"uiType":"InputBox"},
    "key":"Form.Password.Input"}, {"obj":{"uid":"Submit","locator":{"tag":"input",
    "attributes":{"name":"submit","value":"Login","type":"submit"},"uiType":
    "SubmitButton"},"key":"Form.Submit"}];
    tellurium.logManager.isUseLog = true;
    var uim = new UiModule();
    uim.parseUiModule(json);
    var alg = new UiAlg();
    var dom = tejQuery("html");
    alg.santa(uim, dom);
    tellurium.cache.cacheOption = true;
    tellurium.cache.addToCache("Form", uim);
    var context = new WorkflowContext();
    context.alg = alg;
    var uuid = new Uuid();
    var uinput = uim.walkTo(context,
        uuid.convertToUuid("Form.Username.Input"));
    var pinput = uim.walkTo(context,
        uuid.convertToUuid("Form.Password.Input"));
    var smt = uim.walkTo(context, uuid.convertToUuid("Form.Submit"));
    tellurium.teApi.getHTMLSource("Form");
    var attrs = [{"val":"text","key":"type"}];
    var teuids = tellurium.teApi.getUiByTag("input", attrs);
    fbLog("result ", teuids);
};
```

The variable json is the JSON presentation of the UI module. You can obtain the JSON string of a UI module by calling the following method in DslContext.

```
public String toString(String uid);
```

The following test case first runs the Santa algorithm to do group locating and then call *walkTo* to find the DOM reference for the UI element. After that, we test the two Tellurium APIs, *getHTMLSource* and *getUiByTag*.

As we said, Tellurium Engine needs to call Selenium Core code somewhere. we need to mock up Selenium core as follows.

```
function Selenium(){
    this.browserbot = new BrowserBot();
};

function BrowserBot(){

};

BrowserBot.prototype.findElement = function(locator){
    if(locator.startsWith("jquery=")){
        return tejQuery(locator.substring(7));
    }

    return null;
};

function SeleniumError(message) {
    var error = new Error(message);
    if (typeof(arguments.caller) != 'undefined') {
        var result = '';
        for (var a = arguments.caller; a != null; a = a.caller) {
            result += '> ' + a.callee.toString() + '\n';
            if (a.caller == a) {
                result += '*';
                break;
            }
        }
        error.stack = result;
    }
    error.isSeleniumError = true;
    return error;
}

var selenium = new Selenium();
```

Firebug provides very powerful console logging capability, where you can inspect the JavaScript object. Tellurium Engine provides the following wrapper for Firebug console logging.

```
function fbLog(msg, obj){
    if (typeof(console) != "undefined") {
        console.log(msg, obj);
    }
}

function fbInfo(msg, obj){
    if (typeof(console) != "undefined") {
        console.info(msg, obj);
    }
}

function fbDebug(msg, obj){
    if (typeof(console) != "undefined") {
        console.debug(msg, obj);
    }
}

function fbWarn(msg, obj){
    if (typeof(console) != "undefined") {
        console.warn(msg, obj);
    }
}

function fbError(msg, obj){
    if (typeof(console) != "undefined") {
        console.trace();
        console.error(msg, obj);
    }
}
```



```
}

function fbTrace(){
    if (typeof(console) != "undefined") {
        console.trace();
    }
}

function fbAssert(expr, obj){
    if (typeof(console) != "undefined") {
        console.assert(expr, obj);
    }
}

function fbDir(obj){
    if (typeof(console) != "undefined") {
        console.dir(obj);
    }
}
```

For browsers other than Firefox, Tellurium provides the Firebug Lite [http://code.google.com/p/aost/wiki/Tellurium070Update#Engine_Logging] in the custom selenium server so that you can still use the above logging wrapper.

To run the above test page, first, we copy the above page to `JETTY_HOME/webapps/test/` directory. Copy the test script and Selenium mock script to `JETTY_HOME/webapps/test/js/` directory. Then start the Jetty server and open the Firefox browser to point to `http://localhost:8080/testpagename.html`. we can see all JavaScript code including Tellurium Engine code and the test script. Set a breakpoint somewhere and refresh the page, Firebug will stop at breakpoint and then we can start to debug the Engine.

Tellurium UI Module Visual Effect

Have you ever thought of seeing the actual UI on the web page under testing? Tellurium 0.7.0 adds a cool feature to show this visual effect.

Tellurium provides a *show* command to display the UI module that you defined on the actual web page.

```
public show(String uid, int delay);
```

where *uid* is the UI module name and *delay* is in milliseconds. In the meanwhile, Tellurium Core exposes the following two methods for users to start showing UI and clean up UI manually.

```
public void startShow(String uid);
```

```
public void endShow(String uid);
```

How it Works ? Under the hood, Tellurium Engine does the following things.

Build a Snapshot Tree

The Snapshot Tree, or *STree* in short, is different from the UI module. The UI module defines how the UI looks like. Even the UI module group locating algorithm - Santa [<http://code.google.com/p/aost/wiki/SantaUiModuleGroupLocatingAlgorithm>] only locates cachable UI elements and leaves out the dynamic elements. The snapshot tree, however, needs to include every UI elements inside the UI module. For example, the following Google Book List UI module is very simple.

```
ui.Container(uid: "GoogleBooksList", clocator: [tag: "table", id: "hp_table"]){
  List(uid: "subcategory", clocator: [tag: "td", class: "sidebar"],
    separator: "div") {
    Container(uid: "all") {
      TextBox(uid: "title", clocator: [tag: "div", class: "sub_cat_title"])
      List(uid: "links", separator: "p") {
        UrlLink(uid: "all", clocator: [:])
      }
    }
  }
}
```

But its STree may include many book categories and books.

A snapshot tree includes the following different types of nodes.

SNode

SNode can present a non-container type UI object, i.e., UI element without any child.

```
var UiSNode = Class.extend({
  init: function() {

    //parent's rid
    this.pid = null;

    //rid, runtime id
    this.rid = null;

    //point to its parent in the UI SNAPSHOT tree
    this.parent = null;

    //UI object, which is defined in the UI module, reference
    this.objRef = null;

    //DOM reference
    this.domRef = null;
  },
  ...
})
```

CNode

CNode is a container type node and it has children.

```
var UiCNode = UiSNode.extend({
  init: function(){
    this._super();
    //children nodes, regular UI Nodes
    this.components = new Hashtable();
  },
  ...
})
```

TNode

The TNode stands for a table with headers, footers, and one or multiple bodies.

```
var UiTNode = UiSNode.extend({
  init: function(){
    this._super();

    //header nodes
    this.headers = new Hashtable();

    //footer nodes
    this.footers = new Hashtable();

    //body nodes
    this.components = new Hashtable();
  },
  ...
});
```

Finally, the Snapshot tree is defined as

```
function UiSTree(){
  //the root node
  this.root = null;

  //the reference point to the UI module that the UI Snapshot tree is derived
  this.uimRef = null;
}
```

The STree build process is quite complicated. The basic idea is to use the cached DOM references in a cached UI module and use a subset of the Santa algorithm to locate dynamic UI elements.

STree Visitors

The STree defines a traverse method, so that we can pass in different visitors to work on each individual node in the tree for different purpose.

```
UiSTree.prototype.traverse = function(context, visitor){
  if(this.root != null){
    this.root.traverse(context, visitor);
  }
};

var UiSNode = Class.extend({
  ...
  traverse: function(context, visitor){
    visitor.visit(context, this);
  },
  ...
});
```

The STree Visitor class is defined as

```
var STreeVisitor = Class.extend({
  init: function(){

  },

  visit: function(context, snode){

  }
});
```

Tellurium Engine also defines a Visitor Chain to pass in multiple visitors.

```
var STreeChainVisitor = Class.extend({
  init: function(){
    this.chain = new Array();
  },

  removeAll: function(){
    this.chain = new Array();
  },

  addVisitor: function(visitor){
    this.chain.push(visitor);
  },

  size: function(){
    return this.chain.length;
  },

  visit: function(context, snode){
    for(var i=0; i<this.chain.length; i++){
      this.chain[i].visit(context, snode);
    }
  }
});
```

For the show UI method, the following two Visitors are implemented.

Outline Visitor

The outline visitor is used to mark the UI elements inside a UI module to differentiate the UI elements in the UI module from other UI elements.

outline visitor include a worker to outline an element.

```
var UiOutlineVisitor = STreeVisitor.extend({
  visit: function(context, snode){
    var elem = snode.domRef;
    tejQuery(elem).data("originalOutline", elem.style.outline);
    elem.style.outline = tellurium.outlines.getDefaultOutline();
  }
});
```

and a cleaner to restore the UI to the original one.

```
var UiOutlineCleaner = STreeVisitor.extend({
  visit: function(context, snode){
    var elem = snode.domRef;
    var $elem = tejQuery(elem);
    var outline = $elem.data("originalOutline");
    elem.style.outline = outline;
    $elem.removeData("originalOutline");
  }
});
```

Tooltip Visitor

The Tooltip visitor is used to show the full UID of an element in a tooltip fasion. Tellurium exploited jQuery Simpletip plugin [<http://craigsworks.com/projects/simpletip/>] to achieve this visual effect. In additional to that, the tooltip visitor also changes the outlines of the selected UI elements.

We need to change Simpletip plugin code a bit. By default Simpletip create a div element and insert this element inside a UI element that you want to show tooltip. But this would not work if the tag of the UI element is "input", thus, we changed this code to use insertAfter as follows.

```
var tooltip = tejQuery(document.createElement('div'))
    .addClass(conf.baseClass)
    .addClass("teengine")
    .addClass( (conf.fixed) ? conf.fixedClass : '' )
    .addClass( (conf.persistent) ? conf.persistentClass : '' )
    .css({'z-index': '2', 'position': 'right', 'padding': '8',
        'color': '#303030', 'background-color': '#f5f5b5',
        'border': '1', 'solid': '#DECA7E', 'font-family':
        'sans-serif', 'font-size': '8', 'line-height':
        '12px', 'text-align': 'center'
    })
    .html(conf.content)
    .insertAfter(elem);
```

In addition to that, we added a class "teengine" for the div tag so that it will conflict with users' web content.

Like the outline visitor, the tooltip visitor includes a worker to set up the visual effects,

```
var UiSimpleTipVisitor = STreeVisitor.extend({
    visit: function(context, snode) {
        var elem = snode.domRef;
        var frid = snode.getFullRid();

        var $elem = tejQuery(elem);
        $elem.data("level", snode.getLevel());
        $elem.simpletip({
            // Configuration properties
            onShow: function() {
                var $parent = this.getParent();
                var parent = $parent.get(0);
                var level = $parent.data("level");

                var outline = $parent.data("outline");
                if(outline == undefined || outline == null){
                    $parent.data("outline", parent.style.outline);
                }

                parent.style.outline = tellurium.outlines.getOutline(level);
            },
            onHide: function() {
                var $parent = this.getParent();
                var parent = $parent.get(0);

                parent.style.outline = $parent.data("outline");
            },
            content: convertRidToUid(frid),
            fixed: false
        });
    }
});
```

and a cleaner to remove the visual effects.

```
var UiSimpleTipCleaner = STreeVisitor.extend({
    visit: function(context, snode){
        var elem = snode.domRef;
        var frid = snode.getFullRid();
```

```

        var $elem = tejQuery(elem);
        $elem.removeData("outline");
        $elem.removeData("level");
        $elem.find("~ div.teengine.tooltip").remove();
    }
});

```

Demo

To show the UI module visual effect, we consider a Login UI module as follows.

```

public class FormExampleModule extends DslContext {

    public void defineUi() {
        ui.Form(uid: "Form", clocator: [tag: "table"]){
            Container(uid: "Username", clocator: [tag: "tr"]){
                TextBox(uid: "Label", clocator: [tag: "td", text: "Username:", direct: "true"])
                InputBox(uid: "Input", clocator: [tag: "input", type: "text",
                    name: "j_username"])
            }
            Container(uid: "Password", clocator: [tag: "tr"]){
                TextBox(uid: "Label", clocator: [tag: "td", text: "Password:", direct: "true"])
                InputBox(uid: "Input", clocator: [tag: "input", type: "password",
                    name: "j_password"])
            }
            SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit", value: "Login",
                name: "submit"])
        }
    }
}

```

Before the `startShow("Form")` command, the UI looks as follows,

and after the `startShow` command, the UI module is outlined by the outline visitor.

FORM Authentication demo

[Tellurium Test Cases](#)

Test

Menu Nav

Username:

Password:

Login

Test:

Test

Logo

Image 5



If you mouse over the UI module, you will see the visual effects as follows.

FORM Authentication demo

[Tellurium Test Cases](#)

Test

Menu Nav

Username:

Form.Username.Input

Form.Username

Password:

Login

Form

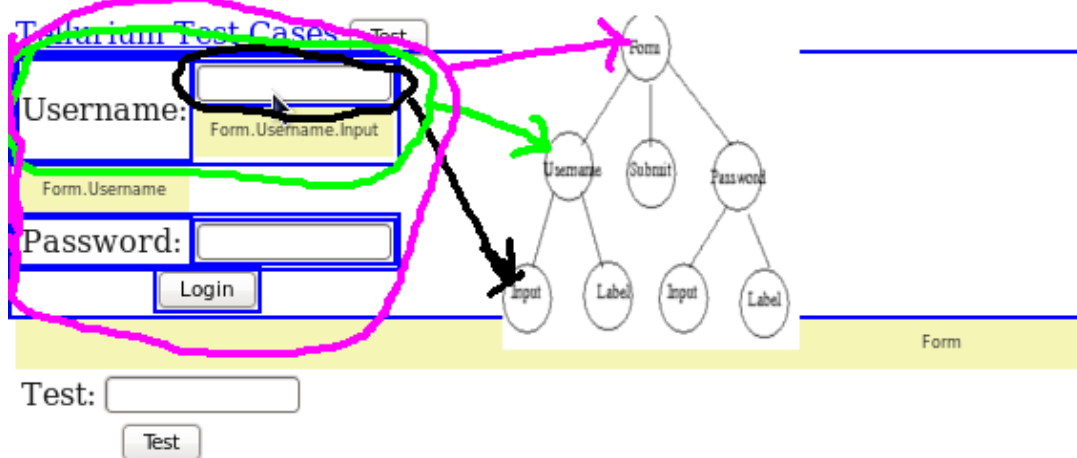
Test:

Test



You may be a bit confused by the multiple UID labels. For example, When user hives over the Username Input box, its parent "Username" and its grandparent "Form" are shown as well. We added color coding to the outlines. Different levels in the hierarchy are presented by different colors. How the layout maps to each individual UI element in the STree can be illustrated more clearly by the following figure.

FORM Authentication demo



Once you call the following command

```
endShow( "Form" );
```

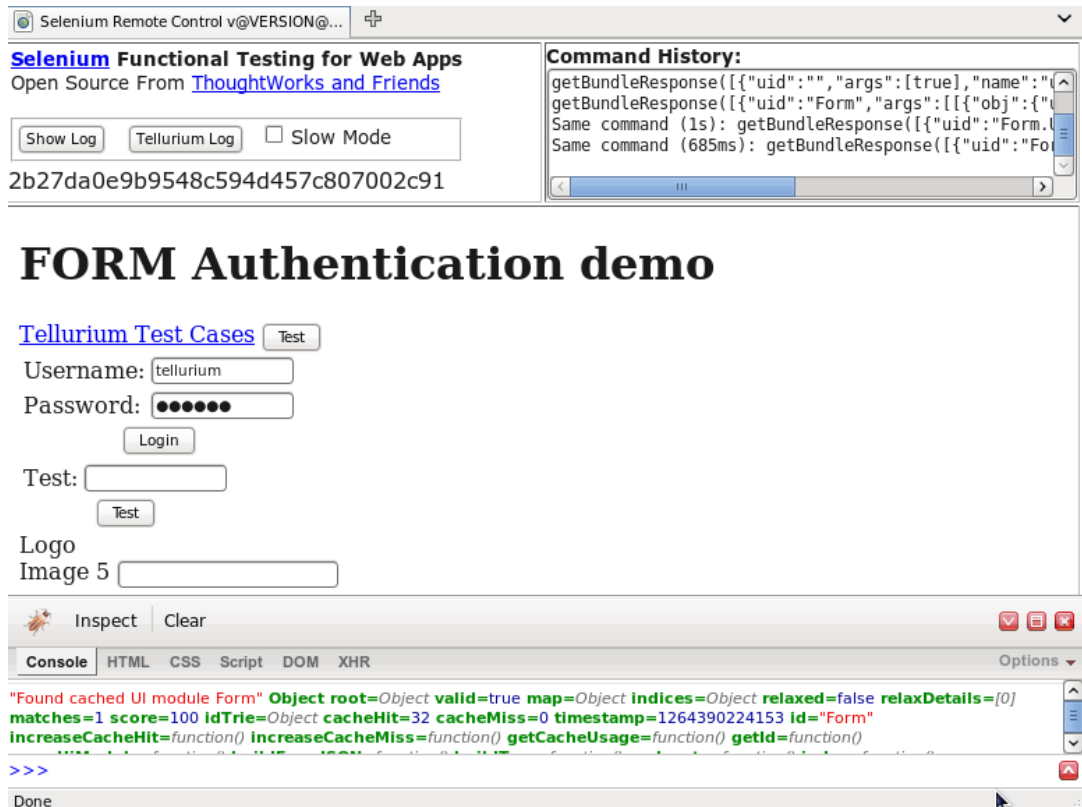
The visual effects are removed and the UI is restored to the original one.

Engine Logging

Tellurium Engine uses Firebug Lite [<http://getfirebug.com/lite.html>] to add debug information to the console. By default the Firebug Lite is off and you will only see a Firebug icon on the bottom right as shown in the following figure.



If you click on the icon and the Firebug Lite console will appear and the log information will be shown on the console as follows.



To turn on the debug, you should either click on the "Tellurium Log" button or call the following method from your test case.

```
void useEngineLog(boolean isUse);
```

JavaScript Error Stack

We utilized the JavaScript Stack Trace project [http://github.com/emwendelin/javascript-stacktrace] to refactor Selenium Errors in Selenium Core so that the JavaScript Error Stack will be passed back Tellurium Core.

The implementation is in `htmlutils.js` in Selenium Core as follows.

```
function SeleniumError(message) {
    if(tellurium.logManager.isUseLog){
        var jstack = printStackTrace();
        if(jstack != null && typeof(jstack) != 'undefined'){
            message = message + "\nJavaScript Error Stack: \n"
                + jstack.join('\n\n');
        }
    }
    var error = new Error(message);
    if (typeof(arguments.caller) != 'undefined') {
        var result = '';
        for (var a = arguments.caller; a != null; a = a.caller) {
            result += '> ' + a.caller.toString() + '\n';
            if (a.caller == a) {
                result += '*';
                break;
            }
        }
        error.stack = result;
    }
}
```

```

        error.isSeleniumError = true;
        fbError("Selenium Error: " + message, error);
        return error;
    }

```

where `printStackTrace()` is the `stacktrace` project API. Be aware that the JavaScript error stack will only be passed back to Tellurium Core if the Engine log is enabled by calling.

```
useEngineLog(true);
```

Example output:

```

com.thoughtworks.selenium.SeleniumException: ERROR: Element uimcal={"rid":
  "search.search_project_button", "locator": "jquery=form:group(input[name=q],
  input[value=Search projects][type=submit], input[value=Search the Web]
  [type=submit]) input[value=Search projects][type=submit]} not found
JavaScript Error Stack:
{anonymous}(null)@http://localhost:4444/selenium-server/core/scripts/utils.js:589

printStackTrace()@http://localhost:4444/selenium-server/core/scripts/utils.js:574

SeleniumError("Element uimcal={"rid":"search.search_project_button",
  \"locator\":\"jquery=form:group(input[name=q], input[value=Search projects]
  [type=submit], input[value=Search the Web][type=submit]) input
  [value=Search projects][type=submit]} not found")
  @http://localhost:4444/selenium-server/core/scripts/htmlutils.js:806

{anonymous}({"uimcal":{"rid":"search.search_project_button","locator":
  \"jquery=form:group(input[name=q], input[value=Search projects][type=submit],
  input[value=Search the Web][type=submit]) input[value=Search projects]
  [type=submit]}")
  @http://localhost:4444/selenium-server/core/scripts/selenium-browserbot.js:1341

{anonymous}({"uimcal":{"rid":"search.search_project_button","locator":
  \"jquery=form:group(input[name=q], input[value=Search projects][type=submit],
  input[value=Search the Web][type=submit]) input[value=Search projects]
  [type=submit]}")
  @http://localhost:4444/selenium-server/core/scripts/selenium-api.js:227

{anonymous}([object Object],[object Object])@http://localhost:4444/
  selenium-server/core/scripts/tellurium.js:922

{anonymous}()@http://localhost:4444/selenium-server/core/scripts/tellurium.js:876

{anonymous}(["[\"uid\":\"search.search_project_button\", \"args\":
  [\"jquery=form:group(input[name=q], input[value=Search projects][type=submit],
  input[value=Search the Web][type=submit]) input[value=Search projects]
  [type=submit]\", \"name\":\"click\", \"sequ\":7}]", ""])
  @http://localhost:4444/selenium-server/core/scripts/tellurium-extensions.js:338

{anonymous}(["[\"uid\":\"search.search_project_button\", \"args\":
  [\"jquery=form:group(input[name=q], input[value=Search projects][type=submit],
  input[value=Search the Web][type=submit]) input[value=Search projects]
  [type=submit]\", \"name\":\"click\", \"sequ\":7}]", ""])
  @http://localhost:4444/selenium-server/core/scripts/htmlutils.js:60

{anonymous}([object Object],[object Object])@http://localhost:4444/
  selenium-server/core/scripts/selenium-commandhandlers.js:330

{anonymous}()@http://localhost:4444/selenium-server/core/scripts/
  selenium-executionloop.js:112

{anonymous}(-3)@http://localhost:4444/selenium-server/core/scripts/
  selenium-executionloop.js:78

{anonymous}(-3)@http://localhost:4444/selenium-server/core/scripts/htmlutils.js:60
  at com.thoughtworks.selenium.HttpCommandProcessor.
    throwAssertionFailureExceptionOrNull(HttpCommandProcessor.java:97)
  at com.thoughtworks.selenium.HttpCommandProcessor.

```

```
doCommand(HttpCommandProcessor.java:91)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:597)
```

Chapter 11. Tellurium Maven Archetypes

Tellurium provides three maven archetypes [<http://code.google.com/p/aost/wiki/TelluriumMavenArchetypes>]. For example: tellurium-junit-archetype for Tellurium JUnit test projects, tellurium-testing-archetype for Tellurium TestNG test projects, and tellurium-widget-archetype for Tellurium Widget projects.

Prerequisites

You will need to be running Maven, downloadable from <http://maven.apache.org/download.html> [<http://maven.apache.org/download.html>].

If you never installed Maven before, please follow the official Maven Installation Guide [<http://maven.apache.org/download.html#Installation>].

settings.xml

Here is a sample settings.xml that will allow you to automatically include Tellurium artifacts in your Maven project. This should go in your ~/.m2/settings.xml file:

```
<settings>
  <profiles>
    <profile>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>kungfuters-public-snapshots-repo</id>
          <name>Kungfuters.org Public Snapshot Repository</name>
          <releases>
            <enabled>>false</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
          <url>http://maven.kungfuters.org/content/repositories/snapshots</url>
        </repository>
        <repository>
          <id>kungfuters-public-releases-repo</id>
          <name>Kungfuters.org Public Releases Repository</name>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
          </snapshots>
          <url>http://maven.kungfuters.org/content/repositories/releases</url>
        </repository>
        <repository>
          <id>kungfuters-thirdparty-releases-repo</id>
          <name>Kungfuters.org Third Party Releases Repository</name>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
          </snapshots>
          <url>http://kungfuters.org/nexus/content/repositories/thirdparty</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
</settings>
```

```
</settings>
```

Tellurium JUnit Archetype

Run the following Maven command to create a new JUnit test project:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
  -DarchetypeArtifactId=tellurium-junit-archetype \
  -DarchetypeGroupId=org.telluriumsource \
  -DarchetypeVersion=0.7.0
```

Without adding the Tellurium Maven repository, specify it in the command line as:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
  -DarchetypeArtifactId=tellurium-junit-archetype \
  -DarchetypeGroupId=org.telluriumsource -DarchetypeVersion=0.7.0 \
  -DarchetypeRepository=http://maven.kungfuters.org/content/repositories/releases
```

To create a Tellurium project based on Tellurium 0.8.0 SNAPSHOT, you should use the Maven archetype 0.8.0-SNAPSHOT. To create a JUnit project, use the following Maven command:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
  -DarchetypeArtifactId=tellurium-junit-archetype \
  -DarchetypeGroupId=org.telluriumsource \
  -DarchetypeVersion=0.8.0-SNAPSHOT \
  -DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

After the project is created, you will find the following files

```
|-- README
|--TelluriumConfig.groovy
|-- pom.xml
|-- src
|   |-- main
|   |   |-- groovy
|   |   |-- resources
|   |-- test
|   |   |-- groovy
|   |   |   |-- module
|   |   |   |   |-- GoogleSearchModule.groovy
|   |   |   |   |-- JettyLogonModule.groovy
|   |   |   |-- test
|   |   |   |   |-- GoogleSearchJUnitTestCase.java
|   |   |   |   |-- JettyLogonJUnitTestCase.java
|   |-- resources
```

The GoogleSearchModule UI module is created using the Trump plugin and the GoogleSearchTestCase is an example Tellurium JUnit Test case.

You should use your IDE to open the project, for example, in IntelliJ IDEA, do the following steps

New Project > Import project from external model > Maven > Project directory > Finish

You will open up a Maven project and make sure you are using Groovy 1.7.0 in your project. After that, compile the project and run the example test `GoogleSearchTestCase`.

If you want to run the tests in command line, you can use the following command

```
mvn test
```

Tellurium TestNG Archetype

For a TestNG archetype project, use a different archetype:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-testng-archetype \
-DarchetypeGroupId=org.telluriumsource \
-DarchetypeVersion=0.7.0 \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/releases
```

To create a 0.8.0-SNAPSHOT TestNG project, use the following command:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-testng-archetype \
-DarchetypeGroupId=org.telluriumsource \
-DarchetypeVersion=0.8.0-SNAPSHOT \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

Tellurium Widget Archetype

To create a Tellurium UI widget project, we can use Tellurium Widget archetype as follows,

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-widget-archetype \
-DarchetypeGroupId=org.telluriumsource \
-DarchetypeVersion=0.7.0 \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/releases
```

create a 0.8.0-SNAPSHOT project, use the following command:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-widget-archetype \
-DarchetypeGroupId=org.telluriumsource \
-DarchetypeVersion=0.8.0-SNAPSHOT \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

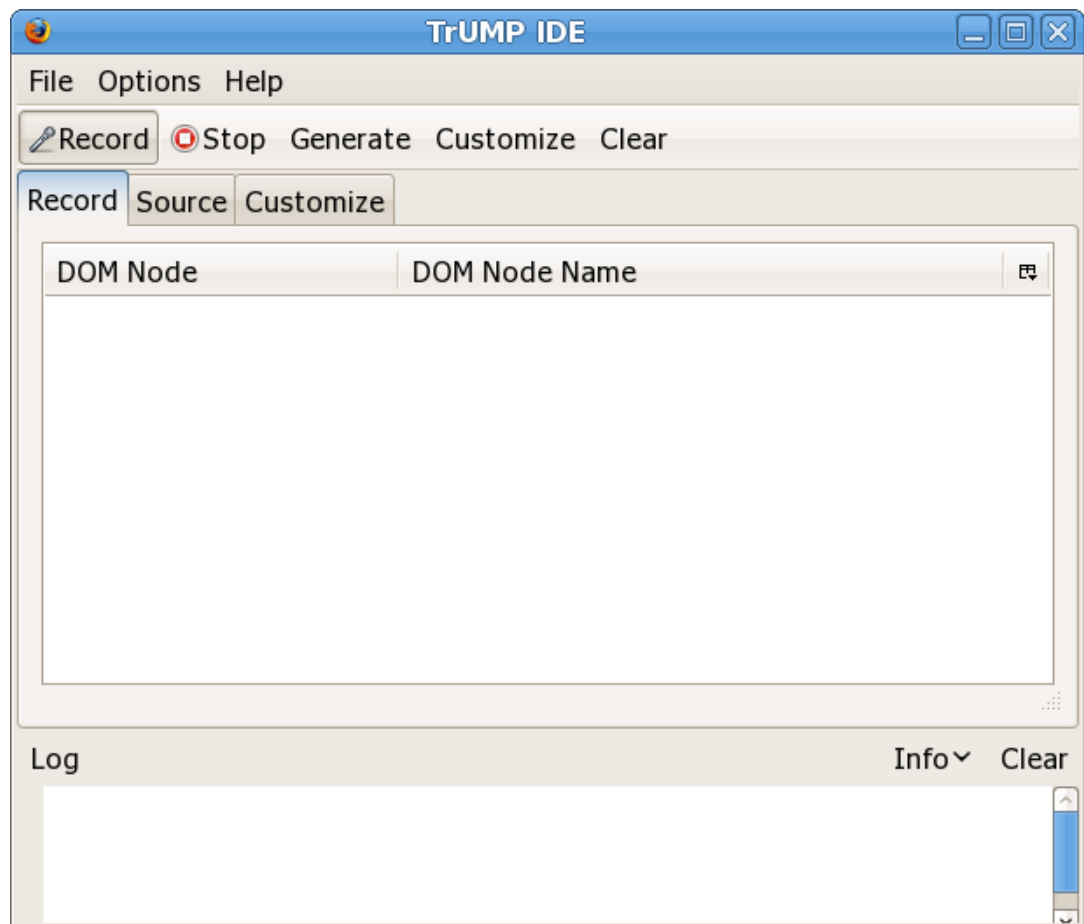
Chapter 12. Tellurium UI Module Firefox Plugin

Install TrUMP

Tellurium UI Model Plugin (TrUMP) is the Firefox plugin to automatically create UI modules for users. Go to the Tellurium project download page and download the TrUMP xpi file or download the Firefox 3 version directly from the Firefox Addons site at:

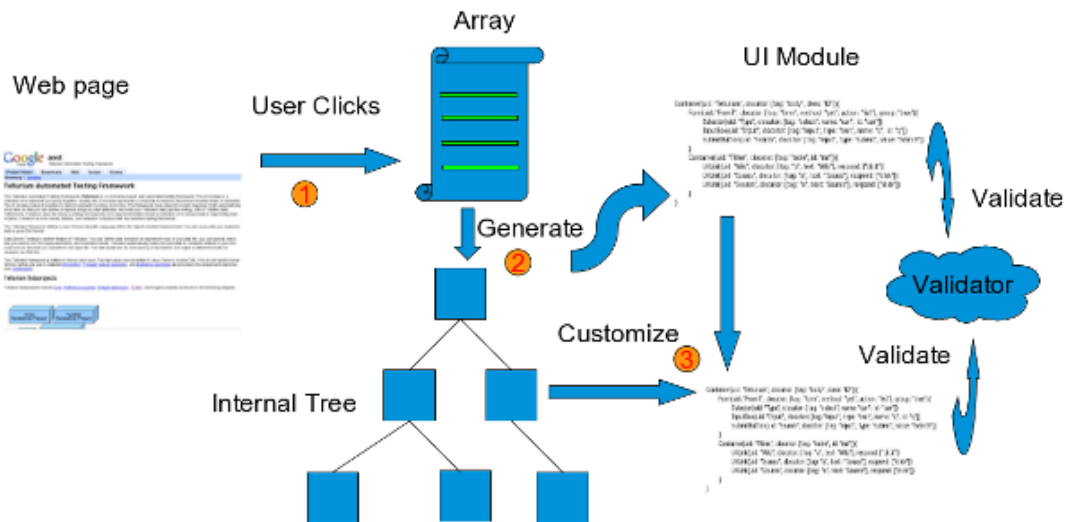
<https://addons.mozilla.org/en-US/firefox/addon/11035> [<https://addons.mozilla.org/en-US/firefox/addon/11035>]

The Tellurium UI Module Plugin (TrUMP) automatically generates UI modules for users. An example of the TrUMP IDE is shown as follows:



TrUMP Workflow

The workflow of TrUMP is shown as follows.



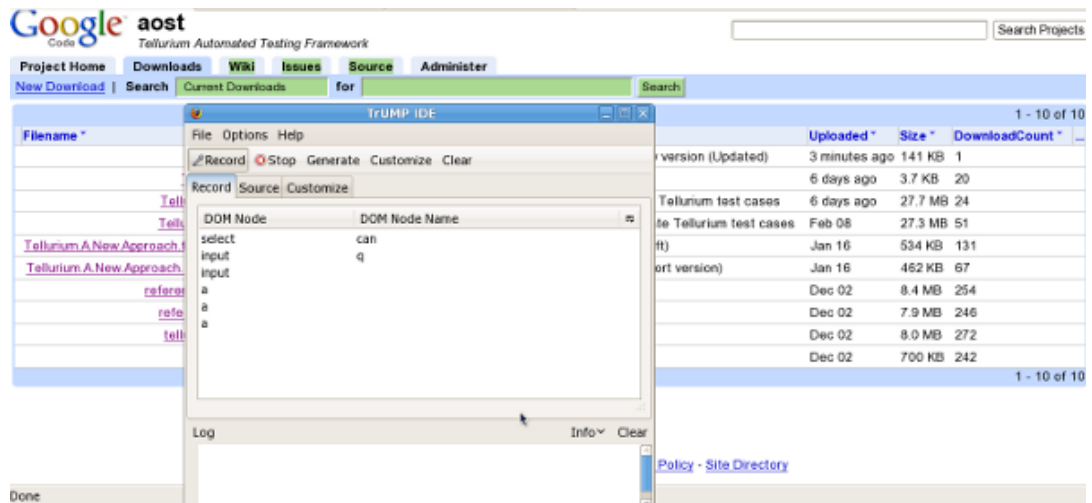
1. Click onto a web page. The corresponding UI element is pushed into an array.

Note: If the element is clicked again, the UI element is removed from the array.

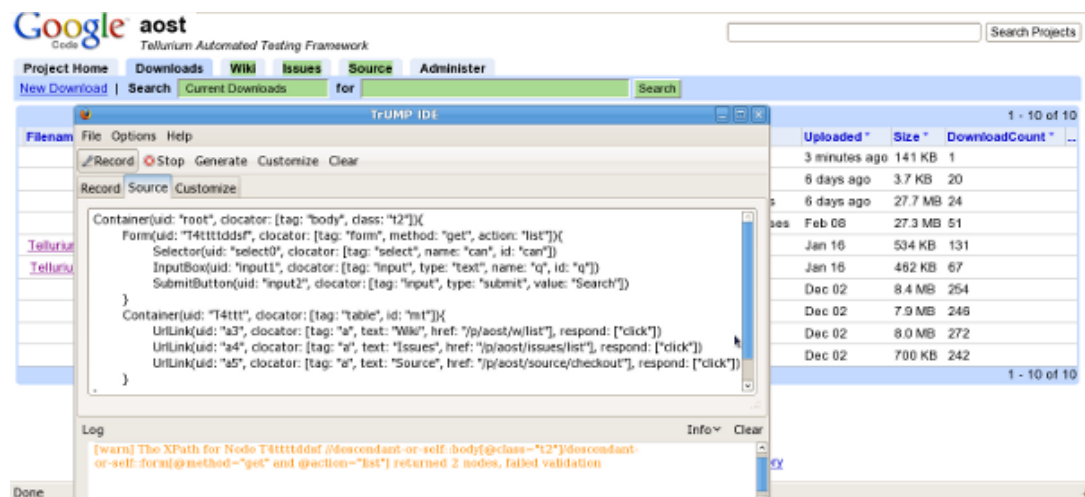
2. Click the "Generate" button. TrUMP implements the following two steps:
 - a. TrUMP generates an internal tree to represent the UI elements using a grouping algorithm. During the tree generating procedure, extra nodes are generated to group UI elements together based on their corresponding location on the DOM tree. The internal tree is very useful and holds all original data that can be used for customization.
 - b. Once the internal tree is built, TrUMP starts the second step, which is to build the default UI module. For each node in the internal tree, TrUMP generates a UI object based on its tag and whether or not it is a parent node.
3. Click the "Customize" button. TrUMP pulls out the original data held in the internal tree and the current attributes utilized by the UI module to create the "Customize" view. When the user clicks on an element, TrUMP lists all available optional attributes in the View for users to customize.
4. TrUMP attempts to validate the UI module automatically whenever a new UI module is generated or updated. TrUMP evaluates each UI element's XPath the same way that Tellurium generates the runtime XPath from the UI module and verifies if the generated runtime XPath is unique in the current web page.
 - a. If it is not unique, a red "X" mark is displayed, and the user should modify the element's attribute to make it disappear.
 - b. If a red "X" is not displayed, the validation is completed. The user can export the generated UI module to a groovy file and start to write Tellurium tests based on the generated UI module.
5. Select "Tools" > "TrUMP IDE" in Firefox. The "Record" button is on by default. Click on "Stop" to stop recording.

How TrUMP Works

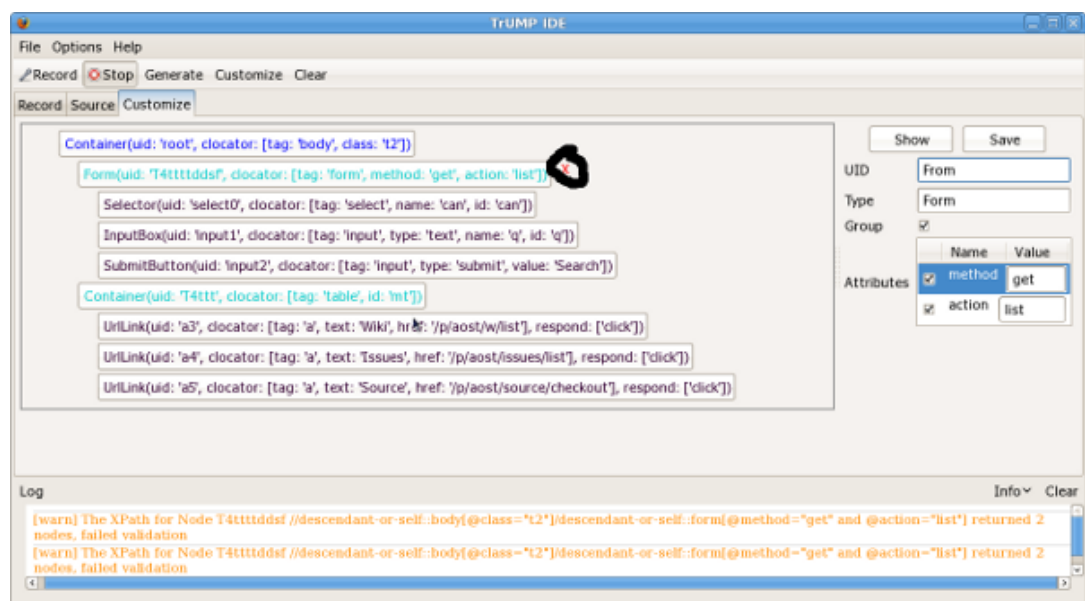
Start to use the TrUMP IDE to record specific UI elements that were selected on the WEB. For example, open the Tellurium Download page and click the search elements and the three links as shown in the following figure:



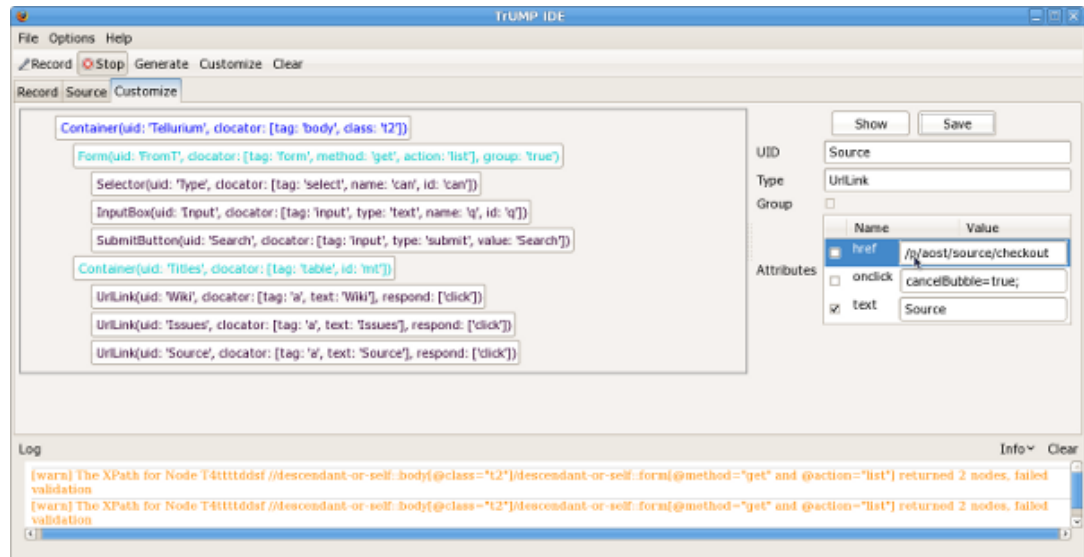
The blue color indicates selected element. Click the selected element again to de-select it. Then, click on the "Generate" button to create the Tellurium UI Module. The user is automatically directed to the "Source" window.



Click the "Customize" button to change the UI module such as UIDs, group locating option, and attributes selected for the UI module.

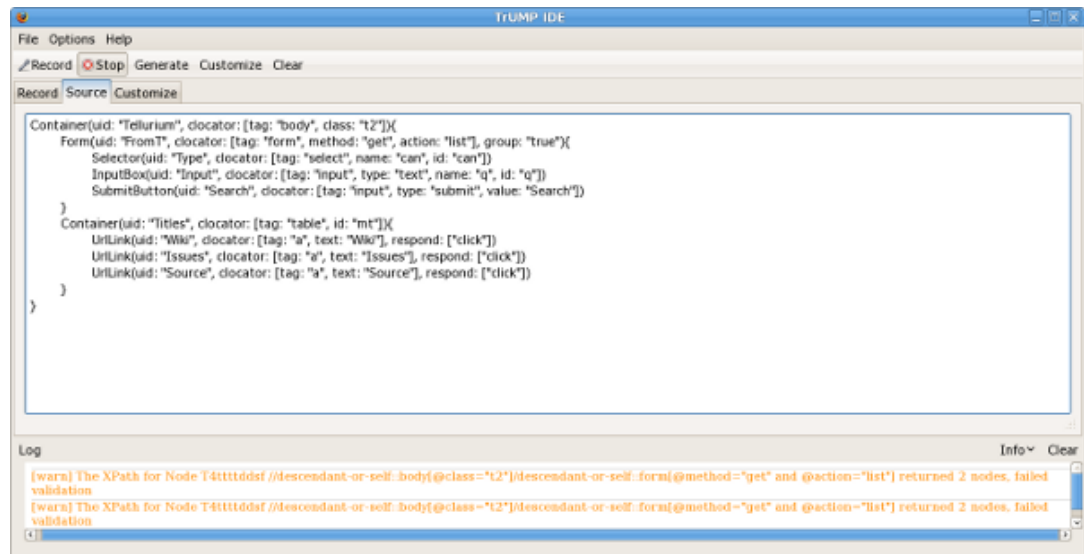


One red "X" mark is displayed, indicating the UI element's XPath is not unique. Select group, or add more attributes to the UI element. The user sees the new customized UI as shown in the following Figure:



Note: The red "X" mark is removed because Tellurium turned on the group locating and the element's xpath is now unique. In the meantime, the UI module in the source tab is automated and updated once the "Save" button is clicked.

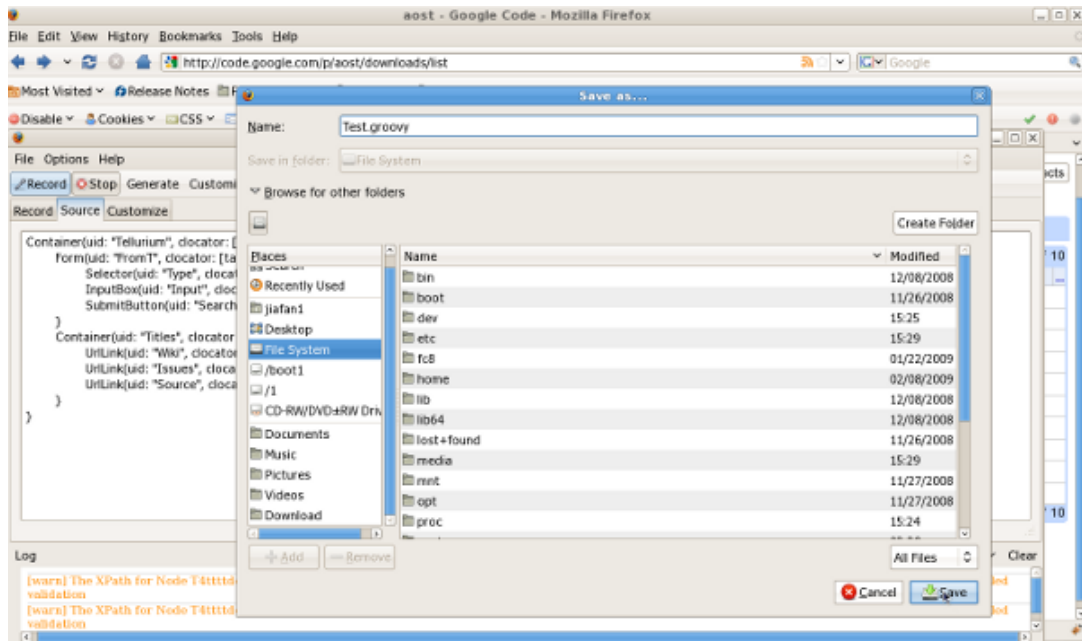
The "Show" button shows the actual Web element on the web for which the UI element is represented.



At this point, export the UI module to a groovy file. Be aware that if any error is seen complaining about the directory, first check the "export directory" in Options > Settings and set it to "C:\\" or other windows directory for the Windows system before you export the file.

For Linux, the user may find there is no "OK" button on the option tab, which is caused by the configure "browser.preferences.instantApply" is set to true by default. Point the firefox to "about:config" and change the option to false.

Once this is completed, the user sees the "OK" button.



Open the groovy file to view the following:

```
package tellurium.ui

import org.tellurium.dsl.DslContext

class NewUiModule extends DslContext{

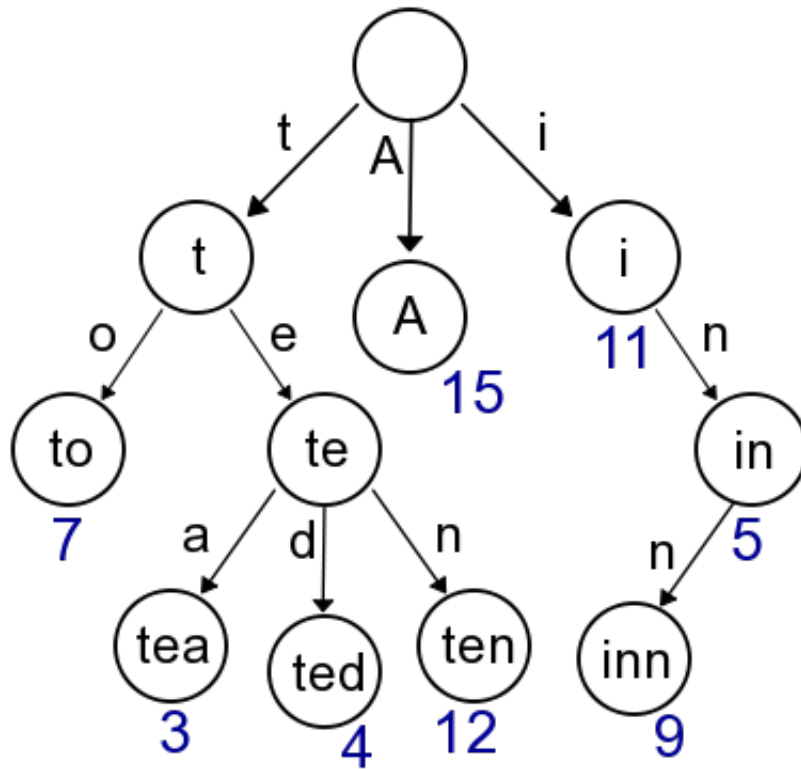
    public void defineUi() {
        ui.Container(uid: "Tellurium", clocator: [tag: "body", class: "t2"]){
            Form(uid: "Form", clocator: [tag: "form", method: "get", action: "list",
                group: "true"])
            {
                Selector(uid: "DownloadType", clocator: [tag: "select", name: "can", id: "can"])
                InputBox(uid: "SearchBox", clocator: [tag: "input", type: "text", name: "q",
                    id: "q"])
                SubmitButton(uid: "Search", clocator: [tag: "input", type: "submit",
                    value: "Search"])
            }
            Container(uid: "Title", clocator: [tag: "table", id: "mt"]){
                UriLink(uid: "Issues", clocator: [tag: "a", text: "Issues"], respond: ["click"])
                UriLink(uid: "Wiki", clocator: [tag: "a", text: "Wiki"], respond: ["click"])
                UriLink(uid: "Downloads", clocator: [tag: "a", text: "Downloads"],
                    respond: ["click"])
            }
        }
    }

    //Add your methods here
}
```

The UI Module Generating Algorithm

You may wonder how the UI module is generated in TrUMP. The main idea behind this is to get the full xpath for each UI element that a user selects. Then, construct the UI module based on the xpaths.

The core algorithm is a similar to the one to build a prefix tree. The data structure of the prefix tree, or Trie in short [<http://en.wikipedia.org/wiki/Trie>] can be illustrated by the following figure, which representing a dictionary with prefix chains.



Usually, a dictionary can be stored in the Trie data structure so that it is very easy to find the words with the same prefix and to do long prefix match [http://en.wikipedia.org/wiki/Longest_prefix_match] .

The UI module generating algorithm is implemented in Javascript. Similarly, we need to first define the node object,

```

function NodeObject(){

    this.constants = {
        TAG : "tag",
        POSITION: "position",
        HEADER : "header",
        TRAILER: "trailer"
    };

    //hold the dom Node associated to the current tree node
    this.domNode = null;
    this.id = null;
    this.xpath = null;
    this.attributes = new HashMap();
    this.parent = null;
    this.children = new Array();
    this.ui = new UiType();

    this.header = null;
    this.tailer = null;
    this.nodexpath = null;

    //flag to indicate whether this node is a new generated
    //during the grouping process, i.e., by the algorithm
    this.newNode = false;
    //tag selection state machine
    this.tagState = new TagState();
    //common methods to process xpath
    this.xpathProcessor = new XPathProcessor();
    //The filter to remove unwanted attributes
    this.filter = new Filter();
    //used to store the element tag
    this.tag = null;
}

```

```
//The UI object associated with this node
this.uiobject = new UiObject();

this.xmlutil = new XmlUtil();
}
```

and some methods.

```
NodeObject.prototype.walkUp = function(){
    var rxp = this.uiobject.buildXPath();

    var xp;

    if(this.parent != null){
        xp = this.parent.walkUp() + rxp;
    }else{
        xp = rxp;
    }

    return xp;
}

NodeObject.prototype.getLevel = function(){
    var level = 0;
    var current = this;

    while(current.parent != null){
        level++;
        current = current.parent;
    }

    return level;
}

NodeObject.prototype.buildUiObject = function(){
    var hasChildren = false;

    if (this.children.length > 0) {
        hasChildren = true;
    }

    this.uiobject.buildUiObject(this, hasChildren);

    if (hasChildren) {
        for (var i = 0; i < this.children.length; i++) {
            this.children[i].buildUiObject();
        }
    }

    this.checkUiDirectAttribute();
}

NodeObject.prototype.printUI = function(layout){
    var hasChildren = false;

    if (this.children.length > 0) {
        hasChildren = true;
    }

    //get the current level of the node so that we can do pretty print
    var level = this.getLevel();

    var strobj = this.uiobject.strUiObject(level);
    layout.push(strobj);

    if (hasChildren) {
        for (var i = 0; i < this.children.length; i++) {
            this.children[i].printUI(layout);
        }
    }

    var strobjft = this.uiobject.strUiObjectFooter(level);
    layout.push(strobjft);
}
```

```
    }  
  }
```

The Trie Tree is defined as follows:

```
function Tree(){  
  this.root = null;  
  this.xpathMatcher = new XPathMatcher();  
  this.uiModel = new Array();  
  this.uid = new Uid();  
  
  //An Array to hold reference to all the UI objects in the Tree  
  //change it to a HashMap so that we can access it by key  
  this.uiObjectMap = null;  
};
```

To do the insertion, we need to implement the tree build process, which is a more complicated than the dictionary one because of the xpath processing.

```
Tree.prototype.addElement = function(element){  
  
  //case I: root is null, insert the first node  
  if (this.root == null) {  
    this.root = new NodeObject();  
    this.root.id = element.uid;  
    this.root.parent = null;  
    this.root.domNode = element.domNode;  
    this.root.xpath = element.xpath;  
    this.root.attributes = element.attributes;  
  } else {  
    //not the first node, need to match element's xpath with current  
    //node's relative xpath starting from the root  
    //First, need to check the root and get the common xpath  
    var common = this.xpathMatcher.match(this.root.xpath, element.xpath);  
  
    var leftover = this.xpathMatcher.remainingXPath(element.xpath, common);  
  
    if (this.root.xpath == common) {  
      //the current node shares the same common xpath as the new node  
      //no extra node need to be added for the current node  
      //then check current node's children  
      if (this.root.children.length == 0) {  
        //no children, so create a new child  
        if (leftover != null && leftover.length > 0) {  
          //only create the child if there are extra xpath  
          var son = new NodeObject();  
          son.id = element.uid;  
          son.xpath = this.xpathMatcher.remainingXPath(element.xpath, common);  
          son.attributes = element.attributes;  
          son.domNode = element.domNode;  
          son.parent = this.root;  
          this.root.addChild(son);  
        }  
      } else {  
        //there are children  
        this.walk(this.root, element.uid, leftover, element.attributes,  
          element.domNode);  
      }  
    } else {  
      var newroot = new NodeObject();  
      newroot.id = "root";  
      newroot.xpath = common;  
      newroot.parent = null;  
      newroot.newNode = true;  
      var newxpath = this.xpathMatcher.remainingXPath(this.root.xpath, common);  
  
      if (this.root.id != null && this.root.id == "root") {  
        this.root.id = this.uid.genUid(newxpath);  
      }  
    }  
  }  
};
```

```

    }
    this.root.xpath = newxpath;
    this.root.parent = newroot;
    newroot.addChild(this.root);

    this.root = newroot;

    if (leftover != null && leftover.length > 0) {
        //only create the child if there are extra xpath
        var child = new NodeObject();
        child.id = element.uid;
        child.xpath = this.xpathMatcher.remainingXPath(element.xpath, common);
        child.attributes = element.attributes;
        child.domNode = element.domNode;
        child.parent = this.root;
        this.root.addChild(child);
    }
}
}
}

Tree.prototype.walk = function(current, uid, xpath, attributes, domnode) {

    if (current.children.length == 0) {
        //there is no children
        if (trimString(xpath).length > 0) {
            //only create the child if there are extra xpath
            var child = new NodeObject();
            child.id = uid;
            child.xpath = xpath;
            child.attributes = attributes;
            child.domNode = domnode;
            child.parent = current;

            current.addChild(child);
        }
    } else {
        var cmp = new Array();
        var maxlen = 0;
        for (var l = 0; l < current.children.length; ++l) {
            var nd = current.children[l];
            var xpt = new XPath();
            xpt.xpath = this.xpathMatcher.match(nd.xpath, xpath);
            xpt.node = nd;
            if (xpt.xpath.length > maxlen) {
                maxlen = xpt.xpath.length;
            }
            cmp.push(xpt);
        }

        //need to handle the situation where there is no common xpath
        if (maxlen == 0) {

            //there is no shared common xpath, add the node directly
            var child = new NodeObject();
            child.id = uid;
            child.xpath = xpath;
            child.attributes = attributes;
            child.domNode = domnode;
            child.parent = current;
            current.addChild(child);
        } else {
            //there are shared common xpath
            var max = new Array();
            for (var m = 0; m < cmp.length; m++) {
                if (cmp[m].xpath.length == maxlen) {
                    max.push(cmp[m])
                }
            }

            var mx = max[0];

            var common = mx.xpath;

            if (mx.node.xpath == common) {

                //The xpath includes the common part, that is to say,
            }
        }
    }
}

```



```

A: "/html/body/table[@id='mt']"
B: "/html/body/table[@id='mt']/tbody/tr/th[3]"
C: "/html/body/table[@id='mt']/tbody/tr/th[3]/div"
D: "/html/body/div[@id='maincol']/div[@id='colcontrol']/div/div[@id='bub']"
E: "/html/body/div[@id='maincol']/div[@id='colcontrol']/div/div[@id='bub']
   /table[@id='resultstable']"
F: "/html/body/div[@id='maincol']/div[@id='colcontrol']/div/div[@id='bub']
   /table[@id='resultstable']/tbody/tr[2]/td[3]/a"

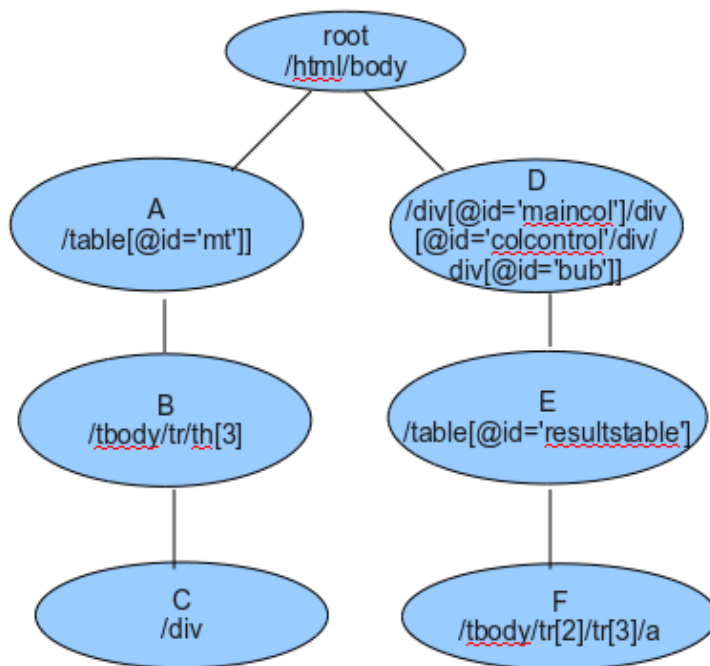
```

Run the algorithm, you will get the following UI module.

```

ui.Container(UID: 'root', clocator: [:]){
  Container(UID: 'A', clocator: [tag:'table']){
    Container(UID: 'B', clocator: [tag:'th']){
      DIV(UID: 'C', clocator: [tag:'div'])
    }
  }
  Container(UID: 'D', clocator: [tag:'div']){
    Container(UID: 'E', clocator: [id:'resultstable',tag:'table']){
      UrlLink(UID: 'F', clocator: [tag:'a'])
    }
  }
}

```



Chapter 13. Tellurium Reference Projects

Introduction

Tellurium 0.7.0 merged the original tellurium-junit-java and tellurium-testng-java two reference projects into a new reference project tellurium-website and add a new reference project ui-examples. The first one is real world tellurium tests using tellurium website as an example and the second one uses html source, mostly contributed by Tellurium users, to demonstrate the usage of different Tellurium UI objects.

Tellurium Website Project

The tellurium website project illustrates the following usages of Tellurium:

- How to create your own UI Objects and wire them into Tellurium core
- How to create UI module files in Groovy
- How to create JUnit or TestNG tellurium testing files in Java
- How to create and run DSL scripts
- How to create Tellurium Data Driven tests

The new code structure is as follows.

```
|-- HowTO
|-- LICENSE.txt
|-- README
|-- TelluriumConfig.groovy
|-- pom.xml
|-- rundsl.bat
|-- rundsl.sh
|-- src
|   |-- main
|   |   |-- groovy
|   |   |   |-- org
|   |   |   |   |-- telluriumsource
|   |   |   |   |   |-- ui
|   |   |   |   |   |   |-- builder
|   |   |   |   |   |   |   |-- SelectMenuBuilder.groovy
|   |   |   |   |   |   |   |-- object
|   |   |   |   |   |   |   |-- SelectMenu.groovy
|   |   |-- resources
|   |-- test
|   |   |-- groovy
|   |   |   |-- org
|   |   |   |   |-- telluriumsource
|   |   |   |   |   |-- ddt
|   |   |   |   |   |   |-- TelluriumIssuesDataDrivenTest.groovy
|   |   |   |   |   |   |-- TelluriumIssuesModule.groovy
|   |   |   |   |   |-- module
|   |   |   |   |   |   |-- TelluriumDownloadsPage.groovy
|   |   |   |   |   |   |-- TelluriumIssuesPage.groovy
|   |   |   |   |   |   |-- TelluriumProjectPage.groovy
|   |   |   |   |   |   |-- TelluriumWikiPage.groovy
|   |   |   |   |-- test
|   |   |   |   |   |-- TelluriumDownloadsPageJUnitTestCase.java
|   |   |   |   |   |-- TelluriumDownloadsPageTestNGTestCase.java
|   |   |   |   |   |-- TelluriumIssuesPageJUnitTestCase.java
|   |   |   |   |   |-- TelluriumIssuesPageTestNGTestCase.java
|   |   |   |   |   |-- TelluriumProjectPageJUnitTestCase.java
|   |   |   |   |   |-- TelluriumProjectPageTestNGTestCase.java
```

Create Custom UI objects

In the tellurium-website project, we defined the custom UI object "SelectMenu" as follows:

For each UI object, you must define the builder so that Tellurium knows how to construct the UI object when it parses the UI modules. For example, we define the builder for the "SelectMenu" object as follows:

195

```

        if(items != null && items.size() > 0){
            menu.addMenuItems(items)
        }

        menu.addTitle(map.get(TITLE))

        return menu
    }
}

```

You may wonder how to hook the custom objects into Tellurium core so that it can recognize the new type. The answer is simple, you just add the UI object name and its builder class name to Tellurium configuration file `TelluriumConfig.groovy`. Update the following section:

```

uiobject{
    builder{
        SelectMenu="org.telluriumsource.ui.builder.SelectMenuBuilder"
    }
}

```

Create UI modules

You should create UI modules in Groovy files, which should extend the `DslContext` class. In the `defineUi` method, define your UIs and then define all methods for them. Take the Tellurium Downloads page as an example:

```

class TelluriumDownloadsPage extends DslContext{

    public void defineUi() {

        //define UI module of a form include download type selector and download search
        ui.Form(uid: "downloadSearch", clocator: [action: "list", method: "GET"], group: "true") {
            Selector(uid: "downloadType", clocator: [name: "can", id: "can"])
            TextBox(uid: "searchLabel", clocator: [tag: "span", text: "for"])
            InputBox(uid: "searchBox", clocator: [type: "text", name: "q"])
            SubmitButton(uid: "searchButton", clocator: [value: "Search"])
        }

        ui.Table(uid: "downloadResult", clocator: [id: "resultstable", class: "results"], group: "true"){
            //define table header
            //for the border column
            TextBox(uid: "{header: 1}", clocator: [:])
            UrlLink(uid: "{header: 2}", clocator: [text: "**Filename"])
            UrlLink(uid: "{header: 3}", clocator: [text: "**Summary + Labels"])
            UrlLink(uid: "{header: 4}", clocator: [text: "**Uploaded"])
            UrlLink(uid: "{header: 5}", clocator: [text: "**Size"])

            UrlLink(uid: "{header: 6}", clocator: [text: "**DownloadCount"])
            UrlLink(uid: "{header: 7}", clocator: [text: "**..."])

            //define table elements
            //for the border column
            TextBox(uid: "{row: all, column: 1}", clocator: [:])

            //the summary + labels column consists of a list of UrlLinks
            List(uid: "{row: all, column: 3}") {
                UrlLink(uid: "{all}", clocator: [:])
            }
            //For the rest, just UrlLink
            UrlLink(uid: "{all}", clocator: [:])
        }

        ui.RadioButton(uid: "test", clocator: [:], respond: "click")
    }

    public String[] getAllDownloadTypes(){
        return getSelectOptions("downloadSearch.downloadType")
    }
}

```

```
    }

    public String getCurrentDownloadType(){
        return getSelectedLabel("downloadSearch.downloadType");
    }

    public void selectDownloadType(String type){
        selectByLabel "downloadSearch.downloadType", type
    }

    public void searchDownload(String keyword){
        type "downloadSearch.searchBox", keyword
        click "downloadSearch.searchButton"
        waitForPageToLoad 30000
    }

    public int getTableHeaderNum(){
        enableCache();
        enableTelluriumApi();
        return getTableHeaderColumnNum("downloadResult")
    }

    public List<String> getHeaderNames(){
        enableCache();
        enableTelluriumApi();

        List<String> headernames = new ArrayList<String>()
        int mcolumn = getTableHeaderColumnNum("downloadResult")
        for(int i=1; i<=mcolumn; i++){
            headernames.add(getText("downloadResult.header[${i}]"))
        }

        return headernames
    }

    public List<String> getDownloadFileNames(){

        int mcolumn = getTableMaxRowNum("downloadResult")
        List<String> filenames = new ArrayList<String>()
        for(int i=1; i<=mcolumn; i++){
            filenames.add(getText("downloadResult[${i}][2]").trim())
        }

        return filenames
    }

    public void clickFileNameColumn(int row){
        click "downloadResult[${row}][2]"
        pause 1000
        chooseCancelOnNextConfirmation()
        pause 500
    }

    ...
}
```

Create Java Test Cases

You can create Java Test Cases by extending the `TelluriumJUnitTestCase` or `TelluriumTestNGTestCase` class. Nothing special, just like regular JUnit test cases. For instance,

```
public class TelluriumDownloadsPageTestNGTestCase extends TelluriumTestNGTestCase{
    private static TelluriumDownloadsPage downloadPage;

    @BeforeClass
    public static void initUi() {
        downloadPage = new TelluriumDownloadsPage();
        downloadPage.defineUi();
        connectSeleniumServer();
        useCache(true);
    }
}
```

```

@BeforeMethod
public void setUpForMethod(){
    connectUrl("http://code.google.com/p/aost/downloads/list");
}

@Test
public void testValidate(){
    downloadPage.validate("downloadResult");
}

@Test
public void testDownloadTypes(){
    String[] allTypes = downloadPage.getAllDownloadTypes();
    assertNotNull(allTypes);
    assertTrue(allTypes[1].contains("All downloads"));
    assertTrue(allTypes[2].contains("Featured downloads"));
    assertTrue(allTypes[3].contains("Current downloads"));
    assertTrue(allTypes[4].contains("Deprecated downloads"));
}

@Test
public void testDefaultDownloadType(){
    // Set download type with other value
    downloadPage.selectDownloadType(" All downloads");

    // Navigate away from download page
    connectUrl("http://code.google.com/p/aost/downloads/list");
    String defaultType = downloadPage.getCurrentDownloadType();
    assertNotNull(defaultType);
    assertTrue(defaultType.contains("Current downloads"));
}

@Test
public void testSearchByText(){
    // Set download type with other value
    downloadPage.selectDownloadType(" All downloads");
    downloadPage.searchDownload("Tellurium-0.6.0");

    useTelluriumApi(true);
    List<String> list = downloadPage.getDownloadFileNames();
    useTelluriumApi(false);
    assertNotNull(list);
    assertFalse(list.isEmpty());
    assertTrue(Helper.include(list, "tellurium-core.0.6.0.tar.gz"));
}

@Test
public void testSearchByLabel(){
    // Set download type with other value
    downloadPage.selectDownloadType(" All downloads");
    downloadPage.searchDownload("label:Featured");

    useTelluriumApi(true);
    List<String> list = downloadPage.getDownloadFileNames();
    useTelluriumApi(false);
    assertNotNull(list);
    assertFalse(list.isEmpty());
}

@Test
public void testDownloadFileNames(){
    int mcolumn = downloadPage.getTableHeaderNum();
    assertEquals(7, mcolumn);
    List<String> list = downloadPage.getHeaderNames();
    assertNotNull(list);
    assertEquals(7, list.size());
    assertTrue(Helper.include(list, "Filename"));
    list = downloadPage.getDownloadFileNames();
    assertNotNull(list);
    assertFalse(list.isEmpty());
    assertTrue(Helper.include(list, "tellurium-core.0.6.0.tar.gz"));
}

.....
}

```

Create and Run DSL Scripts

In Tellurium, you can create test scripts in pure DSL. Take TelluriumPage.dsl as an example.

```
//define Tellurium project menu
ui.Container(uid: "menu", clocator: [tag: "table", id: "mt"], group: "true"){
    //since the actual text is Project Home, we can use partial match
    //here. Note "*" stands for a partial match
    UrlLink(uid: "project_home", clocator: [text: "*Home"])
    UrlLink(uid: "downloads", clocator: [text: "Downloads"])
    UrlLink(uid: "wiki", clocator: [text: "Wiki"])
    UrlLink(uid: "issues", clocator: [text: "Issues"])
    UrlLink(uid: "source", clocator: [text: "Source"])
}

//define the Tellurium project search module, which includes an input box, two search buttons
ui.Form(uid: "search", clocator: [:], group: "true"){
    InputBox(uid: "searchbox", clocator: [name: "q"])
    SubmitButton(uid: "search_project_button", clocator: [value: "Search projects"])
}

openUrl "http://code.google.com/p/aost/"
click "menu.project_home"
waitForPageToLoad 30000
click "menu.downloads"
waitForPageToLoad 30000
click "menu.wiki"
waitForPageToLoad 30000
click "menu.issues"
waitForPageToLoad 30000

openUrl "http://code.google.com/p/aost/"
type "search.searchbox", "Tellurium Selenium groovy"
click "search.search_project_button"
waitForPageToLoad 30000
```

To run the DSL script, you should run the code with Maven

```
mvn test
```

Then, use the rundsl.sh to run the DSL script. For Windows, you should use the rundsl.bat script.

In addition, Tellurium provides a new rundsl.groovy script using the Groovy grape feature.

```
import groovy.grape.Grape;

Grape.grab(group:'org.telluriumsource', module:'tellurium-core', version:'0.7.0-SNAPSHOT',
    classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.stringtree', module:'stringtree-json', version:'2.0.10',
    classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'caja', module:'json_simple', version:'r1',
    classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.seleniumhq.selenium.server', module:'selenium-server',
    version:'1.0.1-te3-SNAPSHOT', classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.seleniumhq.selenium.client-drivers', module:'selenium-java-client-driver',
    version:'1.0.1', classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.apache.poi', module:'poi', version:'3.0.1-FINAL',
    classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'junit', module:'junit', version:'4.7',
    classLoader:this.class.classLoader.rootLoader)

import org.telluriumsource.dsl.DslScriptExecutor

@Grapes([
```

```

@Grab(group='org.codehaus.groovy', module='groovy-all', version='1.7.0'),
@Grab(group='org.seleniumhq.selenium.server', module='selenium-server',
version='1.0.1-te3-SNAPSHOT'),
@Grab(group='org.seleniumhq.selenium.client-drivers', module='selenium-java-client-driver',
version='1.0.1'),
@Grab(group='junit', module='junit', version='4.7'),
@Grab(group='caja', module='json_simple', version='r1'),
@Grab(group='org.apache.poi', module='poi', version='3.0.1-FINAL'),
@Grab(group='org.stringtree', module='stringtree-json', version='2.0.10'),
@Grab(group='org.telluriumsource', module='tellurium-core', version='0.7.0-SNAPSHOT')
])

def runDsl(String[] args) {
    def cli = new CliBuilder(usage: 'rundsdl.groovy -[hf] [scriptname]')
    cli.with {
        h longOpt: 'help', 'Show usage information'
        f longOpt: 'scriptname', 'DSL script name'
    }
    def options = cli.parse(args)
    if (!options) {
        return
    }
    if (options.h) {
        cli.usage()
        return
    }
    if (options.f) {
        def extraArguments = options.arguments()
        if (extraArguments) {
            extraArguments.each {String name ->
                def input = [name].toArray(new String[0])
                DslScriptExecutor.main(input)
            }
        }
    }
}

println "Running DSL test script, press Ctrl+C to stop."

runDsl(args)

```

Before use the script, you need to configure Groovy Grape [http://groovy.codehaus.org/Grape]. Put the following grapeConfig.xml file into your home/.groovy/.

```

<ivysettings>
<settings defaultResolver="downloadGrapes"/>
<property
name="local-maven2-pattern"
value="${user.home}/.m2/repository/[organisation]/[module]/[revision]/
[module]-[revision](-[classifier]).[ext]"
override="false" />
<resolvers>
<chain name="downloadGrapes">
<filesystem name="cachedGrapes">
<ivy pattern="${user.home}/.groovy/grapes/[organisation]/[module]/ivy-[revision].xml"/>
<artifact pattern="${user.home}/.groovy/grapes/[organisation]/[module]/[type]s/[
artifact]-[revision].[ext]"/>
</filesystem>
<filesystem name="local-maven-2" m2compatible="true" local="true">
<ivy pattern="${local-maven2-pattern}"/>
<artifact pattern="${local-maven2-pattern}"/>
</filesystem>
<!-- todo add 'endorsed groovy extensions' resolver here -->
<ibiblio name="kungfuters.3rdparty"
root="http://maven.kungfuters.org/content/repositories/thirdparty/"
m2compatible="true"/>
<ibiblio name="codehaus" root="http://repository.codehaus.org/"
m2compatible="true"/>
<ibiblio name="ibiblio" m2compatible="true"/>
<ibiblio name="java.net2" root="http://download.java.net/maven/2/"
m2compatible="true"/>
<ibiblio name="openqa" root="http://archiva.openqa.org/repository/releases/"
m2compatible="true"/>
<ibiblio name="kungfuters.snapshot"
root="http://maven.kungfuters.org/content/repositories/snapshots/"
m2compatible="true"/>
<ibiblio name="kungfuters.release"
root="http://maven.kungfuters.org/content/repositories/releases/"
m2compatible="true"/>
</chain>

```



```
</resolvers>
</ivysettings>
```

Then run the following command:

```
groovy rundsl.groovy -f DSL_script_name
```

If you are behind a firewall, you can use the http proxy as follow.

```
groovy -Dhttp.proxyHost=proxy_host -Dhttp.proxyPort=proxy_port \
rundsl.groovy -f DSL_script_name
```

If you defined custom UI objects in your project, you should first build the jar artifact, and install it to your local Maven repo. Then update the rundsl.groovy file to add the new dependency. For example, in the tellurium-website reference project, we defined custom UI objects, we added the following two lines into the rundsl.groovy script.

```
Grape.grab(group:'org.telluriumsource', module:'tellurium-website', version:'0.7.0-SNAPSHOT',
classLoader:this.class.classLoader.rootLoader)

...

@Grab(group='org.telluriumsource', module='tellurium-website', version='0.7.0-SNAPSHOT')
```

Data Driven Testing

We use Tellurium Issue page as the data driven testing example, we define tests to search issues assigned to a Tellurium team member and use the input file to define which team members we want the result for.

We first define a TelluriumIssuesModule class that extends TelluriumDataDrivenModule class and includes a method "defineModule". In the "defineModule" method, we define UI modules, input data format, and different tests. The UI modules are the same as defined before for the Tellurium issue page.

The input data format is defined as

```
fs.FieldSet(name: "OpenIssuesPage") {
    Test(value: "OpenTelluriumIssuesPage")
}

fs.FieldSet(name: "IssueForOwner", description: "Data format for test SearchIssueForOwner") {
    Test(value: "SearchIssueForOwner")
    Field(name: "issueType", description: "Issue Type")
    Field(name: "owner", description: "Owner")
}
```

Here we have two different input data formats. The "Test" field defines the test name and the "Field" field define the input data name and description. For example, the input data for the test "SearchIssueForOwner" have two input parameters "issueType" and "owner".

The tests are defined use "defineTest". One of the test "SearchIssueForOwner" is defined as follows,

```

defineTest("SearchIssueForOwner") {
    String issueType = bind("IssueForOwner.issueType")
    String issueOwner = bind("IssueForOwner.owner")
    int headernum = getCacheVariable("headernum")
    int expectedHeaderNum = getTableHeaderNum()
    compareResult(expectedHeaderNum, headernum)

    List<String> headernames = getCacheVariable("headernames")
    String[] issueTypes = getCacheVariable("issuetypes")
    String issueTypeLabel = getIssueTypeLabel(issueTypes, issueType)
    checkResult(issueTypeLabel) {
        assertTrue(issueTypeLabel != null)
    }
    //select issue type
    if (issueTypeLabel != null) {
        selectIssueType(issueTypeLabel)
    }
    //search for all owners
    if ("all".equalsIgnoreCase(issueOwner.trim())) {
        searchForAllIssues()
    } else {
        searchIssue("owner:" + issueOwner)
    }
    .....
}

```

As you can see, we use "bind" to tie the variable to input data field. For example, the variable "issueType" is bound to "IssueForOwner.issueType", i.e., field "issueType" of the input Fieldset "IssueForOwner". "getCacheVariable" is used to get variables passed from previous tests and "compareResult" is used to compare the actual result with the expected result.

input file format looks like:

```

OpenTelluriumIssuesPage
## Test Name | Issue Type | Owner
SearchIssueForOwner|Open| all
SearchIssueForOwner|All| matt.senter
SearchIssueForOwner|Open| John.Jian.Fang
SearchIssueForOwner|All| vivekmongolu
SearchIssueForOwner|All| haroonzone

```

The actual test class is very simple

```

class TelluriumIssuesDataDrivenTest extends TelluriumDataDrivenTest{

    public void testDataDriven() {

        includeModule org.telluriumsource.ddt.TelluriumIssuesModule.class

        //load file
        loadData "src/test/resources/org/telluriumsource/data/TelluriumIssuesInput.txt"

        useCache(true);
        useClosestMatch(true);

        //read each line and run the test script until the end of the file
        stepToEnd()

        //close file
        closeData()

    }
}

```

We first define which Data Driven Module we want to load and then read input data file. After that, we read the input data file line by line and execute the appropriate tests defined in the input file. Finally, close the data file.

The test result looks as follows.

```
<TestResults>
  <Total>6</Total>
  <Succeeded>6</Succeeded>
  <Failed>0</Failed>
  <Test name='OpenTelluriumIssuesPage'>
    <Step>1</Step>
    <Passed>true</Passed>
    <Input>
      <test>OpenTelluriumIssuesPage</test>
    </Input>
    <Assertion Value='10' Passed='true' />
    <Status>PROCEEDED</Status>
    <Runtime>2.579049</Runtime>
  </Test>
  <Test name='SearchIssueForOwner'>
    <Step>2</Step>
    <Passed>true</Passed>
    <Input>
      <test>SearchIssueForOwner</test>
      <issueType>Open</issueType>
      <owner>all</owner>
    </Input>
    <Assertion Expected='10' Actual='10' Passed='true' />
    <Assertion Value=' Open Issues' Passed='true' />
    <Status>PROCEEDED</Status>
    <Runtime>4.118923</Runtime>
    <Message>Found 10 Open Issues for owner all</Message>
    <Message>Issue: Better way to wait or pause during testing</Message>
    <Message>Issue: Add support for JQuery selector</Message>
    <Message>Issue: Export Tellurium to Ruby</Message>
    <Message>Issue: Add check Alter function to Tellurium</Message>
    <Message>Issue: Firefox plugin to automatically generate UI module for users</Message>
    <Message>Issue: Create a prototype for container-like Dojo widgets</Message>
    <Message>Issue: Need to create Wiki page to explain how to setup Maven and use
      Maven to build multiple projects</Message>
    <Message>Issue: Configure IntelliJ to properly load one maven sub-project and not look
      for an IntelliJ project dependency</Message>
    <Message>Issue: Support nested properties for Tellurium</Message>
    <Message>Issue: update versions for extensioin dojo-widget and TrUMP projects</Message>
  </Test>
  ...
</TestResults>
```

Tellurium ui-examples Project

Some of the html sources in the ui-examples project are contributed by tellurium users. Thanks for their contributions. ui-examples project includes the following different types of UI objects.

Selector

HTML source:

```
<form method="POST" action="check_phone">
  <select name="Profile/Customer/Telephone/@CountryAccessCode" style="font-size:92%">
    <option value="1" selected=selected>US</option>
    <option value="2" id="uk">UK</option>
    <option value="3">AT</option>
```

```

        <option value="4">BE</option>
        <option value="4" id="ca">CA</option>
        <option value="6">CN</option>
        <option value="7">ES</option>
        <option value="8">VG</option>
    </select>
    <input type="text" class="medium paxFerryNameInput" value=""
        name="Profile/Customer/Telephone/@PhoneNumber"
        maxlength="16" id="phone1" tabindex="26">
    <input name="submit" type="submit" value="Check">
</form>

```

UI Module:

```

class SelectorExampleModule extends DslContext{

    public void defineUi(){
        ui.Form(uid: "Form", clocator: [method: "POST", action: "check_phone"]){
            Selector(uid: "Country", clocator: [name: "\$CountryAccessCode"])
            InputBox(uid: "Number", clocator: [name: "\$PhoneNumber"])
            SubmitButton(uid: "check", clocator: [value: "Check"])
        }
    }

    public void check(String country, String number){
        select "Form.Country", country
        keyType "Form.Number", number
        click "Form.check"
        waitForPageToLoad 30000
    }
}

```

Container

HTML source:

```

<div class="mainMenu">
    <a href="http://localhost:8080/ContainerExample.html">Events</a>
    <a href="http://localhost:8080/ContainerExample.html">Suppliers</a>
    <a href="http://localhost:8080/ContainerExample.html">Venues</a>
    <a href="http://localhost:8080/ContainerExample.html">Booking Report</a>
    <a href="http://localhost:8080/ContainerExample.html">Notifications</a>
    <a href="http://localhost:8080/ContainerExample.html">Help</a>
</div>

```

UI Module:

```

class ContainerExampleModule extends DslContext {

    public void defineUi(){

        ui.Container(uid: "mainnav", clocator: [tag: "div", class: "mainMenu"], group: true) {
            UrlLink(uid: "events", clocator: [text: "Events"])
            UrlLink(uid: "suppliers", clocator: [text: "Suppliers"])
            UrlLink(uid: "venues", clocator: [text: "Venues"])
            UrlLink(uid: "bookingReport", clocator: [text: "Booking Report"])
            UrlLink(uid: "notifications", clocator: [text: "Notifications"])
            UrlLink(uid: "help", clocator: [text: "Help"])
        }
    }
}

```

Form

HTML source:

```
<H1>FORM Authentication demo</H1>

<form method="POST" action="j_security_check">
  <table border="0" cellspacing="2" cellpadding="1">
    <tr>
      <td>Username:</td>
      <td><input size="12" value="" name="j_username" maxlength="25" type="text"></td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input size="12" value="" name="j_password" maxlength="25" type="password"></td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <input name="submit" type="submit" value="Login">
      </td>
    </tr>
  </table>
</form>
```

UI Module:

```
public class FormExampleModule extends DslContext {

  public void defineUi() {
    ui.Form(uid: "Form", clocator: [tag: "table"]){
      Container(uid: "Username", clocator: [tag: "tr"]){
        TextBox(uid: "Label", clocator: [tag: "td", text: "Username:", direct: "true"])
        InputBox(uid: "Input", clocator: [tag: "input", type: "text", name: "j_username"])
      }
      Container(uid: "Password", clocator: [tag: "tr"]){
        TextBox(uid: "Label", clocator: [tag: "td", text: "Password:", direct: "true"])
        InputBox(uid: "Input", clocator: [tag: "input", type: "password", name: "j_password"])
      }
      SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit", value: "Login",
        name: "submit"])
    }
  }

  public void logon(String username, String password){
    keyType "Form.Username.Input", username
    keyType "Form.Password.Input", password
    click "Form.Submit"
    waitForPageToLoad 30000
  }
}
```

List

HTML source:

```
<table id="hp_table" cellspacing="0" cellpadding="0">
  <tbody>
    <tr>
      <td class="sidebar" valign="top">
        <DIV class="sub_cat_section">
          <DIV class="sub_cat_title" style="">Fiction</DIV>
```

```

        <P><A href="" style="">Literature</A></P>

        <P><A href="" style="">Science fiction</A></P>

        <P><A href="" style="">Fantasy</A></P>

        <P><A href="" style="">Romance</A></P>

        <P><A href="" style="">Mystery</A></P>

        <P><A href="" style="">Fairy tales</A></P>

        <P><A href="" style="">Short stories</A></P>

        <P><A href="" style="">Poetry</A></P></DIV>
<DIV class="sub_cat_section">
  <DIV class="sub_cat_title">Non-fiction</DIV>
  <P><A href="">Philosophy</A>
  </P>

  <P><A href="">Economics</A>
  </P>

  <P><A href="">Political science</A></P>

  <P><A href="">Linguistics</A>
  </P>

  <P><A href="">Mathematics</A>
  </P>

  <P><A href="">Physics</A>
  </P>

  <P><A href="">Chemistry</A>
  </P>

  <P><A href="">Biology</A>
  </P></DIV>
<DIV class="sub_cat_section">
  <DIV class="sub_cat_title">Random subjects</DIV>
  <P><A href="">Toys</A>
  </P>

  <P><A href="">Solar houses</A></P>

  <P><A href="">Stories in rhyme</A></P>

  <P><A href="">Courtship</A>
  </P>

  <P><A href="">Mythology</A>
  </P>

  <P><A href="">Differential equations</A></P>

  <P><A href="">Latin inscriptions</A></P>

  <P><A href="">Nuclear energy</A></P>
</DIV>
</td>
</tr>
</tbody>
</table>

```

UI Module:

```

class ListExampleModule extends DslContext {

  public void defineUi() {
    ui.Container(uid: "GoogleBooksList", clocator: [tag: "table", id: "hp_table"]) {
      List(uid: "subcategory", clocator: [tag: "td", class: "sidebar"], separator: "div") {
        Container(uid: "{all}") {

```

Table

Repeat

HTML source:

```
<form name="selectedSailingsForm">
  <div class="segment clearfix">
    <div class="option">
      <ul class="fares">
        <li>
          <input type="radio">&nbsp;
          <label>Economy</label>
        </li>
        <li>
          <input type="radio">&nbsp;
          <label>Flexible</label>
        </li>
      </ul>
      <div class="details">
        <dl>
          <dt>Ship:</dt>
          <dd>A</dd>
          <dt>Departs</dt>
          <dd>
            <em>08:00</em>
          </dd>
          <dt>Arrives</dt>
          <dd>
            <em>11:45</em>
          </dd>
        </dl>
      </div>
    </div>
    <div class="option">
      <ul class="fares">
        <li>
          <input type="radio">&nbsp;
          <label>Economy</label>
        </li>
        <li>
          <input type="radio">&nbsp;
          <label>Flexible</label>
        </li>
      </ul>
      <div class="details">
        <dl>
          <dt>Ship:</dt>
          <dd>B</dd>
          <dt>Departs</dt>
          <dd>
            <em>17:30</em>
          </dd>
          <dt>Arrives</dt>
          <dd>
            <em>21:15</em>
          </dd>
        </dl>
      </div>
    </div>
  </div>
  <div class="segment clearfix">
    <div class="option">
      <ul class="fares">
        <li>
          <input type="radio">&nbsp;
          <label>Economy</label>
        </li>
        <li>
          <input type="radio">&nbsp;
          <label>Flexible</label>
        </li>
      </ul>
      <div class="details">
        <div class="photo"><img/></div>
        <dl>
```



```

        <dt>Ship:</dt>
        <dd>C</dd>
        <dt>Departs</dt>
        <dd>
            <em>02:00</em>
        </dd>
        <dt>Arrives</dt>
        <dd>
            <em>06:00</em>
        </dd>
    </dl>
</div>
</div>
<div class="option">
    <ul class="fares">
        <li>
            <input type="radio">&nbsp;
            <label>Economy</label>
        </li>
        <li>
            <input type="radio">&nbsp;
            <label>Flexible</label>
        </li>
    </ul>
    <div class="details">
        <dl>
            <dt>Ship:</dt>
            <dd>D</dd>
            <dt>Departs</dt>
            <dd>
                <em>12:45</em>
            </dd>
            <dt>Arrives</dt>
            <dd>
                <em>16:30</em>
            </dd>
        </dl>
    </div>
</div>
</div>
</form>

```

UI Module:

```

class RepeatExampleModule extends DslContext {

    public void defineUi(){

        ui.Form(uid: "SailingForm", clocator: [name: "selectedSailingsForm"] ){
            Repeat(uid: "Section", clocator: [tag: "div", class: "segment clearfix"]){
                Repeat(uid: "Option", clocator: [tag: "div", class: "option", direct: "true"]){
                    List(uid: "Fares", clocator: [tag: "ul", class: "fares", direct: "true"],
                        separator: "li")
                    {
                        Container(uid: "{all}") {
                            RadioButton(uid: "radio", clocator: [:], respond: ["click"])
                            TextBox(uid: "label", clocator: [tag: "label"])
                        }
                    }
                }
                Container(uid: "Details", clocator: [tag: "div", class: "details"]){
                    Container(uid: "ShipInfo", clocator: [tag: "dl"]){
                        TextBox(uid: "ShipLabel", clocator: [tag: "dt", position: "1"])
                        TextBox(uid: "Ship", clocator: [tag: "dd", position: "1"])
                        TextBox(uid: "DepartureLabel", clocator: [tag: "dt", position: "2"])
                        Container(uid: "Departure", clocator: [tag: "dd", position: "2"]){
                            TextBox(uid: "Time", clocator: [tag: "em"])
                        }
                        TextBox(uid: "ArrivalLabel", clocator: [tag: "dt", position: "3"])
                        Container(uid: "Arrival", clocator: [tag: "dd", position: "3"]){
                            TextBox(uid: "Time", clocator: [tag: "em"])
                        }
                    }
                }
            }
        }
    }
}

```

}
}
}
}

Chapter 14. Tellurium Reference

Tellurium 0.7.0 starts to use docbook [<http://docbook.sourceforge.net/>] to build the reference document. What is docbook? We quote from <http://docbook.sourceforge.net/>, "DocBook is an XML vocabulary that lets you create documents in a presentation-neutral form that captures the logical structure of your content. Using free tools along with the DocBook XSL stylesheets, you can publish your content as HTML pages and PDF files, and in many other formats."

Maven

docbkx tools [<http://code.google.com/p/docbkx-tools/>] provides a number of tools supporting DocBook in a Maven environment including the Docbkx Maven Plugin [<http://docs.codehaus.org/display/MAVENUSER/Docbkx+Maven+Plugin>]. The good news is that the Docbkx Maven plugin supports DocBook version 5.0 [<http://www.docbook.org/tdg5/en/html/docbook.html>].

The Maven pom is as follows.

```
<build>
  <plugins>
    <plugin>
      <groupId>com.agilejava.docbkx</groupId>
      <artifactId>docbkx-maven-plugin</artifactId>
      <version>2.0.10-SNAPSHOT</version>
      <executions>
        <execution>
          <id>documentation identifier</id>
          <phase>pre-site</phase>
          <goals>
            <goal>generate-pdf</goal>
          </goals>
          <configuration>
            <!-- per execution configuration -->
            <includes>tellurium-reference.xml</includes>
            <!--<draftMode>yes</draftMode>-->
          </configuration>
        </execution>
      </executions>
      <configuration>
        <!-- shared configuration -->
        <generatedSourceDirectory>${project.build.directory}
          /docbkx/generated</generatedSourceDirectory>
        <xincludeSupported>true</xincludeSupported>
        <paperType>A4</paperType>
        <fop1Extensions>1</fop1Extensions>

        <foCustomization>src/docbkx-stylesheet/fo/docbook.xsl</foCustomization>

        <customizationParameters>
          <!-- additional XSLT parameters-->
          <parameter>
            <name>key</name>
            <value>value</value>
          </parameter>
        </customizationParameters>
      </configuration>
    </plugin>
  </plugins>
</build>

<pluginRepositories>
  <pluginRepository>
    <id>docbkx.snapshots</id>
    <name>Maven Plugin Snapshots</name>
    <url>http://docbkx-tools.sourceforge.net/snapshots/</url>
    <releases>
      <enabled>false</enabled>
    </releases>
  </pluginRepository>
</pluginRepositories>
```

```
<snapshots>
  <enabled>true</enabled>
</snapshots>
</pluginRepository>
</pluginRepositories>
```

The project structure is listed as follows.

```
[jfang@Mars tellurium-reference]$ tree .
.
|-- pom.xml
`-- src
    |-- docbkx
    |   |-- media
    |   |   |-- EngineGroupLocatingFlow.png
    |   |   |-- ExportToGroovySmall.png
    |   |   |-- FireFoxUserProfile.png
    |   |-- overview.xml
    |   |-- quickstart.xml
    |   |-- referenceprojects.xml
    |   |-- resources.xml
    |   |-- tellurium-reference.xml
    |   |-- trace.xml
    |   |-- trump.xml
    |   |-- udl.xml
    |   |-- uiobjects.xml
    |   |-- whatsnew.xml
    |   |-- widgets.xml
    |-- docbkx-stylesheet
    |-- fo
    |-- docbook.xsl
```

DocBook

Book

DocBook has two types, i.e., article and book. Tellurium reference uses the book format, which includes the following sections.

```
<?xml version="1.0" encoding="UTF-8"?>
<book version="5.0" xmlns="http://docbook.org/ns/docbook"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  xmlns:db="http://docbook.org/ns/docbook" xml:lang="en">
  <info>
    <title>Tellurium Automated Testing Framework</title>
    <subtitle>Reference Documentation</subtitle>
    <pubdate>May 15, 2010</pubdate>
    <releaseinfo>v0.7.0</releaseinfo>

    <authorgroup>
      <author>
        <firstname>Jian</firstname>
        <surname>Fang</surname>
      </author>
      ...
    </authorgroup>

    <copyright>
      <year>2010</year>
      <holder>TelluriumSource</holder>
    </copyright>

    <legalnotice>
      <para>
```

```

        Licensed under the Apache License, Version 2.0 (the "License"); you may not use this
        file except in compliance with the License. You may obtain a copy of the License at:
    </para>
    <para>
        http://www.apache.org/licenses/LICENSE-2.0
    </para>
    <para>
        Unless required by applicable law or agreed to in writing, software distributed under
        the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
        CONDITIONS OF ANY KIND, either express or implied. See the License for the specific
        language governing permissions and limitations under the License.
    </para>
</legalnotice>

</info>
...
<book>

```

Chapter

Each chapter has the following XML format. For example, the overview.xml is as follows.

```

<?xml version="1.0" encoding="UTF-8"?>
<chapter version="5.0" xmlns="http://docbook.org/ns/docbook"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xi="http://www.w3.org/2001/XInclude"
    xmlns:ns5="http://www.w3.org/2000/svg"
    xmlns:ns4="http://www.w3.org/1998/Math/MathML"
    xmlns:ns3="http://www.w3.org/1999/xhtml"
    xmlns:db="http://docbook.org/ns/docbook">

    <title>Overview of Tellurium</title>

    <section>
        <title>What is Tellurium</title>
        ...
    </chapter>

```

Then in the main XML file tellurium-reference.xml, we can include the xml file in the following way.

```

<xi:include href="overview.xml"/>

<xi:include href="whatsnew.xml"/>

<xi:include href="quickstart.xml"/>

```

Similarly, the appendix is defined as:

```

<appendix version="5.0" xmlns="http://docbook.org/ns/docbook"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xi="http://www.w3.org/2001/XInclude"
    xmlns:ns5="http://www.w3.org/2000/svg"
    xmlns:ns4="http://www.w3.org/1998/Math/MathML"
    xmlns:ns3="http://www.w3.org/1999/xhtml"
    xmlns:db="http://docbook.org/ns/docbook">

    <title>FAQs</title>
    ...
</appendix>

```

section

section can be nested in docbook, for example.

```
<section>
  <title>Setup Tellurium Project in IDEs</title>
  <para>A Tellurium Project can be run in IntelliJ, NetBeans, Eclipse, or other
    IDEs that have Groovy support.
  </para>

  <para>If using Maven, open the POM file to let the IDE automatically build the
    project files.
  </para>

  <section>
    <title>IntelliJ IDEA</title>
    <para>
      IntelliJ IDEA Community edition is free and can be downloaded from
    </para>
  </section>
  ...
</section>
```

Link

Link can be defined with xlink:href.

```
<para>
  IntelliJ IDEA Community edition is free and can be downloaded from
  <link xlink:href="http://www.jetbrains.com/idea/download/">
    http://www.jetbrains.com/idea/download/</link>. A detailed guide
    is found on <link
      xlink:href="http://code.google.com/p/aost/wiki/CustomTelluriumIntelliJProject">
        How to create your own Tellurium testing project with IntelliJ 9.0 Community
        Edition</link>.
  </para>
```

Image

```
<mediaobject>
  <imageobject>
    <imagedata fileref="./media/tellurium3.png" scalefit="1" width="100%">
    </imagedata>
  </imageobject>
</mediaobject>
```

where scalefit is an option to fit to the width. Another option is scale to change the image size.

screen

screen is used to show console output.

```
<screen>
  java -jar selenium-server.jar -singlewindow
</screen>
```

programlist

```
<programlisting language="xml"><?db-font-size 75% ?>
<![CDATA[
ui.Form(uid: "Form", clocator: [tag: "form"]){
  Div(uid: "User", clocator: [:]){
    Selector(uid: "Sex", clocator: [:])
    InputBox(uid: "Input", clocator: [tag: "input", type: "text",
      name: "j_username"])
  }
  Container(uid: "Finish", clocator: [tag: "tr"]){
    SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit",
      value: "Login", name: "submit"])
  }
}
]]&gt;
&lt;/programlisting&gt;
```

where `<?db-font-size 75% ?>` is used to scale the code font size. In the meanwhile, the xsl style sheet must use the following transformation to change font size.

```
<xsl:attribute-set name="monospace.verbatim.properties">
  <xsl:attribute name="font-size">
    <xsl:choose>
      <xsl:when test="processing-instruction('db-font-size')"><xsl:value-of
        select="processing-instruction('db-font-size')"/></xsl:when>
      <xsl:otherwise>inherit</xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
</xsl:attribute-set>
```

list

`itemlist` is used to represent a list, for example.

```
<itemizedlist>
  <listitem>
    <para>tellurium-junit-archetype</para>
  </listitem>
  <listitem>
    <para>tellurium-testng-archetype</para>
  </listitem>
</itemizedlist>
```

If we need an ordered list with number index, we can use `orderedlist` instead.

```
<orderedlist>
  <listitem>
    <para>address the dynamic factors in Tellurium UI templates</para>
  </listitem>
  <listitem>
    <para>increase the flexibility of Tellurium UI templates.</para>
  </listitem>
</orderedlist>
```

table

The table tag is used to represent a table in docbook. For example.

```
<table id="uiobjectattributes">
  <title>UI Object Attributes</title>
  <tgroup cols="2">
    <colspec colname="c1" colwidth="1*" />
    <colspec colname="c2" colwidth="1*" />
    <thead>
      <row>
        <entry>ATTRIBUTE</entry>
        <entry>DESCRIPTION</entry>
      </row>
    </thead>
    <tbody>
      <row>
        <entry>UI Object</entry>
        <entry>Basic Tellurium component</entry>
      </row>
      <row>
        <entry>UiID</entry>
        <entry>UI object's identifier</entry>
      </row>
      <row>
        <entry>Namespace</entry>
        <entry>Used for XHTML</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

Tools

There are couple tools [<http://wiki.docbook.org/topic/ConvertOtherFormatsToDocBook>] to convert docBook to and from different formats. For example, we can use the following command herold [<http://www.michael-a-fuchs.de/>] to convert html to docbook.

```
herold --in=udl.html --out=udl.xml -r book -T
```

But the converted docbook version is 1.0 and we still need manually change some docbook formats in the converted file.

Appendix A. FAQs

When Did Tellurium Start?

Tellurium is over one year old in June 2010 if we count the date from the day it became an open source project. But actually, Tellurium had been through two phases of prototyping before that. The first prototype was created in 2007 to test our company's Dojo web applications, which was basically a Java framework based on Spring XML wiring and no UI modules. You have to use factories to create all UI objects.

As a result, it was not convenient to use. The second prototype was created in early 2008 to improve the usability of the first prototype. The UI module was introduced in the second prototype. Both prototypes had been used for a few internal projects before it was re-written in Groovy and became an open source project in June 2008. Notice that prototype framework is called AOST and it was officially renamed to the Tellurium Automated Testing framework (Tellurium) in July 2008 when it moved out of the prototyping phase and became a team project.

What Are the Main Differences Between Tellurium and Selenium?

Tellurium was created when I was a Selenium user and tried to address some of the shortcomings of the Selenium framework such as verbosity and fragility to changes. Selenium is a great web testing framework and up to 0.6.0, Tellurium uses Selenium core as the test driving engine. From Tellurium 0.7.0, we will gradually replace the Selenium core with our own Engine.

Although Tellurium was born from Selenium, there are some fundamental differences between Tellurium and Selenium, which mainly come from the fact that Tellurium is a UI module-based testing framework. For example, Tellurium focuses on a set of UI elements instead of individual ones. The UI module represents a composite UI object in the format of nested basic UI elements. For example, the download search module in Tellurium project site is defined as follows:

```
ui.Form(uid: "downloadSearch", clocator: [action: "list", method: "get", group: "true"]) {  
    Selector(uid: "downloadType", clocator: [name: "can", id: "can"])  
    InputBox(uid: "searchBox", clocator: [name: "q"])  
    SubmitButton(uid: "searchButton", clocator: [value: "Search"])  
}
```

With the UI module, Tellurium automatically generates runtime locators for you and there is no need to define XPath or other types of locators by yourself. Tellurium is robust, expressive, flexible, and reusable.

Do I Need to Know Groovy Before I Use Tellurium?

Tellurium Core is implemented in Groovy and Java to achieve expressiveness. But that does not mean you have to be familiar with Groovy before you start to use Tellurium. Tellurium creates DSL expressions for UI module, actions, and testing. Use a Groovy class to implement the UI module by extending the `DslContext` Groovy class. Then the user can write the rest using Java syntax. The test cases can be created in Java, Groovy, or DSL scripts. However, we do encourage getting familiar with Groovy to leverage its meta programming features.

To create a Tellurium project, install a Groovy plugin for your IDE. There are Groovy plugins for commonly used IDEs such as Eclipse, Netbeans, and IntelliJ. Refer to the following WIKI pages on how to set up Groovy and use Tellurium in different IDEs,

<http://code.google.com/p/aost/wiki/TelluriumReferenceProjectEclipseSetup> [http://code.google.com/p/aost/wiki/TelluriumReferenceProjectEclipseSetup]

<http://code.google.com/p/aost/wiki/TelluriumReferenceProjectNetBeansSetup> [http://code.google.com/p/aost/wiki/TelluriumReferenceProjectNetBeansSetup]

<http://code.google.com/p/aost/wiki/TelluriumReferenceProjectIntelliJSetup> [http://code.google.com/p/aost/wiki/TelluriumReferenceProjectIntelliJSetup]

What Unit Test and Functional Test Frameworks Does Tellurium Support?

Tellurium supports both JUnit and TestNG frameworks. Extend `TelluriumJavaTestCase` for JUnit and `TelluriumTestNGTestCase` for TestNG. For more details, please check the following WIKI pages:

<http://code.google.com/p/aost/wiki/BasicExample> [http://code.google.com/p/aost/wiki/BasicExample]

<http://code.google.com/p/aost/wiki/Introduction> [http://code.google.com/p/aost/wiki/Introduction]

Tellurium also provides data driven testing. Data Driven Testing is a different way to write tests. For example, test data are separated from the test scripts and the test flow is not controlled by the test scripts, but by the input file instead. In the input file, users can specify which test to run, what the input parameters are, and what the expected results are. More details can be found from "Tellurium Data Driven Testing" WIKI page,

<http://code.google.com/p/aost/wiki/DataDrivenTesting> [http://code.google.com/p/aost/wiki/DataDrivenTesting]

Does Tellurium Provide Any Tools to Automatically Generate UI Modules?

Tellurium UI Model Plugin (TrUMP) is a Firefox Plugin used to automatically generate UI modules simply by clicking on the web page under testing. A user can download it from the Tellurium download page at:

<http://code.google.com/p/aost/download/list> [http://code.google.com/p/aost/downloads/list]

or from Firefox Addons site at:

<https://addons.mozilla.org/en-US/firefox/addon/11035> [https://addons.mozilla.org/en-US/firefox/addon/11035]

The detailed user guide for TrUMP 0.1.0 is at:

<http://code.google.com/p/aost/wiki/TrUMP> [http://code.google.com/p/aost/wiki/TrUMP]

To understand more about how TrUMP works, please read "How does TrUMP work?" at:

<http://code.google.com/p/aost/wiki/HowTrUMPWorks> [http://code.google.com/p/aost/wiki/HowTrUMPWorks]

What Build System Does Tellurium Use?

Tellurium supports both Ant and Maven build systems. The ant build scripts are provided in Tellurium core and Tellurium reference projects. For Maven, please check out the Tellurium Maven guide at:

<http://code.google.com/p/aost/wiki/MavenHowTo>
MavenHowTo]

[<http://code.google.com/p/aost/wiki/>

What is the Best Way to Create a Tellurium Project?

Tellurium provides two reference projects for JUnit and TestNG project, respectively. Use one of them as a template project. Please see the reference project guide at:

<http://code.google.com/p/aost/wiki/ReferenceProjectGuide>
ReferenceProjectGuide]

[<http://code.google.com/p/aost/wiki/>

However, the best and easiest way to create a Tellurium project is to use Tellurium Maven archetypes. Tellurium provides two Maven archetypes. For example, tellurium-junit-archetype and tellurium-testng-archetype for Tellurium JUnit test project and Tellurium TestNG test project, respectively.

As a result, you can create a Tellurium project using one Maven command. For a Tellurium JUnit project, use:

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-junit-archetype\
-DarchetypeGroupId=org.telluriumsource\
-DarchetypeVersion=0.7.0-SNAPSHOT \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

and for a Tellurium TestNG project, use

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
-DarchetypeArtifactId=tellurium-testng-archetype\
-DarchetypeGroupId=org.telluriumsource \
-DarchetypeVersion=0.7.0-SNAPSHOT \
-DarchetypeRepository=http://maven.kungfuters.org/content/repositories/snapshots
```

For more details, please read "Tellurium Maven archetypes",

<http://code.google.com/p/aost/wiki/TelluriumMavenArchetypes>
wiki/TelluriumMavenArchetypes]

[<http://code.google.com/p/aost/>

Where Can I Find API Documents for Tellurium?

The user guide for Tellurium DSLs, other APIs, and default UI objects could be found at:

<http://code.google.com/p/aost/wiki/UserGuide070Introduction>
UserGuide070Introduction]

[<http://code.google.com/p/aost/wiki/>

Is There a Tellurium Tutorial Available?

Tellurium provides very detailed tutorials including basic examples, advanced examples, data driven testing examples, and Dsl script examples. We also provide Tellurium Tutorial Series. Please use Tellurium tutorial WIKI page as your starting point,

<http://code.google.com/p/aost/wiki/Tutorial> [<http://code.google.com/p/aost/wiki/Tutorial>]

We also provide a quick start, "Ten Minutes To Tellurium", at

<http://code.google.com/p/aost/wiki/TenMinutesToTellurium> [<http://code.google.com/p/aost/wiki/TenMinutesToTellurium>]

Where Can I Find a Sample Tellurium Configuration File?

Tellurium Sample Configuration File is available here [<http://code.google.com/p/aost/wiki/TelluriumSampleConfigurationFile>]

Tellurium Dependencies

Tellurium is built on top of Selenium at the current stage and it uses Selenium 1.0.1. Tellurium 0.7.0 was tested with Groovy 1.7.0 and Maven 2.0.9.

You can go to Tellurium core and run the following Maven command to check the dependencies. For example, the dependency tree for 0.7.0 is shown as follows:

```
$ mvn dependency:tree

[INFO] [dependency:tree {execution: default-cli}]
[INFO] org.telluriumsource:tellurium-core:jar:0.7.0-SNAPSHOT
[INFO] +- junit:junit:jar:4.7:compile
[INFO] +- org.testng:testng:jar:jdk15:5.8:compile
[INFO] +- caja:json_simple:jar:rl:compile
[INFO] +- org.apache.poi:poi:jar:3.0.1-FINAL:compile
[INFO] |   \- commons-logging:commons-logging:jar:1.1:compile
[INFO] |     \- log4j:log4j:jar:1.2.13:compile
[INFO] +- org.telluriumsource:tellurium-udl:jar:0.7.0-SNAPSHOT:compile
[INFO] |   \- org.antlr:antlr:jar:3.1.3:compile
[INFO] |     \- org.antlr:antlr-runtime:jar:3.1.3:compile
[INFO] |       \- org.antlr:stringtemplate:jar:3.2:compile
[INFO] |         \- antlr:antlr:jar:2.7.7:compile
[INFO] +- org.stringtree:stringtree-json:jar:2.0.10:compile
[INFO] +- org.seleniumhq.selenium.server:selenium-server:jar:1.0.1-te3-SNAPSHOT:compile
[INFO] +- org.seleniumhq.selenium.client-drivers:selenium-java-client-driver:jar:1.0.1:compile
[INFO] +- org.codehaus.groovy:groovy-all:jar:1.7.0:compile
[INFO] |   \- jline:jline:jar:0.9.94:compile
[INFO] +- org.codehaus.gmaven.runtime:gmaven-runtime-1.6:jar:1.2:compile
[INFO] |   +- org.slf4j:slf4j-api:jar:1.5.10:compile
[INFO] |   +- org.codehaus.gmaven.feature:gmaven-feature-support:jar:1.2:compile
[INFO] |   |   \- org.codehaus.gmaven.feature:gmaven-feature-api:jar:1.2:compile
[INFO] |   \- org.codehaus.gmaven.runtime:gmaven-runtime-support:jar:1.2:compile
[INFO] |     +- org.codehaus.gmaven.runtime:gmaven-runtime-api:jar:1.2:compile
[INFO] |     +- org.sonatype.gshell:gshell-io:jar:2.0:compile
[INFO] |     |   \- org.sonatype.gossip:gossip:jar:1.0:compile
[INFO] |     +- org.codehaus.plexus:plexus-utils:jar:1.5.5:compile
[INFO] |     \- com.thoughtworks.qdox:qdox:jar:1.8:compile
[INFO] \- bouncycastle:bcprov-jdk15:jar:140:compile
[INFO] -----
```

But be aware that some of the dependencies are required ONLY for Maven itself, for example, gmaven-runtime, bouncycastle, and plexus.

If you use ant, please download <http://maven.kungfuters.org/content/repositories/thirdparty/org/seleniumhq/selenium/server/selenium-server/1.0-te-3/selenium-server-1.0.1-te3.jar> [] and the Apache POI - the Java API for Microsoft Documents [<http://poi.apache.org/download.html>].

What Is the ui. in UI Module?

Very often, you will see the ui. symbol when you define Tellurium UI modules. For instance, look at the following GoogleSearchModule UI module:

```
ui.Container(uid: "GoogleSearchModule", clocator: [tag: "td"], group: "true"){
  InputBox(uid: "Input", clocator: [title: "Google Search"])
  SubmitButton(uid: "Search", clocator: [name: "btnG", value: "Google Search"])
  SubmitButton(uid: "Imfeelinglucky", clocator: [value: "I'm Feeling Lucky"])
}
```

If you have read the Tellurium core code, you will find the following line in the BaseDslContext class,

```
UiDslParser ui = new UiDslParser()
```

The ui is actually an instance of UiDslParser. On the above UI module, call the method "Container" on UiDslParser with a map of attributes plus a Closure with the following nested code.

```
{
  InputBox(uid: "Input", clocator: [title: "Google Search"])
  SubmitButton(uid: "Search", clocator: [name: "btnG", value: "Google Search"])
  SubmitButton(uid: "Imfeelinglucky", clocator: [value: "I'm Feeling Lucky"])
}
```

Look at what the UiDslParser actually does from the source code:

```
class UiDslParser extends BuilderSupport{

  def registry = [:]

  def UiObjectBuilderRegistry builderRegistry = new UiObjectBuilderRegistry()

  protected Object createNode(Object name) {
  }

  ....
}
```

The UiDslParser extends the Groovy BuilderSupport class and works as a parser for what ever you passed in starting from Container(uid: "GoogleSearchModule", clocator: [tag: "td"], group: "true") in the above example.

You may notice that the BuilderSupport class needs to handle couple call back methods such as:

```
protected Object createNode(Object name)

protected Object createNode(Object name, Object value)

protected Object createNode(Object name, Map map)

protected Object createNode(Object name, Map map, Object value)
```

```
protected void nodeCompleted(Object parent, Object node)

protected void setParent(Object parent, Object child)
```

If you are familiar with XML parser, you will see that this is really similar to the XML PUSH style parser. Define call back methods and the parser will parse the message to the end automatically.

The above callback methods are doing the similar thing. For example, to create a UI object when it sees the name like "Container", "InputBox", and "SubmitButton". The different createNode methods are used for different use cases.

Basically, what the UiDslParser does is to get the object name such as "Container" and then look at the UI builder registry to find the builder for that object, then use the builder to build that UI object. The UI builder registry is a hash map and you can find the Container builder by the object name "Container".

Also the UiDslParser will keep the parse results. For example, UI objects in a registry so that you can refer to them by UID such as "GoogleSearchModule.Search", The object hierarchy is handled by the setParent method.

How Do I Add My Own UI Object to Tellurium?

First, create your UI object groovy class by extending class UiObject or Container if it is a container type object. Then, create your UI object builder by extending class UiObjectBuilder. Finally, register your ui builder for your ui object by call method in class TelluriumFramework:

```
public void registerBuilder(String uiObjectName, UiObjectBuilder builder)
```

You can also register your builder in class UiObjectBuilderRegistry if you work on Tellurium source code directly.

From Tellurium 0.4.0, a global configuration file TelluriumConfig.groovy is used to customize Tellurium. You can also define your own UI object in this file as follows,

```
uiobject{
    builder{
        Icon="org.tellurium.builder.IconBuilder"
    }
}
```

That is to say, create the UI object and its builder and then in the configuration file specify the UI object name and its builder full class name. "Note": this feature is included from Tellurium 0.5.0.

How to Build Tellurium from Source

If you want to build Tellurium from source, you can check out the trunk code using the subversion command:

```
svn checkout http://aost.googlecode.com/svn/trunk/ tellurium
```

or using the Maven command:

```
mvn scm:checkout -DconnectionUrl=scm:svn:http://aost.googlecode.com/svn/trunk \
-DcheckoutDirectory=tellurium
```

Be aware that the Maven command calls the subversion client to do the job and you must have the client installed in your system.

To build the whole project, use:

```
mvn clean install
```

and Maven compiles source code and resources, compiles test code and test resources, runs all tests, and then installs all artifacts to your local repository under `YOUR_HOME/.m2/repository`.

Sometimes, tests may break and if you still want to proceed, please use the ignore flag:

```
mvn clean install -Dmaven.test.failure.ignore=true
```

To build an individual project, go to that project directory and run the same command as above.

To run the tests, use the command:

```
mvn test
```

The sub-projects under the tools directory include Tellurium Maven archetypes and TrUMP code, you may not really want to build them by yourself. For TrUMP, the artifacts include a .xpi file.

The assembly project just creates a set of tar files and you may not need to build it either.

Tellurium also provides ant build scripts. You may need to change some of the settings in the `build.properties` file so that it matches your environment. For example, the settings for `javahome` and `javac.compiler`.

What is the Issue with Selenium XPath Expressions and Why is There a Need to Create a UI Module?

The problem is not in XPath itself, but the way you use it. If the following XPath locator is:

```
"//div/table[@id='something']/div[2]/div[3]/div[1]/div[6]"
```

then the problem is easily seen. It is not robust. Along the path

```
div -> table -> div -> div ->div -> div
```

if anything is changed there, your XPath is no longer valid. For example, if you add additional UI elements and the new XPath was changed to:

```
"//div[2]/table[@id='something']/div[3]/div[3]/div[1]/div[6]"
```

you would have to keep updating the XPath. For Tellurium, it focuses on element attributes, not the XPath, and it can be adaptive to the changes to some degree.

More importantly, Tellurium uses the group locating concept to use information from a group of UI elements to locate them in the DOM. In most cases, the group of elements are enough to decide their locations in the DOM, that is to say, your UI element's location does not depend on any parent or grandparent elements.

For instance, in the example above, if you use the group locating concept to find locators for the following part of UI elements directly:

```
"div[3]/div[1]/div[6]"
```

then they do not depend on the portion certainly.

```
"div[2]/table[@id='something']/div[3]"
```

The UI elements can address any changes in the portion of :

```
"div[2]/table[@id='something']/div[3]"
```

Note: In Tellurium, the user will not use an XPath locator directly.

Furthermore, the syntax of:

```
selenium.type("//input[@title='Google Search']", input)
selenium.click("//input[@name='btnG' and @type='submit']")

...

selenium.type("//input[@title='Google Search']", input)
selenium.click("//input[@name='btnG' and @type='submit']")

...

selenium.type("//input[@title='Google Search']", input)
selenium.click("//input[@name='btnG' and @type='submit']")

...
```


everywhere is really ugly to users. Especially if someone needs to take over your code. In Tellurium, the UiID is used and it is very clear to users what you are acting upon.

```
click "google_start_page.googlesearch"
```

The test script created by Selenium IDE is a mess of actions, not modularized. Other people may take quite some time to figure out what the script actually does. And it is quite difficult to refactor and reuse them. Even the UI is not changed, there are data dependence there and for most cases, you simply cannot just "record and replay" in practical tests.

In Tellurium, once you defined the UI module, for example, the Google search module, you can always reuse them and write as many test cases as possible.

Selenium is cool and the idea is brilliant. But it is really for low level testing only, focusing on one element at a time and it does not have the whole UI module in mind. That is why another tier on top of it is needed so that you can have a UI module-oriented testing script and not the locator-oriented one. Tellurium is one of the frameworks designed for this purpose.

As mentioned above, Selenium is quite a low level process and it is really difficult to handle more complicated UI components like a data grid. Tellurium can handle them easily. Please see the test scripts for the Tellurium project web site.

How to write assertions in Tellurium DSL scripts

Tellurium DSL scripts are actually Groovy scripts written in DSL syntax. Thus, Tellurium DSL scripts support all assertions in JUnit 3.8, which GroovyTestCase extends.

But for Tellurium Data Driven testing scripts, it is a bit different. Usually, you should use:

```
compareResult expected, actual
```

and it in turn calls

```
assertEquals(expected, actual)
```

This is because DDT script has to be general enough for different input data. If you want to use your own assertions, Tellurium provides the capability for that. You should use a Groovy closure to replace the default asserEquals. For example, in your DDT DSL script, you can overwrite the default behaviour using

```
compareResult(expected, actual){
    assertNotNull(expected)
    assertNotNull(actual)
    assertTrue(expected.size() == actual.size())
}
```

This brings up one interesting question "why should I put assertions inside `compareResult`, not anywhere in the script?" The answer is that you can put assertions any where in the DDT script, but that will cause different behaviour if the assertion fails.

If you put assertions in `compareResult` and the assertion fails, the `AssertionFailedError` will be captured and that comparison fails, but the rest script inside a test will continue executing. But if you put assertions outside of `compareResult`, the `AssertionFailedError` will lead to the failure of the current test. The exception will be recorded and the current test will be stopped. The next test will take over and execute.

How to upgrade Firefox version in Selenium server

You can do the following steps:

- unpack the custom selenium-server by running

```
jar xvf selenium-server.jar
```

- Find the versions in `install.rdf` and change them, for instance,

```
/customProfileDirCUSTFF/extensions/readyst...@openqa.org/install.rdf
./customProfileDirCUSTFF/extensions/{538F0036-F358-4f84-A764-89FB437166B4}/install.rdf
./customProfileDirCUSTFFCHROME/extensions/readyst...@openqa.org/install.rdf
./customProfileDirCUSTFFCHROME/extensions/{503A0CD4-EDC8-489b-853B-19E0BAA8F0A4}/install.rdf
./customProfileDirCUSTFFCHROME/extensions/{538F0036-F358-4f84-A764-89FB437166B4}/install.rdf
./customProfileDirCUSTFFCHROME/extensions/{636fd8b0-ce2b-4e00-b812-2afbe77ee899}/install.rdf
```

change the versions from

```
<em:targetApplication>
  <Description>
    <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
    <em:minVersion>1.4.1</em:minVersion>
    <em:maxVersion>3.5.*</em:maxVersion>
  </Description>
</em:targetApplication>
```

to

```
<em:targetApplication>
  <Description>
    <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
    <em:minVersion>1.4.1</em:minVersion>
    <em:maxVersion>3.6.*</em:maxVersion>
  </Description>
</em:targetApplication>
```

- repack the jar file.

```
jar cmf META-INF/MANIFEST.MF selenium-server.jar *
```

How to run Selenium server remotely in Tellurium

The steps to use remote selenium server in Tellurium are first to run selenium sever on the remote machine, saying 192.168.1.106

```
java -jar selenium-server.jar -port 4444
```

for more selenium server options, please use the following commands:

```
java -jar selenium-server.jar --help
```

Then, you should modify the TelluriumConfig.groovy as follows,

```
tellurium{
    //embedded selenium server configuration
    embeddedservlet {
        //port number
        port = "4444"
        //whether to use multiple windows
        useMultiWindows = false
        //whether to run the embedded selenium server.
        //If false, you need to manually set up a selenium server
        runInternally = false
    }
    //the configuration for the connector that connects the selenium client
    //to the selenium server
    connector{
        //selenium server host
        //please change the host if you run the Selenium server remotely
        serverHost = "192.168.1.106"

        //server port number the client needs to connect
        port = "4444"
        //base URL
        baseUrl = "http://localhost:8080"
        //Browser setting, valid options are
        // *firefox [absolute path]
        // *iexplore [absolute path]
        // *chrome
        browser = "*iehta"
    }
    .....
}
```

That is to say, you should disable the embedded selenium server by specifying

```
runInternally = false
```

and specify the remote selenium server host as

```
serverHost = "192.168.1.106"
```

After that, you can run the test just like using the embedded selenium server. But be aware that there are some performance degradation, i.e., the test is slower with remote selenium server.

Differences among Tellurium Table, List, and Container

Container is most like an abstract object and it can be of any type of UI objects that can hold other UI objects. The UI objects inside the Container are fixed once it is defined and inner objects can be referred directly by "container_uid.object_uid". Be aware that Tellurium Container type objects can hold any UI objects including container type objects and nested UI can be constructed in this way.

>Table and List are both Container type UI objects and are designed mainly for dynamic size UI objects. For example, table can be used to mode data grid, whose size is not fixed and is dynamic at run-time. For this purpose, the UI objects inside the table can be used as templates and how they are used is totally dependent on their UIDs. For more details on how the UIDs for List and Table are defined, please see Tellurium UID Description Language [<http://code.google.com/p/aost/wiki/TelluriumUIDDescriptionLanguage>].

Once the templates are defined and you use `table[i][j]` to refer the inner object, Tellurium will automatically apply the above rules and find the actual UI object for you. If no templates can be found, Tellurium will use default UI object TextBox.

One such good example is the data grid of Tellurium downloads page:

```
ui.Table(uid: "downloadResult", clocator: [id: "resultstable", class: "results"]){
    //define table elements
    //for the border column
    TextBox(uid: "{row: all, column: 1}", clocator: [:])
    //the summary + labels column consists of a list of UrlLinks
    List(uid: "{row:all, column: 3}", clocator: [:]){
        UrlLink(uid: "{all}", clocator: [:])
    }
    //For the rest, just UrlLink
    UrlLink(uid: "{row: all, column: all}", clocator: [:])
}
```

How do I use a Firefox profile in Tellurium

You can specify the profile in Tellurium Configuration file `TelluriumConfig.groovy` as follows,

```
embeddedserver {
    .....
    //profile location, for example,
    profile = "/home/jfang/.mozilla/firefox/820j3ca9.default"
}
```

This is especially useful if you are behind a firewall. Please read more about Firefox Profiles [http://support.mozilla.com/en-US/kb/Profiles#On_Windows_2000_and_XP].

If you run the Selenium server externally, you can specify the Firefox profile using the following option:

`-firefoxProfileTemplate <dir>`: normally, we generate a fresh empty Firefox profile every time we launch. You can specify a directory to make us use that directory instead.

How to Overwrite Tellurium Settings in My Test Class

TelluriumConfig.groovy acts like a global setting file if you do not want to manually change it. Now, the BaseTelluriumJavaTestCase provides two methods for you to overwrite the default settings,

```
public static void setCustomConfig(boolean runInternally, int port, String browser,
                                   boolean useMultiWindows, String profileLocation)

public static void setCustomConfig(boolean runInternally, int port, String browser,
                                   boolean useMultiWindows, String profileLocation, String serverHost)
```

As you result, if you want to use your custom settings for your specific test class, you can use the following way taking the Google test case as an example,

```
public class GoogleStartPageJavaTestCase extends TelluriumJavaTestCase
{
    static{
        setCustomConfig(true, 5555, "*chrome", true, null);
    }
    ...
}
```

How to reuse a frequently used set of elements

The "Include" syntax in Ui module definition can be used for this purpose. You can put frequently used UI modules into a base class, for example,

```
public class BaseUiModule extends DslContext {
    public void defineBaseUi() {
        ui.Container(uid: "SearchModule", clocator: [tag: "td", group: "true"]) {
            InputBox(uid: "Input", clocator: [title: "Google Search"])
            SubmitButton(uid: "Search", clocator: [name: "btnG", value: "Google Search"])
            SubmitButton(uid: "ImFeelingLucky", clocator: [value: "I'm Feeling Lucky"])
        }

        ui.Container(uid: "GoogleBooksList", clocator: [tag: "table", id: "hp_table"],
                    group: "true")
        {
            TextBox(uid: "category", clocator: [tag: "div", class: "sub_cat_title"])
            List(uid: "subcategory", clocator: [tag: "div", class: "sub_cat_section"],
                separator: "p")
            {
                UrlLink(uid: "{all}", clocator: [:])
            }
        }
    }
}
```

Then you can extend this base Ui module as follows,

```
public class ExtendUiModule extends BaseUiModule {

    public void defineUi() {
        defineBaseUi()

        ui.Container(uid: "Google", clocator: [tag: "table"]) {
            Include(ref: "SearchModule")
            Container(uid: "Options", clocator: [tag: "td", position: "3"]) {
                UriLink(uid: "LanguageTools", clocator: [tag: "a", text: "Language Tools"])
                UriLink(uid: "SearchPreferences", clocator: [tag: "a",
                                                            text: "Search Preferences"])
                UriLink(uid: "AdvancedSearch", clocator: [tag: "a", text: "Advanced Search"])
            }
        }

        ui.Container(uid: "Test", clocator: [tag: "div"]) {
            Include(uid: "newcategory", ref: "GoogleBooksList.category")
            Include(uid: "secondcategory", ref: "GoogleBooksList.category")
            Include(uid: "newsubcategory", ref: "GoogleBooksList.subcategory")
        }
    }
}
```

Note that the "Include" must have the ref attribute to refer to the element it wants to include. You can still specify the uid for the object (if you do not need a different uid, you do not need the uid), if the object uid is not equal to the original one, Tellurium will clone a new object for you so that you can have multiple objects with different uids.

How to handle Table with multiple tbody elements

The StandardTable is designed for tables with the following format

```
table
  thead
    tr
      td
      ...
      td
  tbody
    tr
      td
      ...
      td
    ...
    tbody (multiple tbodies)
      tr
        td
        ...
        td
      ...
  tfoot
    tr
      td
      ...
      td
```

For a StandardTable, you can specify UI templates for different tbodies. For Example:

```
ui.StandardTable(uid: "table", clocator: [id: "std"]) {
    UrlLink(uid: "{header: 2}", clocator: [text: "**Filename"])
    UrlLink(uid: "{header: 3}", clocator: [text: "**Uploaded"])
    UrlLink(uid: "{header: 4}", clocator: [text: "**Size"])
    TextBox(uid: "{header: all}", clocator: [:])

    Selector(uid: "{tbody: 1, row:1, column: 3}", clocator: [name: "can"])
    SubmitButton(uid: "{tbody: 1, row:1, column:4}",
        clocator: [value: "Search", name: "btn"])
    InputBox(uid: "{tbody: 1, row:2, column:3}", clocator: [name: "words"])
    InputBox(uid: "{tbody: 2, row:2, column:3}", clocator: [name: "without"])
    InputBox(uid: "{tbody: 2, row:all, column:1}", clocator: [name: "labels"])

    TextBox(uid: "{footer: all}", clocator: [tag: "td"])
}
```

How to Run Tellurium Tests in Different Browsers

You could use the `openUrlWithBrowserParameters()` methods to change browser settings for different test cases in the same test class,

```
public static void openUrlWithBrowserParameters(String url, String
    serverHost, int serverPort, String baseUrl, String browser, String
    browserOptions)

public static void openUrlWithBrowserParameters(String url, String
    serverHost, int serverPort, String browser, String browserOptions)

public static void openUrlWithBrowserParameters(String url, String
    serverHost, int serverPort, String browser)
```

For example,

```
public class GoogleStartPageTestNGTestCase extends TelluriumTestNGTestCase {
    protected static NewGoogleStartPage ngsp;

    @BeforeClass
    public static void initUi() {
        ngsp = new NewGoogleStartPage();
        ngsp.defineUi();
    }

    @DataProvider(name = "browser-provider")
    public Object[][] browserParameters() {
        return new Object[][]{
            new Object[] {"localhost", 4444, "**chrome"},
            new Object[] {"localhost", 4444, "**firefox"}};
    }

    @Test(dataProvider = "browser-provider")
    @Parameters({"serverHost", "serverPort", "browser"})
    public void testGoogleSearch(String serverHost, int serverPort, String browser){
        openUrlWithBrowserParameters("http://www.google.com", serverHost,
            serverPort, browser);
        ngsp.doGoogleSearch("tellurium selenium Groovy Test");
        disconnectSeleniumServer();
    }

    @Test(dataProvider = "browser-provider")
    @Parameters({"serverHost", "serverPort", "browser"})
    public void testGoogleSearchFeelingLucky(String serverHost, int serverPort,
        String browser){
```

```
        openUrlWithBrowserParameters("http://www.google.com", serverHost,
                                     serverPort, browser);
        ngsp.doImFeelingLucky("tellurium selenium DSL Testing");
        disconnectSeleniumServer();
    }
}
```

How to use the new XPath library in Selenium

There are three methods in `DslContext` for you to select different XPath Library,

```
public void useDefaultXPathLibrary()

public void useJavascriptXPathLibrary()

public void useAjaxsltXPathLibrary()
```

The default one is the same as the "Ajaxslt" one. To use faster xpathlibrary, please call `useJavascriptXPathLibrary()`.

For example, in the test case file,

```
protected static NewGoogleStartPage ngsp;

@BeforeClass
public static void initUi() {
    ngsp = new NewGoogleStartPage();
    ngsp.defineUi();
    ngsp.useJavascriptXPathLibrary();
}
```

How to Debug Selenium Core

You can use Microsoft Script Debugger to debug the Selenium Core in IE. To debug the JavaScript code, follow the following step,

1. Start custom selenium server in multiWindow mode and another useful command option is `-debug`, which will print out all trace messages

```
java -jar selenium-server -multiWindow
```

2. Debug the Java code in IDE and set a break point somewhere in the code
3. Once the Java process paused, open up the Microsoft script debugger or Editor MSE7.exe.
4. Attach you debugger to the running IE instance and you will see the JavaScript you want to debug, set a break point there.
5. Resume you Java process and it will wait there once the breakpoint is hit in the JavaScript debugger. Then you can step into, step over, or run the JavaScript.

For Firefox, you can debug using Firebug using the same steps. But you may need to use Firefox profile to start the custom server server so that Firebug will be included in the launched instance.

How to Debug Tellurium in IE

Script to add Firebug Lite to a page

```
String script = "var firebug=document.createElement('script');" +
    "firebug.setAttribute('src',"
    + "'http://getfirebug.com/releases/lite/1.2/firebug-lite-compressed.js');" +
    "document.body.appendChild(firebug);" +
    "(function(){if(window.firebug.version){firebug.init();}else"
    + "{setTimeout(arguments.callee;)}})();" + "void(firebug);";
```

Call addScript Selenium command, which will add a script to the Selenium runner window.

```
addScript(script, "firebug-lite");
```

After the code has run you should see a Firebug Lite window in the Selenium runner window (use multi-window mode). From there you can do the usual debug stuff, such as this, which uses jQuery to find all parents of an element.

```
tejQuery(selenium.browserbot.findElement("your-locator")).parents()
```

Call the following command to remove the script.

```
removeScript("firebug-lite");
```

How to use jQuery Selector with weird characters in its ID

You should escape the "." or other jQuery reserved characters. For example, use "dateOfBirth.\\month" for "dateOfBirth.month" as the ID.

How to Use Tellurium for XHTML

For XHTML, you need to use the *namespace* attribute in the UI object, for example,

```
ui.Container(uid:"caseRecordPopUp", clocator:[id:"CaseRecordsPopUp",
    tag:"div"],namespace:"xhtml",group:"true") {
    Container(uid:"date",clocator:[id:"caseRecordPopUpDate",
    tag:"input"], namespace:"xforms")
    .....
}
```

You can register a custom namespace as follows,

```
registerNamespace("xforms", "http://www.w3.org/2002/xforms")
```

and use the following method to get back the namespace

```
getNamespace("xforms");
```

What Are the Differences Between `connectUrl` and `openUrl`

Starting from Tellurium 0.7.0, `connectUrl()` and `openUrl()` are used for different purposes. That is to say, `openUrl()` is used to instantiate a new browser session for each call, while `connectUrl()` reuses the same browser session. Under the hood, you have

```
openUrl() = connectSeleniumServer() + connectUrl()
```

As a result, you need to call `connectSeleniumServer()` before you call `connectUrl()` and then keep calling `connectUrl()` if you want to reuse the same browser session.

How to do Attribute Partial Matching in Tellurium

Tellurium adopts jQuery style partial matching, that is to say, you should use the following format for partial matching:

```
"PARTIAL_MATCHING_SIGNYour_string"
```

where the partial matching signs include:

- `!` : either don't have the specified attribute or do have the specified attribute but not with a certain value.
- `^` : have the specified attribute and it starts with a certain value.
- `\$` : have the specified attribute and it ends with a certain value. Be aware that `$` is reserved in Groovy for `GString`, please add `"\"` to escape it.
- `*` : have the specified attribute and it contains a certain value.

Be aware, due to the limitations of XPath, `$` and `*` are the same in XPath locator.

>Examples:

```
clocator: [href: '!xevent']
clocator: [href: '^xevent']
clocator: [href: '\$xevent']
clocator: [href: '*xevent']
```

What are the rules to define Tellurium UIDs

With the addition of Tellurium UID Description Language (UDL), for most UI objects, the UIDs are IDs, but for List and Table objects, the UIDs include UI template definitions and IDs.

The rules for the ID definition are as follows:

- Starts with a letter and followed by either letter, digits, or underscore " _".
- `row`, `column`, `header`, `tbody`, `tfooter` are reserved by UDL as tokens and you should not use them as IDs.

How to load Tellurium configuration from a String

From 0.7.0, Tellurium added support to load Tellurium configuration from a JSON String. For example, you can define the Tellurium configuration JSON string as follows.

```
public class JettyLogonModule extends DslContext {
    public static String JSON_CONF = ""{"tellurium":{"test":{"result":
        {"reporter":"XMLResultReporter","filename":"TestResult.output",
            "output":"Console"},"exception":{"filenamePattern":
                "Screenshot?.png","captureScreenshot":false},"execution":
                {"trace":false},"accessor":{"checkElement":false},
                "embeddedserver":{"port":"4444","browserSessionReuse":false,
                "debugMode":false,"ensureCleanSession":false,"interactive":
                false,"avoidProxy":false,"timeoutInSeconds":30,"runInternally"
                :true,"trustAllSSLCertificates":true,"useMultiWindows":false,
                "userExtension":"","profile":""},"uiobject":{"builder":{}}},
                "eventhandler":{"checkElement":false,"extraEvent":false},
                "i18n":{"locale":"en_US"},"connector":{"baseUrl":
                "http://localhost:8080","port":"4444","browser":"*chrome",
                "customClass":"","serverHost":"localhost","options":""},
                "bundle":{"maxMacroCmd":5,"useMacroCommand":true},"datadriven":
                {"dataproducer":{"reader":"PipeFileReader"},"widget":
                {"module":{"included":""}}}}"";
    ...
}
```

where the pretty look version of the JSON String is shown as follows.

```
{
  "tellurium": {
    "test": {
      "result": {
        "reporter": "XMLResultReporter",
        "filename": "TestResult.output",
        "output": "Console"
      },
      "exception": {
        "filenamePattern": "Screenshot?.png",
        "captureScreenshot": false
      },
      "execution": {
        "trace": false
      }
    },
    "accessor": {
      "checkElement": false
    },
    "embeddedserver": {
      "port": "4444",
      "browserSessionReuse": false,

```

```
        "debugMode": false,
        "ensureCleanSession": false,
        "interactive": false,
        "avoidProxy": false,
        "timeoutInSeconds": 30,
        "runInternally": true,
        "trustAllSSLCertificates": true,
        "useMultiWindows": false,
        "userExtension": "",
        "profile": ""
    },
    "uiobject": {
        "builder": { }
    },
    "eventhandler": {
        "checkElement": false,
        "extraEvent": false
    },
    "i18n": {
        "locale": "en_US"
    },
    "connector": {
        "baseUrl": "http://localhost:8080",
        "port": "4444",
        "browser": "*chrome",
        "customClass": "",
        "serverHost": "localhost",
        "options": ""
    },
    "bundle": {
        "maxMacroCmd": 5,
        "useMacroCommand": true
    },
    "datadriven": {
        "dataproducer": {
            "reader": "PipeFileReader"
        }
    },
    "widget": {
        "module": {
            "included": ""
        }
    }
}
}
```

Then in the test case, we can load the configuration.

```
public class JettyLogonJUnitTestCase extends TelluriumMockJUnitTestCase {
    private static JettyLogonModule jlm;
    static{
        Environment env = Environment.getEnvironment();
        env.useConfigString(JettyLogonModule.JSON_CONF);
    }
    ...
}
```

How to register a custom method in Tellurium API

For a custom method, the new Tellurium Engine provides the following method for users to register the custom method.

```
Tellurium.prototype.registerApi = function(apiName, requireElement, returnType)
```

where *apiName* is the method name, *requireElement* means it requires a locator. The *returnType* is defined by the enum in core

```
public enum ReturnType {  
    VOID,  
    NUMBER,  
    STRING,  
    BOOLEAN,  
    ARRAY,  
    MAP,  
    OBJECT  
}
```

For example, you can register a custom method as follows,

```
tejQuery(document).ready(function() {  
    tellurium.registerApi("getUiByTag", false, "ARRAY");  
});
```

How to Access Tellurium Maven Repo Behind a Firewall

You can download NTLM Authorization Proxy Server [<http://ntlmaps.sourceforge.net/>] and install it as a local proxy. Then add the following proxy settings in your HOME/.m2/settings.xml.

```
<proxies>  
  <proxy>  
    <active>true</active>  
    <protocol>http</protocol>  
    <host>localhost</host>  
    <port>5865</port>  
  </proxy>  
</proxies>
```

The *nonProxyHosts* parameter should include all the hosts you don't want them to go through the proxy server, for example, we have the following setting for that.

```
<nonProxyHosts>*.mycompany.com|*.mycompanydomain2.com</nonProxyHosts>
```

How to Generate IDE project files

Tellurium provides Maven archetypes [http://code.google.com/p/aost/wiki/UserGuide070TelluriumSubprojects#Tellurium_Maven_Archetypes] for you to generate a Tellurium Maven project. Once you create the Maven project, you can create the IDE project files automatically.

For IntelliJ IDEA, you don't need to do anything and just simply open the project by picking the pom file.

For NetBeans, you can use the following Maven command [<http://maven.apache.org/guides/mini/guide-ide-netbeans/guide-ide-netbeans.html>] to create NetBeans project files.

```
mvn netbeans-freeform:generate-netbeans-project
```

For Eclipse, you can use a similar Maven command [<http://maven.apache.org/guides/mini/guide-ide-eclipse.html>]

```
mvn eclipse:eclipse
```

How to Run Tellurium Tests in Google Chrome

Selenium supports the following web browsers,

```
*firefox
*mock
*firefoxproxy
*piifirefox
*chrome
*iexploreproxy
*iexplore
*firefox3
*safariproxy
*googlechrome
*konqueror
*firefox2
*safari
*piiexplore
*firefoxchrome
*opera
*iehta
*custom
```

For Google Chrome, you should use `*googlechrome`. But there are bugs for the current Google Chrome support in Selenium:

- There have been reports that Chrome support for selenium won't work on Windows XP 64-bit.
- The Google Chrome launcher does not support the `avoidProxy` option (SRC-524 [<http://jira.openqa.org/browse/SRC-524>])
- `http://localhost` doesn't work as a starting URL in Google Chrome (SRC-529 [<http://jira.openqa.org/browse/SRC-529>])

How to run headless tests with Xvfb

First, you need to install Xvfb. In linux, do

```
[root@Mars ~]# yum search Xvfb
```

you will see which rpm is for your linux os. For example, I found `xorg-x11-server-Xvfb.x86_64` : A X Windows System virtual framebuffer X server.

I use the following command to install it.

```
[root@Mars ~]# yum install xorg-x11-server-Xvfb.x86_64
```

If you use Maven to run test, you can use the following script

```
#!/bin/bash

nohup startx -- `which Xvfb` :20 -screen 0 1024x768x24 & sleep 7
DISPLAY=:20 firefox & DISPLAY=:20 mvn test
pkill Xvfb
```

For ant, you can replace "mvn test" with your ant task.

How to setup Groovy Grape

Tellurium 0.7.0 starts to use Groovy Grape to run DSL script. To install Groovy Grape, you need to customize the ivy settings that Grape uses by creating a ~/.groovy/grapeConfig.xml file.

A sample grapeConfig.xml is as follows.

```
<ivysettings>
  <settings defaultResolver="downloadGrapes"/>
  <property
    name="local-maven2-pattern"
    value="${user.home}/.m2/repository/[organisation]/[module]/[revision]/
      [module]-[revision](-[classifier]).[ext]"
    override="false" />
  <resolvers>
    <chain name="downloadGrapes">
      <filesystem name="cachedGrapes">
        <ivy pattern="${user.home}/.groovy/grapes/[organisation]/[module]/ivy-[revision].xml"/>
        <artifact pattern="${user.home}/.groovy/grapes/[organisation]/[module]/[type]s/
          [artifact]-[revision].[ext]"/>
      </filesystem>
      <filesystem name="local-maven-2" m2compatible="true" local="true">
        <ivy pattern="${local-maven2-pattern}"/>
        <artifact pattern="${local-maven2-pattern}"/>
      </filesystem>
      <!-- todo add 'endorsed groovy extensions' resolver here -->
      <ibiblio name="kungfutures.3rdparty" root="http://maven.kungfutures.org/content/repositories/thirdparty/"
        m2compatible="true"/>
      <ibiblio name="codehaus" root="http://repository.codehaus.org/" m2compatible="true"/>
      <ibiblio name="ibiblio" m2compatible="true"/>
      <ibiblio name="java.net2" root="http://download.java.net/maven/2/" m2compatible="true"/>
      <ibiblio name="openqa" root="http://archiva.openqa.org/repository/releases/" m2compatible="true"/>
      <ibiblio name="kungfutures.snapshot" root="http://maven.kungfutures.org/content/repositories/snapshots/"
        m2compatible="true"/>
      <ibiblio name="kungfutures.release" root="http://maven.kungfutures.org/content/repositories/releases/"
        m2compatible="true"/>
    </chain>
  </resolvers>
</ivysettings>
```

How to Search Tellurium Documents

The trick is to use google's site meta command, which will narrow the search to one domain. You can use the following key words to do Google search,

```
site:aost.googlecode.com YOUR_KEY_WORDS
```

For example,

```
site:aost.googlecode.com datepicker
```

How to Contribute to Tellurium

We welcome contributions in many different ways, for example,

1. Try out Tellurium
2. Use Tellurium in your project and report bugs
3. Ask questions and answer other users' questions
4. Promote Tellurium and post your experience on Tellurium
5. Fix bugs for Tellurium
6. Bring in new ideas and suggestions

Our project team is open for new members and is expecting new members. We need some new developers who are JavaScript and jQuery experts to join our development team. If you have free time and want to contribute to Tellurium, please contact Jian Fang or other existing Tellurium members.

We also expect to have some Evangelism Team members, who could write blogs and articles about Tellurium, or present Tellurium to people, i.e., promote Tellurium in various ways. We welcome Evangelism Team members from different parts of the world and promote Tellurium in different Languages.

Tellurium Future Directions

- Use jQuery to re-implement Selenium APIs. 0.7.0 has implemented a set of Selenium APIs, but there are still a lot of remaining work.
- Algorithm design, for example, automatically build UI module from HTML source. Reverse engineering to build UI templates from HTML source is very challenging.
- Tellurium Widgets, create reusable Dojo, ExtJS widgets so that other people can reuse the widgets simply by including the jar files to their projects.
- Tellurium UID Description Language improvement.
- Improve Trump Firefox Plugin, for example, generate test script as well as UI modules so that Tellurium is not only good for developers, but also QA people.
- Add Behavior Driven Testing support
- Add testing flow support. Many unit testing and functional testing frameworks do not really have the testing flow/stage concept.
- Tellurium as a cloud testing tool.
- Tellurium as a web security testing tool.
- IDE and other plugins.
- Improve code quality.

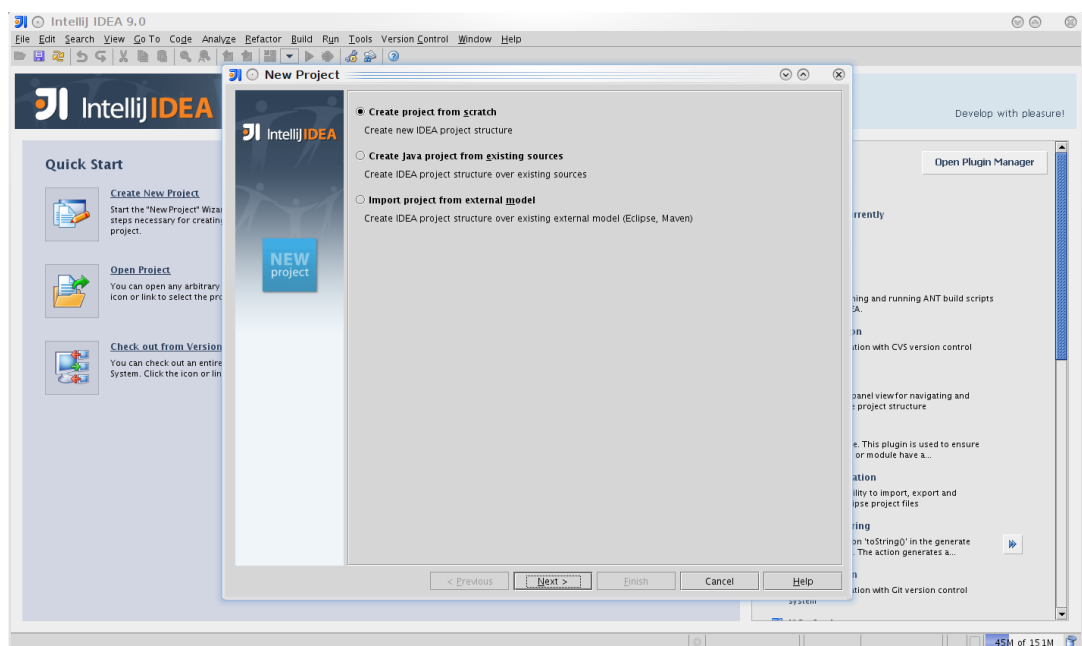
Appendix B. Tellurium Project Setup with IntelliJ

Prerequisites

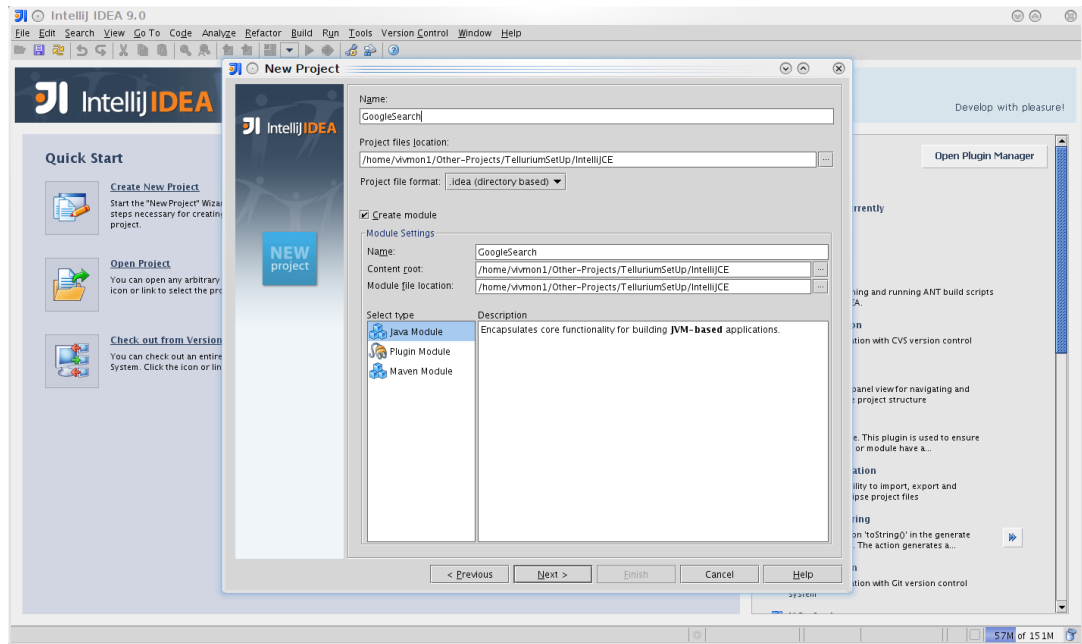
- Download IntelliJ IDEA Community edition. Its free and its got the best Groovy support.
<http://www.jetbrains.com/idea/download/>
- Download the latest Groovy from the following location and unpack to your system.
<http://groovy.codehaus.org/Download>
- Create a **lib** folder to include the dependent jar files
- Download the **tellurium-core.jar** file into the created **lib** folder from the following location.
<http://code.google.com/p/aost/downloads/list>
- Download tellurium dependencies jars into the created **lib** folder.
- Download the **test_source.zip** file from the following location. Zip file contains the UI Groovy file NewGoogleStartPage.groovy, Java Test case NewGoogleStartPageTestCase.java and Config Groovy file TelluriumConfig.groovy, which will be used to setup the project.
<http://code.google.com/p/aost/downloads/list>

Project Setup

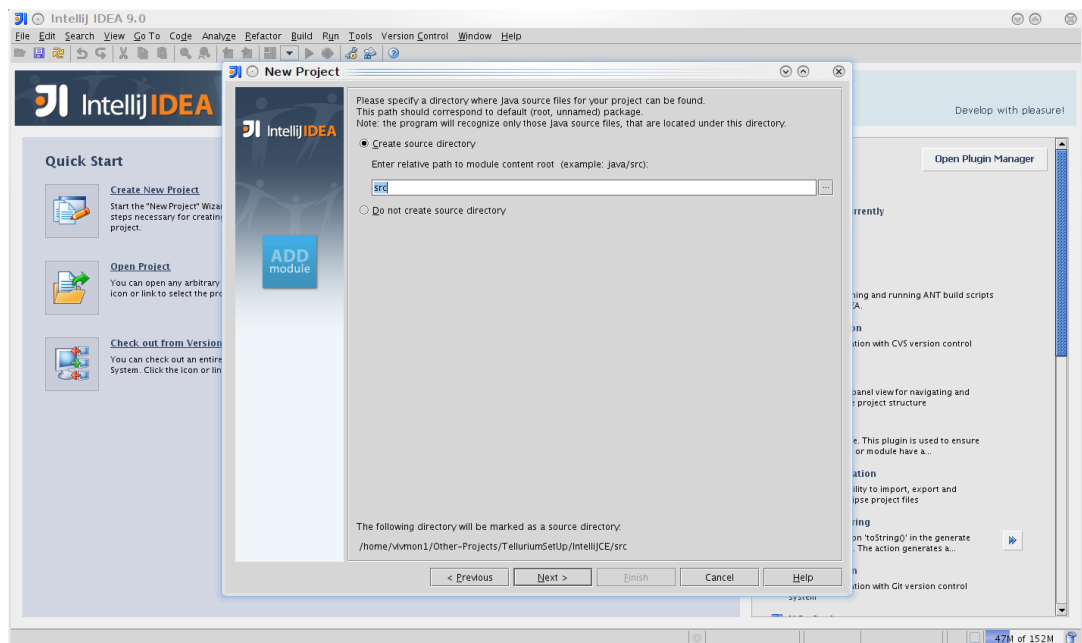
- Launch IntelliJ. Create a new Project.**File > New Project > Create Project From Scratch.**



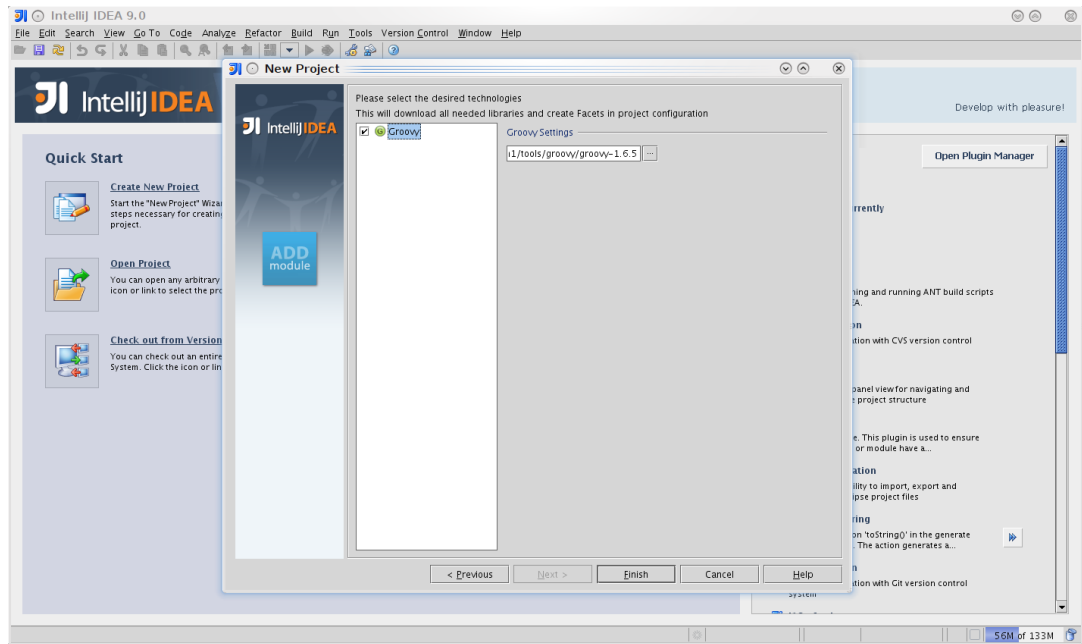
- Set the project details as shown below and Click Next.



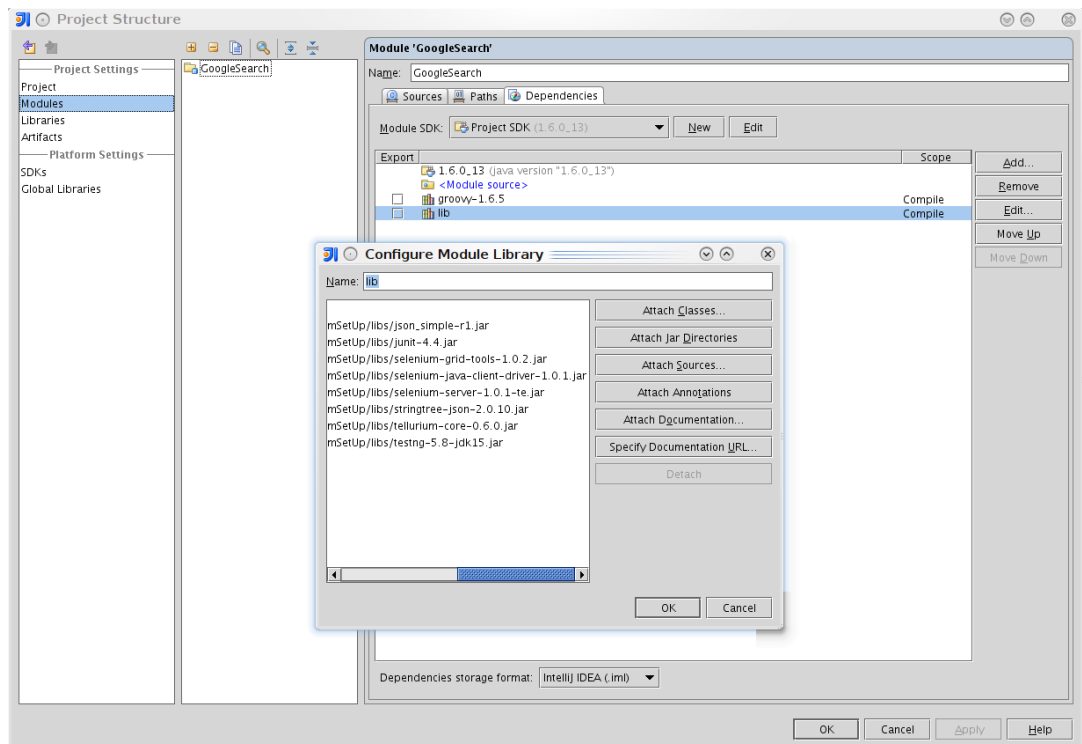
- Set the source directory and Click Next.



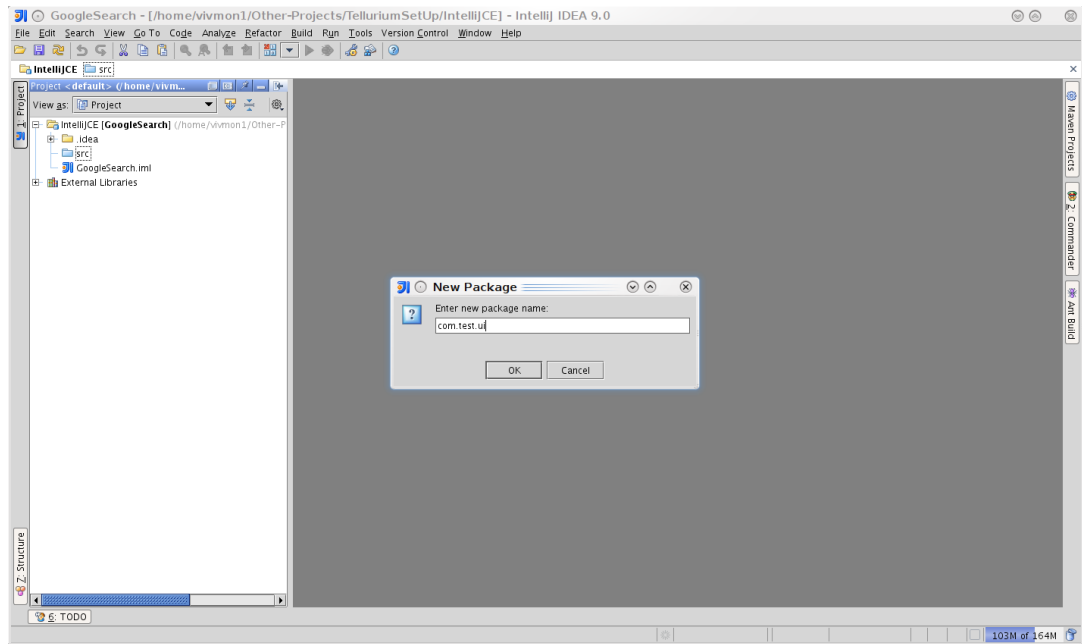
- Select the Groovy checkbox and the Groovy version. Click Next.



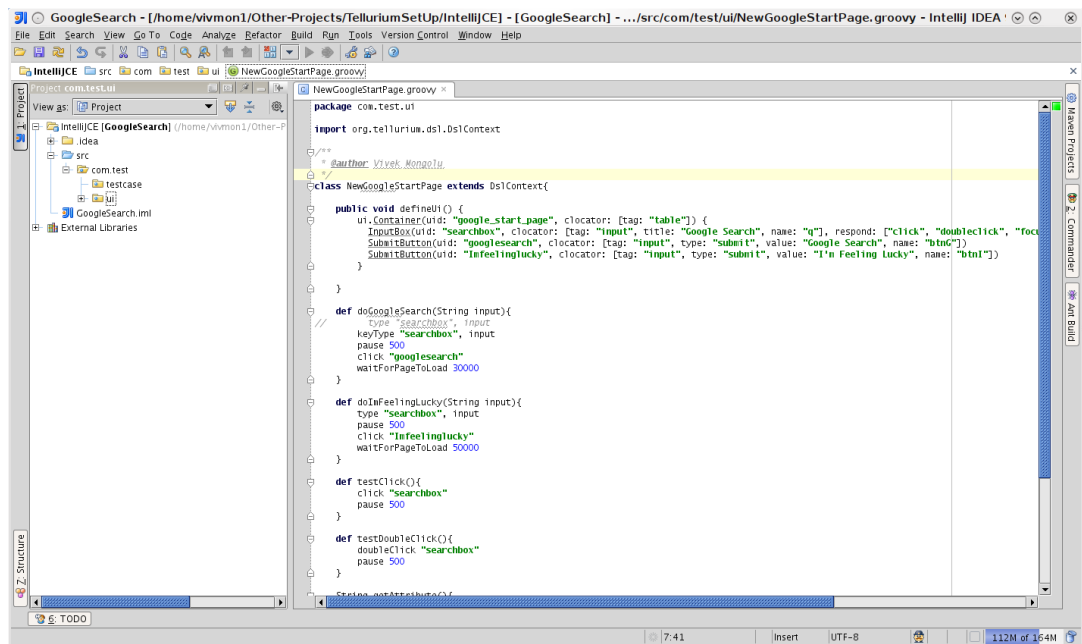
- **Project Settings > Module > GoogleSearch > Dependencies.** Create new module library named **lib** by clicking on the **Add**. Click **Attach Classes** and select the downloaded jar files from the **lib** folder as shown.



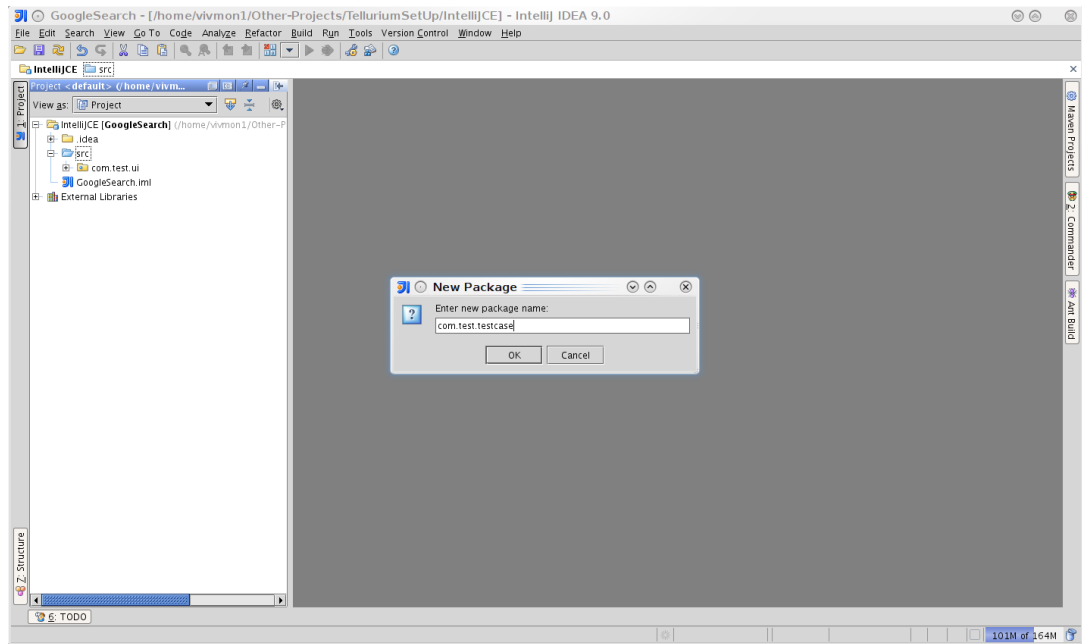
- Create a **com.test.ui** package by right clicking src -> New -> Package



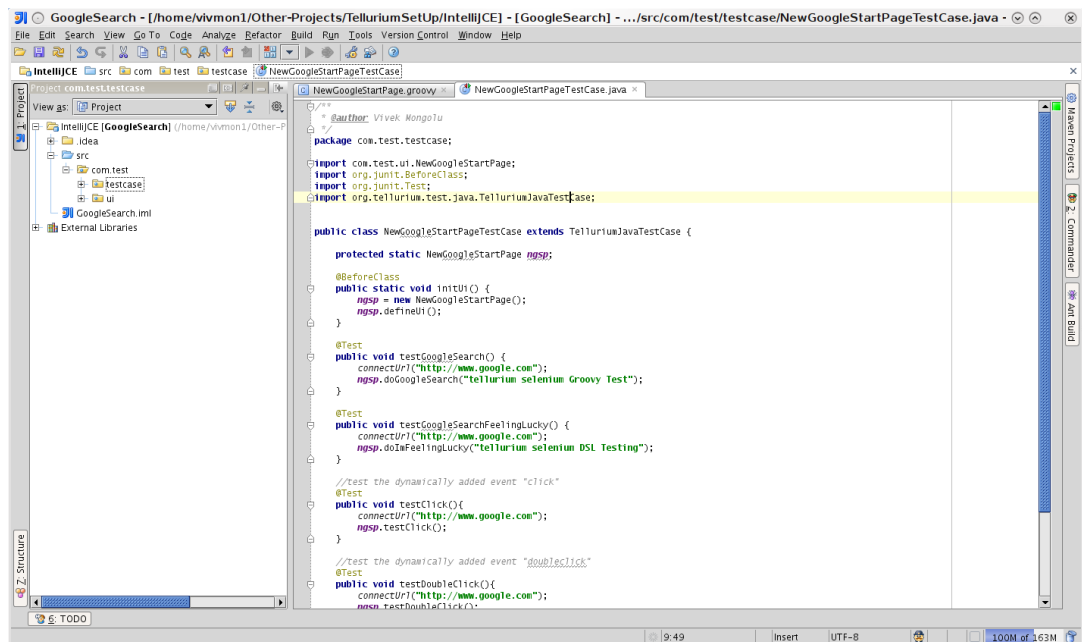
- Create a new groovy file, **NewGoogleStartPage** in **com.test.ui** package by right clicking the package New -> Groovy Class. Copy the content for this file from the download test_source.zip.



- Create a **com.test.testcase** package by right clicking src -> New -> Package

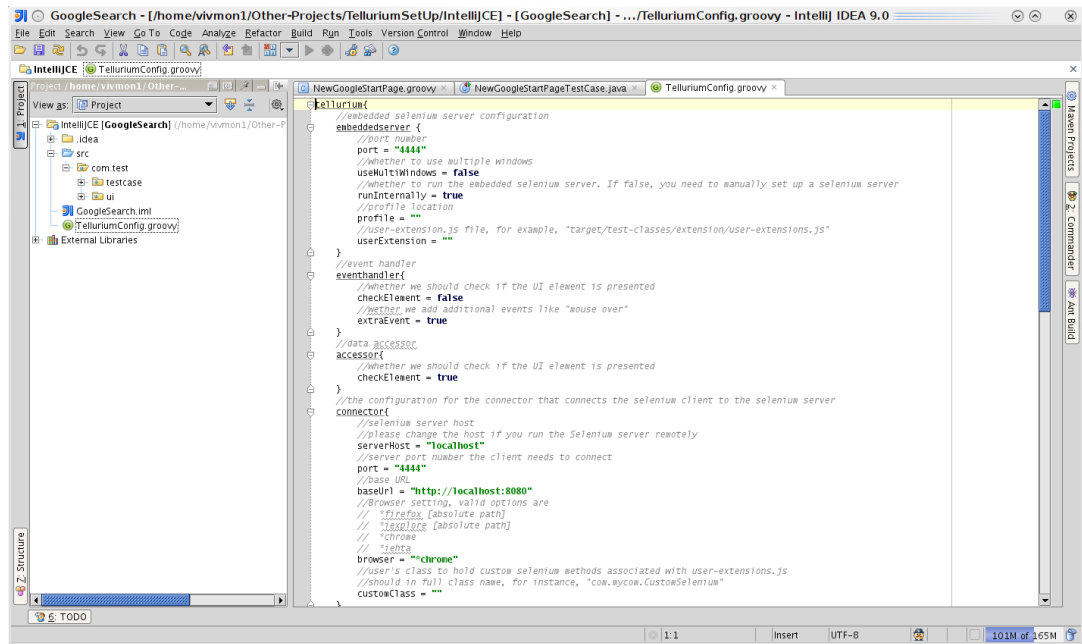


- Create a new Java file. **NewGoogleStartPageTestCase** in **com.test.ui** package by right clicking the package New -> Java Class. Copy the content for this file from the download test_source.zip.

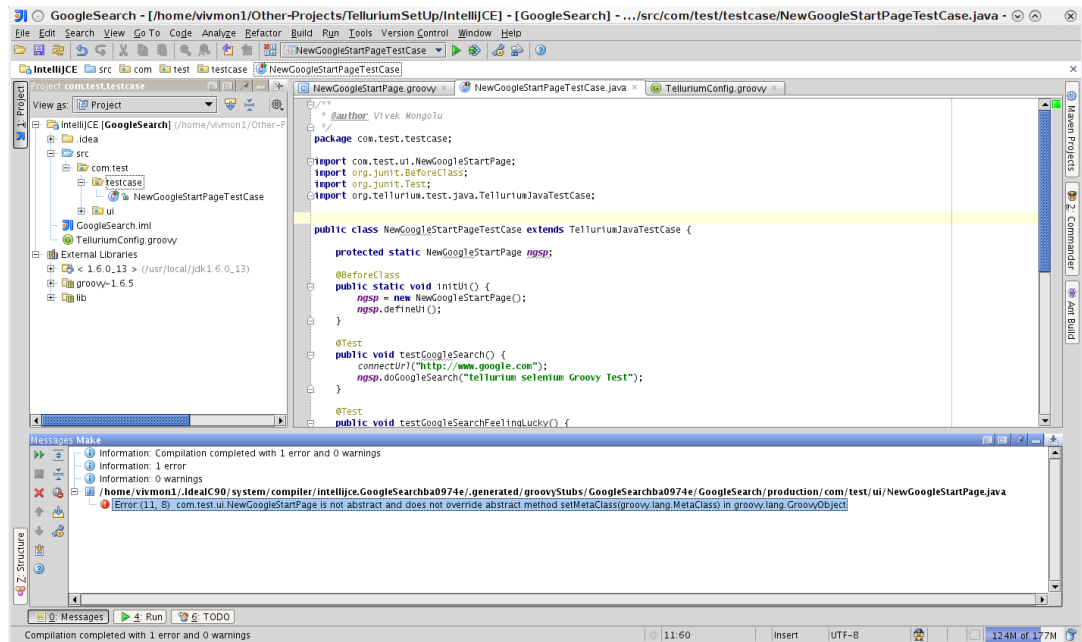


- Create a new **TelluriumConfig.groovy** file at the Project root level and not in the src directory as shown. Copy the content for this file from the download test_source.zip.

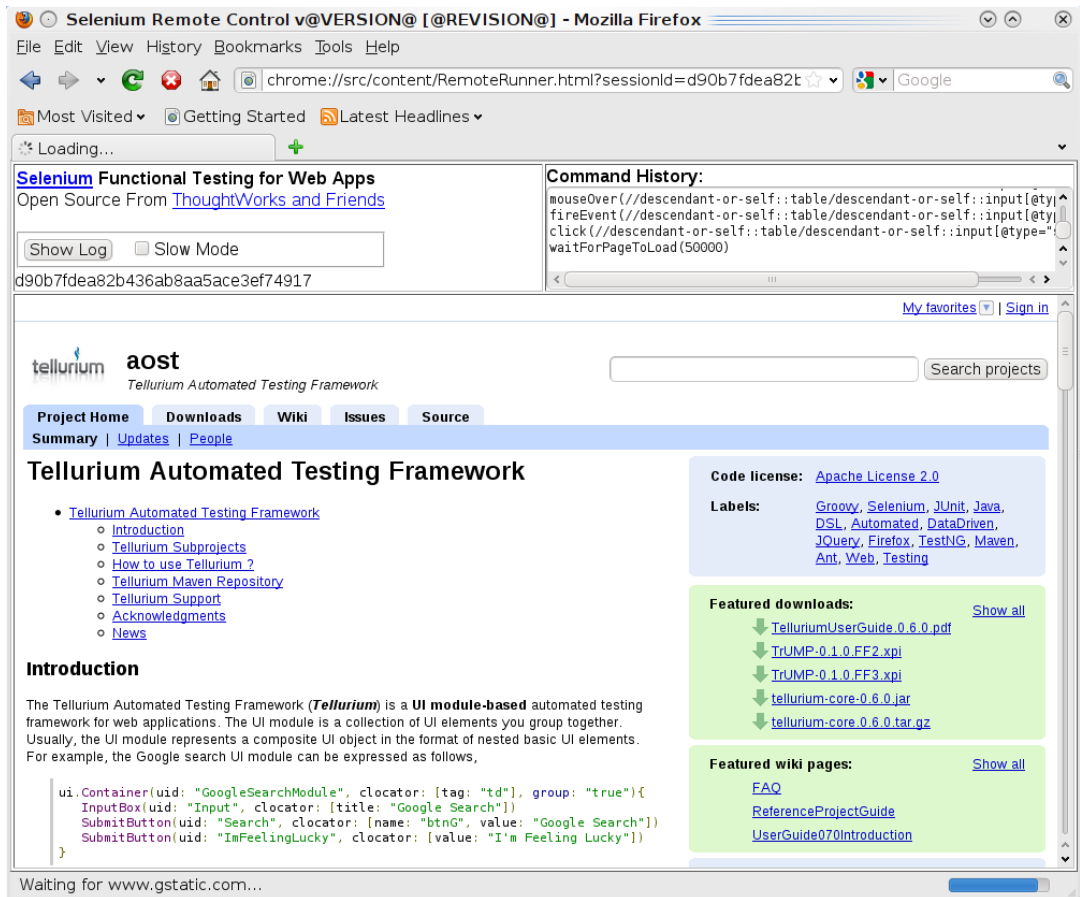
Tellurium Project Setup with IntelliJ



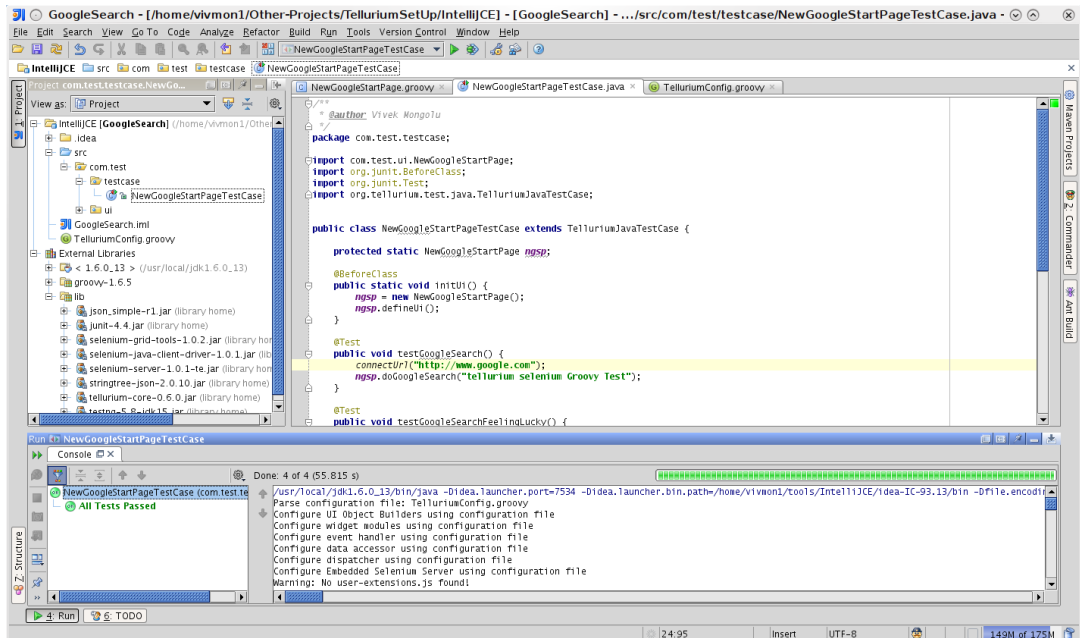
- Build the project by clicking **Build -> Make Project**. If you get an error as shown, then first compile the `NewGoogleStartPage.groovy` file and then do **Make Project**.



- Right click on the `NewGoogleStartPageTest.java` -> **RunNewGoogleStartPageTest.java**.



The test will result as passed.



Summary

This guide is for Ant users to manually create a Tellurium project. For Maven users, you can simply use Tellurium archetypes [http://code.google.com/p/aost/wiki/UserGuide070TelluriumSubprojects#Tellurium_Maven_Archetypes] to create a Tellurium project

and then load them up to IntelliJ IDEA as a Maven project. Apart from that, you can also use Tellurium reference project as a template project instead of creating everything from scratch.

Appendix C. Integration Tests with Maven Cargo Plugin

Introduction

Tellurium provides users with a way to create structured integration tests. This article introduced how to use the Maven Cargo Plugin [<http://cargo.codehaus.org/Maven2+plugin>] to automatically deploy web application and run Tellurium Integration Tests.

Example

We use the Spring Finance Manager sample application [<http://code.google.com/p/spring-finance-manager/>] as the example. We upgraded this web application to Spring 3.0 release and all the code examples are available from Tellurium project [<http://aost.googlecode.com/svn/branches/finance-manager/>].

Maven Update

To add Tellurium tests to the Spring Finance Manager sample application, we need to create the following directory to support GMaven.

```
src
### test
### groovy
#   ### org
#       ### telluriumsource
#           ### integration
#               ### FinanceManager_FuncTest.java
#               ###module
#                   ###AccountModule.groovy
#                   ###LoginModule.groovy
#                   ###MainMenu.groovy
```

We need to add the following Maven repositories to the original pom.xml for the Spring Finance Manager sample application.

```
<repository>
  <id>caja</id>
  <url>http://google-caja.googlecode.com/svn/maven</url>
</repository>
<repository>
  <id>kungfuters-public-snapshots-repo</id>
  <name>Kungfuters.org Public Snapshot Repository</name>
  <releases>
    <enabled>>false</enabled>
  </releases>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
  <url>http://maven.kungfuters.org/content/repositories/snapshots</url>
</repository>
<repository>
  <id>kungfuters-public-releases-repo</id>
  <name>Kungfuters.org Public Releases Repository</name>
  <releases>
    <enabled>true</enabled>
  </releases>
```

```
<snapshots>
  <enabled>false</enabled>
</snapshots>
<url>http://maven.kungfuters.org/content/repositories/releases</url>
</repository>
<repository>
  <id>kungfuters-thirdparty-releases-repo</id>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>false</enabled>
  </snapshots>
  <url>http://maven.kungfuters.org/content/repositories/thirdparty</url>
</repository>
<repository>
  <id>openqa-release-repo</id>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>false</enabled>
  </snapshots>
  <url>http://archiva.openqa.org/repository/releases</url>
</repository>
```

and the following dependencies,

```
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>com.springsource.org.testng</artifactId>
  <version>5.8.0</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.codehaus.gmaven</groupId>
  <artifactId>gmaven-mojo</artifactId>
  <version>${gmaven-version}</version>
  <exclusions>
    <exclusion>
      <groupId>org.codehaus.gmaven.runtime</groupId>
      <artifactId>gmaven-runtime-1.5</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>org.codehaus.gmaven.runtime</groupId>
  <artifactId>gmaven-runtime-1.6</artifactId>
  <version>${gmaven-version}</version>
</dependency>

<dependency>
  <groupId>org.telluriumsource</groupId>
  <artifactId>tellurium-core</artifactId>
  <version>${tellurium-version}</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.seleniumhq.selenium.server</groupId>
  <artifactId>selenium-server</artifactId>
  <version>${selenium-version}-${tellurium-engine-version}</version>
  <!--classifier>standalone</classifier-->
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.seleniumhq.selenium.client-drivers</groupId>
  <artifactId>selenium-java-client-driver</artifactId>
  <version>${selenium-version}</version>
  <scope>test</scope>
```

```
<exclusions>
  <exclusion>
    <groupId>org.codehaus.groovy.maven.runtime</groupId>
    <artifactId>gmaven-runtime-default</artifactId>
  </exclusion>
  <exclusion>
    <groupId>org.seleniumhq.selenium.core</groupId>
    <artifactId>selenium-core</artifactId>
  </exclusion>
  <exclusion>
    <groupId>org.seleniumhq.selenium.server</groupId>
    <artifactId>selenium-server</artifactId>
  </exclusion>
</exclusions>
</dependency>

<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-all</artifactId>
  <version>${groovy-version}</version>
</dependency>
<dependency>
  <groupId>caja</groupId>
  <artifactId>json_simple</artifactId>
  <version>rl</version>
</dependency>
<dependency>
  <groupId>org.stringtree</groupId>
  <artifactId>stringtree-json</artifactId>
  <version>2.0.10</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>3.0.1-FINAL</version>
</dependency>
```

the following resources:

```
<testResource>
  <directory>src/test/groovy</directory>
</testResource>
```

and the following Maven plugins

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.4.3</version>
  <configuration>
    <junitArtifactName>org.junit:com.springsource.org.junit</junitArtifactName>
    <testNGArtifactName>org.testng:com.springsource.org.testng</testNGArtifactName>
    <excludes>
      <exclude>**/integration/**</exclude>
    </excludes>

    <includes>
      <include>**/*Test.java</include>
    </includes>
  </configuration>
  <executions>
    <execution>
      <phase>integration-test</phase>
      <goals>
        <goal>test</goal>
      </goals>
      <configuration>
        <excludes>
          <exclude>none</exclude>
        </excludes>
      </configuration>
    </execution>
  </executions>
</plugin>
```

```
        </excludes>
        <includes>
            <include>**/integration/*_FuncTest.java</include>
        </includes>
    </configuration>
</execution>
</executions>
</plugin>

<plugin>
    <groupId>org.codehaus.gmaven</groupId>
    <artifactId>gmaven-plugin</artifactId>
    <version>${gmaven-version}</version>
    <configuration>
        <providerSelection>1.7</providerSelection>
        <targetBytecode>${java-version}</targetBytecode>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>generateStubs</goal>
                <goal>compile</goal>
                <goal>generateTestStubs</goal>
                <goal>testCompile</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

We use the following properties for Maven

```
<java-version>1.6</java-version>
<groovy-version>1.7.0</groovy-version>
<gmaven-version>1.2</gmaven-version>
<selenium-version>1.0.1</selenium-version>
<tellurium-engine-version>te2-SNAPSHOT</tellurium-engine-version>
<tellurium-version>0.7.0-SNAPSHOT</tellurium-version>
<javac-debug>true</javac-debug>
```

Tellurium Tests

Take the following Login UI module as an example.

```
package org.telluriumsource.module

import org.telluriumsource.dsl.DslContext

class LoginModule extends DslContext {

    public void defineUi() {
        ui.Container(uid: "LoginMenu", clocator: [tag: "ul"]) {
            TextBox(uid: "Security", clocator: [tag: "h2", text: "Security"])
            UrlLink(uid: "Login", clocator: [tag: "a", text: "Login", href: "/FinanceManager/login.jsp"])
            TextBox(uid: "Product", clocator: [tag: "h2", text: "Product"])
            UrlLink(uid: "List", clocator: [tag: "a", text: "List", href: "/FinanceManager/product"])
        }

        ui.Span(uid: "LoginSlide", clocator: [tag: "span", text: "Spring Security Login",
            class: "diigitTitlePaneTextNode"])

        ui.Form(uid: "Login", clocator: [tag: "form", method: "POST",
            action: "/FinanceManager/j_spring_security_check", name: "f"]) {
            TextBox(uid: "UserNameLabel", clocator: [tag: "label", text: "Email:"])
            InputBox(uid: "UserName", clocator: [tag: "input", type: "text", name: "j_username",
                class: "diigitReset", id: "j_username"])
            TextBox(uid: "PasswordLabel", clocator: [tag: "label", text: "Password:"])
            InputBox(uid: "Password", clocator: [tag: "input", type: "password", name: "j_password",
                class: "diigitReset", id: "j_password"])
        }
    }
}
```

```
        SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit", value: "Submit",
            id: "proceed"])
        Button(uid: "Reset", clocator: [tag: "input", type: "reset", value: "Reset", id: "reset"])
    }
}

public void login(String username, String password){
    keyType "Login.UserName", username
    keyType "Login.Password", password
    click "Login.Submit"
    waitForPageToLoad 30000
}

public void selectLogin(){
    click "LoginMenu.Login"
    waitForPageToLoad 30000
}

public void listProduct(){
    click "LoginMenu.List"
    waitForPageToLoad 30000
}

public void toggle(){
    click "LoginSlide"
}
}
```

We created the integration tests for the login functionality as follows.

```
public class FinanceManager_FuncTest extends TelluriumTestNGTestCase {
    private static LoginModule lnm;
    @BeforeClass
    public static void initUi() {
        lnm = new LoginModule();
        lnm.defineUi();
        connectSeleniumServer();
        useCssSelector(true);
        useTelluriumEngine(true);
        useTrace(true);
    }

    @Test
    public void testLogin(){
        connectUrl("http://localhost:8080/FinanceManager/login.jsp");
        String username = "super@admin.com";
        String password = "admin";
        lnm.login(username, password);
    }

    @AfterClass
    public static void tearDown(){
        showTrace();
    }
}
```

Cargo Maven Plugin

The cargo Maven plugin is difficult to configure and we managed to set up cargo Maven plugin to deploy the web application to an installed tomcat instance and to download and create a tomcat instance.

Assume we have a tomcat 6 installed at /usr/local/tomcat, the configuration is as follows.

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>
```

```
<artifactId>cargo-maven2-plugin</artifactId>
<configuration>
  <wait>false</wait>
  <container>
    <type>installed</type>
    <containerId>tomcat6x</containerId>
    <home>/usr/local/tomcat</home>
  </container>
  <configuration>
    <type>existing</type>
    <home>/usr/local/tomcat</home>
    <properties>
      <cargo.servlet.port>8080</cargo.servlet.port>
      <cargo.logging>high</cargo.logging>
    </properties>
  </configuration>
  <deployer>
    <type>installed</type>
    <deployables>
      <deployable>
        <groupId>net.stsmedia.financemanager</groupId>
        <artifactId>FinanceManager</artifactId>
        <type>war</type>
        <properties>
          <context>FinanceManager</context>
        </properties>
      </deployable>
    </deployables>
  </deployer>
</configuration>

<executions>
  <execution>
    <id>start-container</id>
    <phase>pre-integration-test</phase>
    <goals>
      <goal>deployer-deploy</goal>
      <goal>start</goal>
    </goals>
  </execution>
  <execution>
    <id>stop-container</id>
    <phase>post-integration-test</phase>
    <goals>
      <goal>stop</goal>
    </goals>
  </execution>
</executions>
</plugin>
```

Be aware the line `<context>FinanceManager</context>` is used to override the default context from `FinanceManager-0.5` to `FinanceManager`, otherwise the web application will break.

Seems if the `<deployer>` section is defined, we have to call `<goal>deployer-deploy</goal>` explicitly.

If we want to download and install a new Tomcat instance, the configuration looks somehow different.

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <configuration>
    <wait>false</wait>
    <container>
      <containerId>tomcat6x</containerId>

      <zipUrlInstaller>
        <url>http://www.apache.org/dist/tomcat/tomcat-6/v6.0.24/bin/apache-tomcat-6.0.24.zip</url>
        <installDir>${project.build.directory}/cargoinstalls</installDir>
      </zipUrlInstaller>
      <log>${project.build.directory}/tomcat6x/cargo.log</log>
    </container>
```

```
<configuration>
  <home>${project.build.directory}/tomcat6x/config</home>
  <properties>
    <cargo.servlet.port>8080</cargo.servlet.port>
    <cargo.logging>high</cargo.logging>
  </properties>
</configuration>
<deployer>
  <type>installed</type>
  <deployables>
    <deployable>
      <groupId>net.stsmedia.financemanager</groupId>
      <artifactId>FinanceManager</artifactId>
      <type>war</type>
      <properties>
        <context>FinanceManager</context>
      </properties>
    </deployable>
  </deployables>
</deployer>
</configuration>
<executions>
  <execution>
    <id>start-container</id>
    <phase>pre-integration-test</phase>
    <goals>
      <goal>start</goal>
      <goal>deploy</goal>
    </goals>
  </execution>
  <execution>
    <id>stop-container</id>
    <phase>post-integration-test</phase>
    <goals>
      <goal>stop</goal>
    </goals>
  </execution>
</executions>
</plugin>
```

To run the tellurium integration tests, simply run:

```
mvn install
```

or

```
mvn integration-test
```

Appendix D. Use Firebug and JQuery to Trace Problems in Tellurium Tests

From time to time, a lot of Tellurium users run into problems for their Tellurium tests and asked our Tellurium developers for help. The difficult thing is that our Tellurium developers do not have access to their their whole test code, let alone their applications. For most cases, we can only give some suggestions/hints on what the problems might be. As a result, a practical way to trace/debug Tellurium test code will benefit all our users. That is why I create this wiki page to share our experiences on how to debug Tellurium code with Firebug and jQuery.

Tellurium test mainly consists of two parts, i.e., the Java/Groovy test code and JavaScript code in our customized Selenium server. Tracing and debugging Java/Groovy code in IDE is very simple and thus we only focus on the custom Selenium server part here.

Prerequisites

Firefox Profile

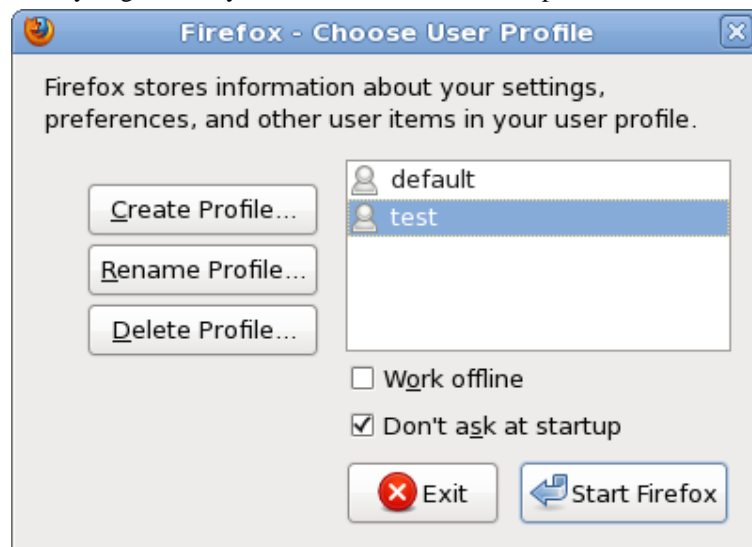
Usually, you will see a lot of posts about using the following command to manage Firefox profiles

```
firefox -ProfileManager
```

Unfortunately, this seems not working for Firefox 3.5. We have to use the following command instead

```
firefox -ProfileManager -no-remote
```

Once you get there, you can create a new Firefox profile.



Firebug Support

To add Firebug support, one way is to install the Firebug plugin to your web browser. You can get Firebug from

<https://addons.mozilla.org/en-US/firefox/addon/1843>

[<https://addons.mozilla.org/en-US/firefox/addon/1843>]

Then, use the Firefox profile in your Tellurium Tests. For example, you can add the Firefox profile in TelluriumConfig.groovy as follows,

```
tellurium{
    //embedded selenium server configuration
    embeddedserver {
        //port number
        port = "4444"
        //whether to use multiple windows
        useMultiWindows = false
        //whether to run the embedded selenium server. If false,
        //you need to manually set up a selenium server
        runInternally = true
        //profile location
        profile = "/home/jfang/.mozilla/firefox/zlduhghq.test"
        //user-extension.js file
        userExtension = "target/test-classes/extension/user-extensions.js"
    }
}
```

Or you can use the following command to specify the profile if you run the Selenium server externally,

```
[jfang@Mars ~]$ java -jar selenium-server.jar \
    -profilesLocation /home/jfang/.mozilla/firefox/zlduhghq.test
```

But sometimes, Selenium server has trouble to create a new profile from your profile and it might be better to add the Firebug plugin directly to the Selenium server. To do this, you need to following the following steps.

First, unpack the custom Selenium server

```
[jfang@Mars ~]$ jar xvf selenium-server.jar
```

You will see all the files and directories listed as follows

```
[jfang@Mars ~]$ ls -l
-rw-rw-r--. 1 jfang jfang 1677 2009-06-09 12:59 coding-conventions.txt
drwxrwxr-x. 6 jfang jfang 4096 2009-06-17 18:41 core
drwxrwxr-x. 3 jfang jfang 4096 2009-06-17 18:41 customProfileDirCUSTFF
drwxrwxr-x. 3 jfang jfang 4096 2009-08-14 16:58 customProfileDirCUSTFFCHROME
drwxrwxr-x. 3 jfang jfang 4096 2009-06-17 18:41 cybervillains
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 doctool
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 hudsuckr
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 images
-rw-rw-r--. 1 jfang jfang 1933 2009-06-09 12:59 index.html
-rw-rw-r--. 1 jfang jfang 620 2009-06-09 12:59 install-readme.txt
drwxrwxr-x. 3 jfang jfang 4096 2009-06-17 18:41 javax
drwxrwxr-x. 6 jfang jfang 4096 2009-06-17 18:41 jsunit
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 killableprocess
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 konqueror
drwxrwxr-x. 3 jfang jfang 4096 2009-06-17 18:41 META-INF
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 opera
drwxrwxr-x. 6 jfang jfang 4096 2009-06-17 18:41 org
-rw-rw-r--. 1 jfang jfang 2020 2009-06-09 12:59 readyState.xpi
-rw-rw-r--. 1 jfang jfang 129458 2009-06-09 12:59 reference.html
-rw-rw-r--. 1 jfang jfang 55 2009-06-12 15:12 selenium-ant.properties
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 sslSupport
drwxrwxr-x. 2 jfang jfang 4096 2009-06-17 18:41 strands
drwxrwxr-x. 5 jfang jfang 4096 2009-06-17 18:41 tests
drwxrwxr-x. 3 jfang jfang 4096 2009-06-17 18:41 unittest
-rw-rw-r--. 1 jfang jfang 153 2009-06-12 15:14 VERSION.txt
```

Then, copy your Firebug installed in your Firefox profile to the profiles in Selenium Server.

```
[jfang@Mars Mars]$ cp -rf
/home/jfang/.mozilla/firefox/zlduhghq.test/extensions/firebug\@software.joehewitt.com
customProfileDirCUSTFF/extensions/

[jfang@Mars Mars]$ cp -rf
/home/jfang/.mozilla/firefox/zlduhghq.test/extensions/firebug\@software.joehewitt.com
customProfileDirCUSTFFCHROME/extensions/
```

After that, re-pack the custom Selenium server

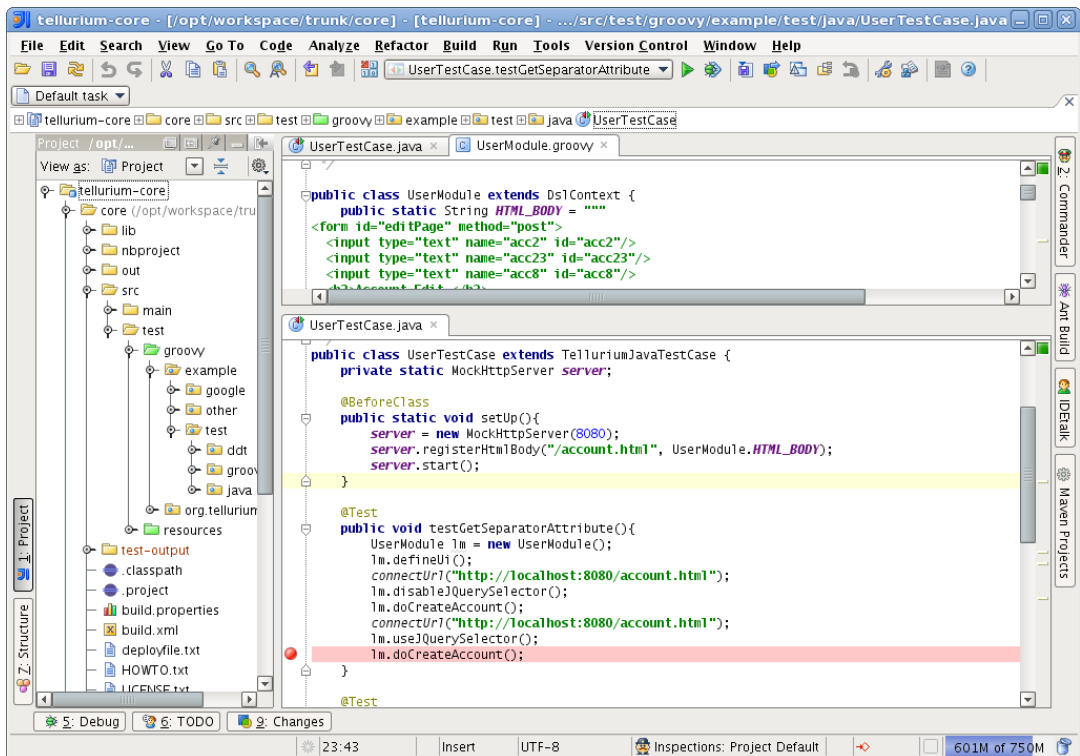
```
jar cmf META-INF/MANIFEST.MF selenium-server.jar *
```

Fortunately, you don't need to repeat the above step any more, we provide a custom Selenium server with Firebug support in our Maven repository. You should access it by using the following Maven dependency,

Debug and Trace

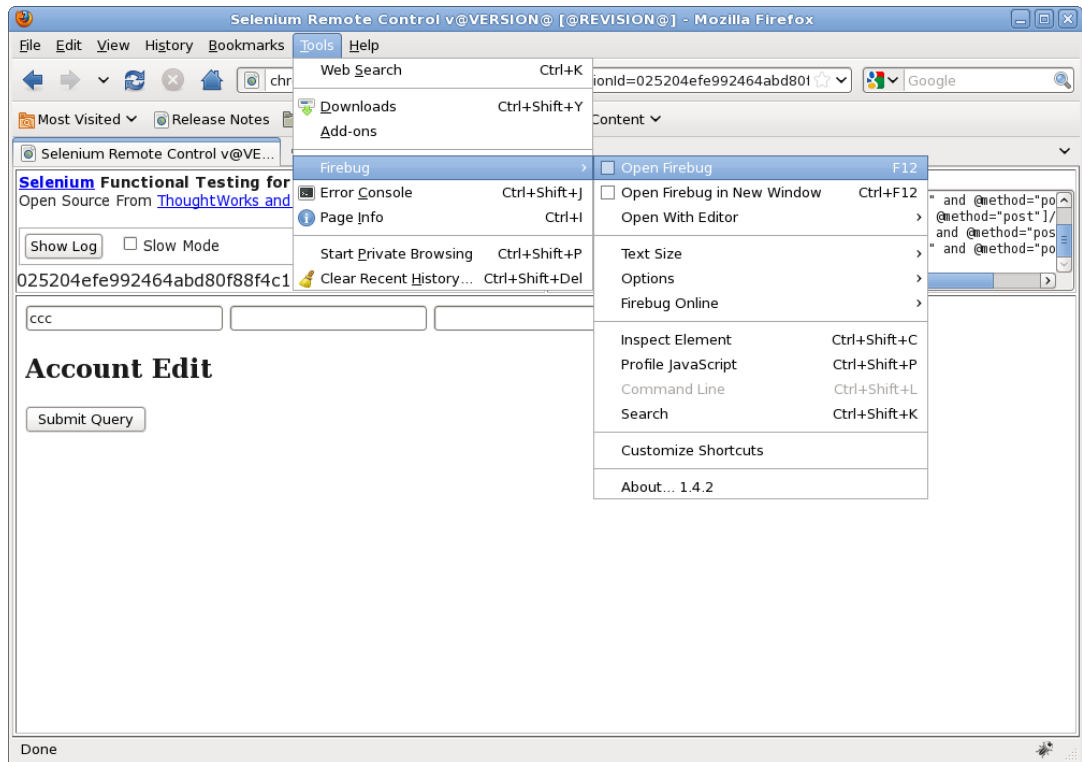
Debug JavaScript Using Firebug

First, you need to put a breakpoint in your Java/Groovy test code, for example, we put a breakpoint on the `UserTestCase` class,

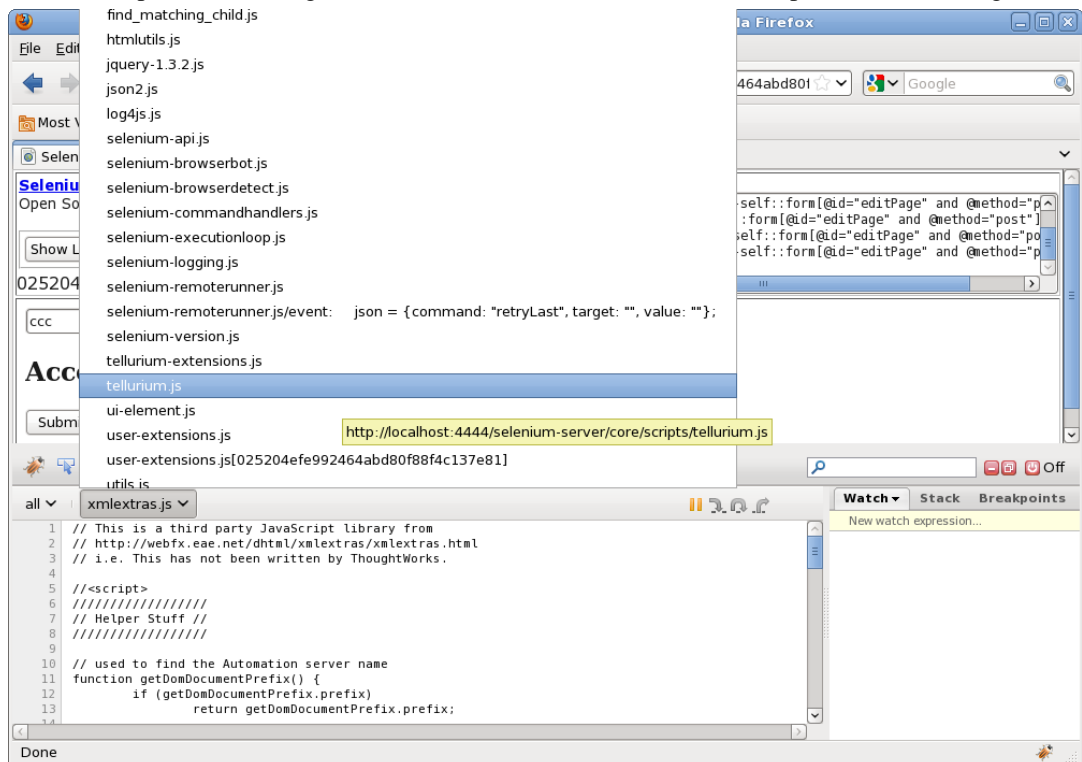


Then, use the "Debug" menu to start your test case. Once the test reaches breakpoint, you can go to the Firefox browser to open Firebug.

Use Firebug and JQuery to Trace Problems in Tellurium Tests



Sometime, the Firebug console is disabled by default, you need to enable it. After that, you can select the JavaScript files including those from Selenium core from the Javascripts menu in Firebug.



You can set a breakpoint in the JavaScript file and resume the test until it hits the breakpoint in the JavaScript file. You can find more details on how to debug Javascript from Firebug JavaScript debugging [http://getfirebug.com/js.html].

Trace Problems Using jQuery

The custom Selenium server is bundled with jQuery 1.4.2 when we added support for jQuery selector in Tellurium. We yielded the "\$" sign and also renamed *jQuery* to *tejQuery* to avoid conflicts with user's jQuery library.

To use jQuery, you need to use the single window module for the custom Selenium server, i.e., change settings in TelluriumConfig.groovy to

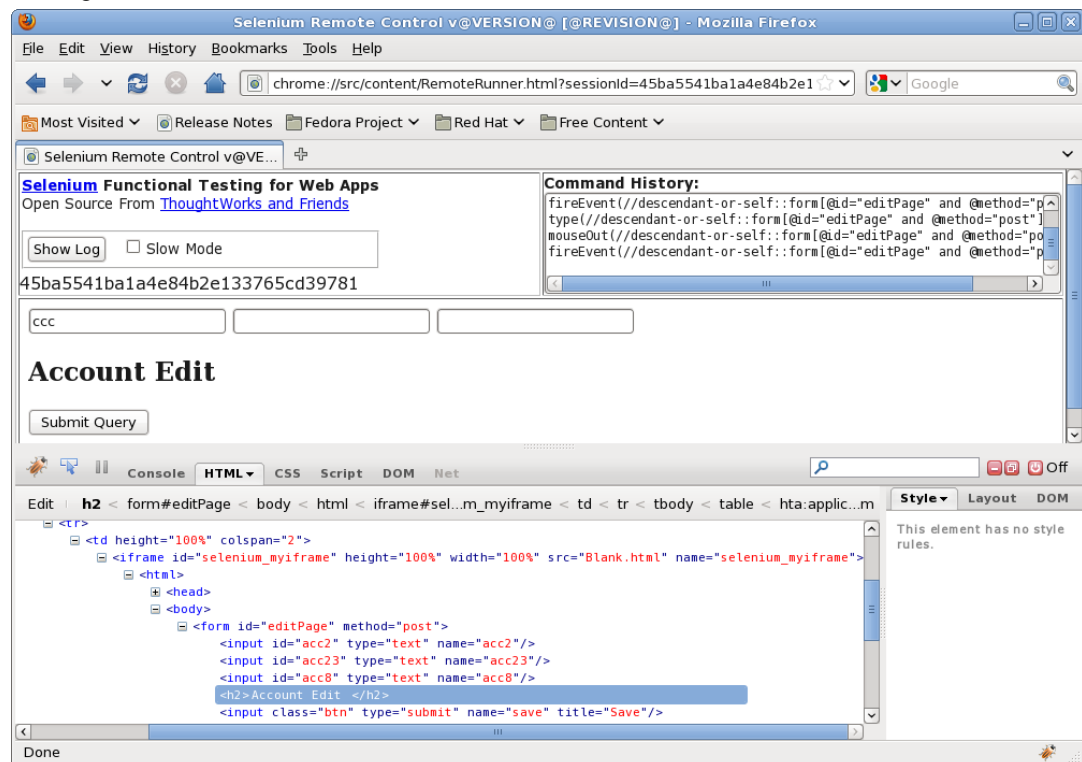
```
//whether to use multiple windows
useMultiWindows =false
```

If you run the Selenium server externally, you should use the following command to start it in a single window mode,

```
java -jar selenium-server.jar -singlewindow
```

Similarly, you need to set a breakpoint in your Java/Groovy test code so that you can work on the Firefox browser using Firebug when the test suspends.

If you open Firebug and look at the html content, you will see that your web application is actually running inside an IFrame in Selenium server shown as follows,



To access elements in the IFrame using jQuery, you need to use the following trick

```
tejQuery("#selenium_myiframe").contents().find(YOUR_JQUERY)
```

For example, we use the following jQuery to check if a button is there

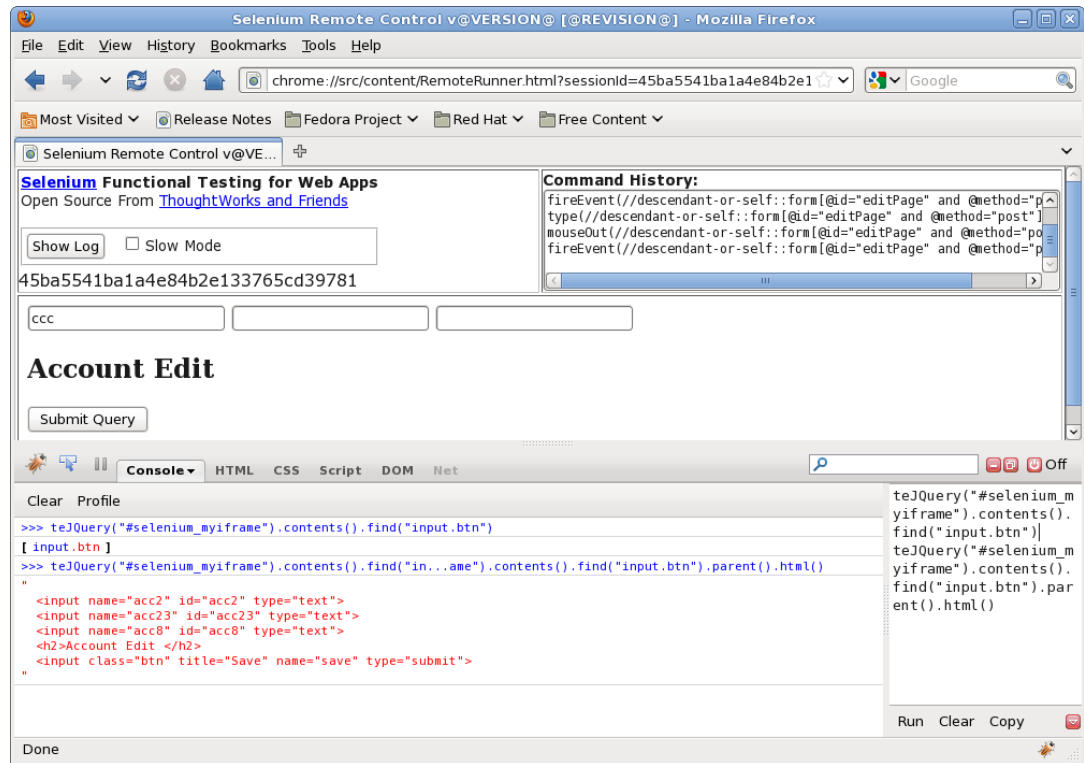
Use Firebug and JQuery to Trace Problems in Tellurium Tests

```
tejQuery("#selenium_myiframe").contents().find("input.btn")
```

We can also dump out the html source starting from the button's parent,

```
tejQuery("#selenium_myiframe").contents().find("input.btn").parent().html()
```

The output is shown as follows,



Thanks to Dominic. For multiple window mode, you can use the following way to find an element.

```
tejQuery(selenium.browserbot.getCurrentWindow().document).find("#username")
```

Appendix E. Resources

Tellurium Community

- Tellurium Project website [<http://code.google.com/p/aost/>]
- Tellurium User Group [<http://groups.google.com/group/tellurium-users>]
- Tellurium Developer Group [<http://groups.google.com/group/tellurium-developers>]
- Tellurium on Twitter [<http://twitter.com/TelluriumSource>]
- Tellurium on GitHub [<http://github.com/telluriumsource/tellurium>]
- TelluriumSource [<http://telluriumsource.org/>]
- Tellurium LinkedIn Group [<http://www.linkedin.com/groups?gid=1900807>]
- Tellurium Facebook Group [<http://www.facebook.com/group.php?gid=298577410337&ref=mf>]
- Tellurium on Reddit [<http://www.reddit.com/r/Tellurium/>]
- Tellurium on Wikipedia [http://en.wikipedia.org/wiki/Tellurium_%28software%29]

Users' Experiences

Here is the list of experiences provided by Tellurium developers and users

- Functional Testing with Tellurium by Mikhail Koryak [<http://notetodogself.blogspot.com/2009/02/functional-testing-with-tellurium.html>]
- Tellurium and test automation process by Haroon Rasheed [<http://epyramp.wordpress.com/2009/06/02/tellurium-automation-process/>]
- Tellurium: A Better Functional Test by Mikhail Koryak [<http://notetodogself.blogspot.com/2009/06/tellurium-better-functional-test.html>]
- Experience with Web Testing Frameworks by Harihara Vinayakaram [<http://startupmusings.blogspot.com/2009/06/experience-with-web-testing-frameworks.html>]
- Why I use the Tellurium framework - an Automation QA perspective by Dominic Mooney [<http://domsqablog.blogspot.com/2009/06/browser-based-testing-qa-perspective.html>]
- An Introduction to Tellurium For Web Testing by TestLabs.com [<http://blog.testlabs.com/2009/09/introduction-to-tellurium-for-web.html>]

Interviews

- Interview with Tellurium creator, Dr. Jian Fang, by Kevin Zhang from InfoQ China [<http://www.infoq.com/cn/articles/tellurium-testing-framework>].

Presentations and Videos

- Tellurium UI Module Visual Demo at Vimeo [<http://vimeo.com/9305675>]
- Tellurium at Rich Web Experience 2009 by Jian Fang and Vivek Mongolu [<http://aost.googlecode.com/files/TelluriumPresentation.pdf>] (online version [<http://www.slideshare.net/John.Jian.Fang/tellurium-at-rich-web-experience2009-2806967>])

- Tellurium was covered in the presentation Groovy Testing in Agile 2009 by Paul King and Craig Smith [http://www.slideshare.net/paulk_asert/groovy-testing-aug2009-1945995]
- Tellurium at CodeStock 2009 [<http://wiki.codestock.org/Home/2009-slides-and-code>]
- Tellurium video tutorial [http://aost.googlecode.com/files/tellurium_video_1.avi] (Online Version [<http://vimeo.com/8601173>])
- How to use TrUMP to Create Tellurium Test Cases [<http://code.google.com/p/aost/downloads/list>] (Flash Version [<http://programmingdrunk.com/flv/>])
- Tellurium Demo Video [<http://code.google.com/p/aost/downloads/list>] (Online VersionPart I [<http://vimeo.com/8598478>]) (Part II [<http://vimeo.com/8599445>])
- Online Presentation: Tellurium - A New Approach For Web Testing [<http://www.slideshare.net/John.Jian.Fang/telluriumanewapproachforwebtesting>]
- Online Presentation: 10 Minutes to Tellurium [<http://www.slideshare.net/John.Jian.Fang/ten-minutes-to-tellurium>]
- Screencast video: 10 Minutes to Tellurium [<http://aost.googlecode.com/files/TenMinutesToTellurium.ogg>] (online version [<http://vimeo.com/8600410>])

IDEs

- IntelliJ [<http://www.jetbrains.com/idea/>]
- NetBeans [<http://www.netbeans.org/>]
- Eclipse [<http://www.eclipse.org/>]

Build

- Ant [<http://ant.apache.org/>]
- Maven [<http://maven.apache.org/>]

Related

- Groovy [<http://groovy.codehaus.org/>]
- JQuery [<http://jquery.com/>]
- Dojo [<http://www.dojotoolkit.org/>]
- Ext JS [<http://extjs.com/products/extjs/>]
- Selenium [<http://seleniumhq.org/>]
- Selenium Grid [<http://selenium-grid.seleniumhq.org/>]
- Selenium Community [<http://clearspace.openqa.org/community/selenium>]
- Canoo WebTest [<http://webtest.canoo.com/>]
- Twill [<http://twill.idyll.org/>]
- JUnit [<http://www.junit.org/>]
- TestNG [<http://testng.org/>]

Appendix F. Sample Maven Configurations

settings.xml

```
<settings>
  <!-- comment out the proxy section if you connect directly to Internet-->
  <proxies>
    <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>localhost</host>
      <port>5865</port>
      <!--username>proxyuser</username>
      <password>someword</password-->
      <nonProxyHosts></nonProxyHosts>
    </proxy>
  </proxies>
  <profiles>
    <profile>
      <id>tellurium</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>kungfuters-public-snapshots-repo</id>
          <name>Kungfuters.org Public Snapshot Repository</name>
          <releases>
            <enabled>>false</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
          <url>http://kungfuters.org/nexus/content/repositories/snapshots</url>
        </repository>
        <repository>
          <id>kungfuters-public-releases-repo</id>
          <name>Kungfuters.org Public Releases Repository</name>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
          </snapshots>
          <url>http://kungfuters.org/nexus/content/repositories/releases</url>
        </repository>
        <repository>
          <id>kungfuters-thirdparty-releases-repo</id>
          <name>Kungfuters.org Third Party Releases Repository</name>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
          </snapshots>
          <url>http://kungfuters.org/nexus/content/repositories/thirdparty</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
</settings>
```

Sample Tellurium Project Maven POM


```

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>Your_Group_ID</groupId>
  <artifactId>Your_Artifact_Id</artifactId>
  <version>Your_Version</version>
  <name>Tellurium JUnit Test Project</name>

  <repositories>
    <repository>
      <id>caja</id>
      <url>http://google-caja.googlecode.com/svn/maven</url>
    </repository>
    <repository>
      <id>kungfuters-public-snapshots-repo</id>
      <name>Kungfuters.org Public Snapshot Repository</name>
      <releases>
        <enabled>false</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
      <url>http://maven.kungfuters.org/content/repositories/snapshots</url>
    </repository>
    <repository>
      <id>kungfuters-public-releases-repo</id>
      <name>Kungfuters.org Public Releases Repository</name>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <url>http://maven.kungfuters.org/content/repositories/releases</url>
    </repository>
    <repository>
      <id>kungfuters-thirdparty-releases-repo</id>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <url>http://maven.kungfuters.org/content/repositories/thirdparty</url>
    </repository>
    <repository>
      <id>openqa-release-repo</id>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <url>http://archiva.openqa.org/repository/releases</url>
    </repository>
  </repositories>

  <dependencies>
    <dependency>
      <groupId>org.codehaus.gmaven</groupId>
      <artifactId>gmaven-mojo</artifactId>
      <version>${gmaven-version}</version>
      <exclusions>
        <exclusion>
          <groupId>org.codehaus.gmaven.runtime</groupId>
          <artifactId>gmaven-runtime-1.5</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.codehaus.gmaven.runtime</groupId>
      <artifactId>gmaven-runtime-1.6</artifactId>
      <version>${gmaven-version}</version>
    </dependency>
  </dependencies>
</project>

```

```

</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.7</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.telluriumsource</groupId>
  <artifactId>tellurium-core</artifactId>
  <version>${tellurium-version}</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.telluriumsource</groupId>
  <artifactId>tellurium-udl</artifactId>
  <version>${tellurium-version}</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.seleniumhq.selenium.server</groupId>
  <artifactId>selenium-server</artifactId>
  <version>${selenium-server-version}-${tellurium-engine-version}</version>
  <!--classifier>standalone</classifier-->
</dependency>
<dependency>
  <groupId>org.seleniumhq.selenium.client-drivers</groupId>
  <artifactId>selenium-java-client-driver</artifactId>
  <version>${selenium-version}</version>
  <exclusions>
    <exclusion>
      <groupId>org.codehaus.groovy.maven.runtime</groupId>
      <artifactId>gmaven-runtime-default</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.seleniumhq.selenium.core</groupId>
      <artifactId>selenium-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.seleniumhq.selenium.server</groupId>
      <artifactId>selenium-server</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-all</artifactId>
  <version>${groovy-version}</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>caja</groupId>
  <artifactId>json_simple</artifactId>
  <version>rl</version>
</dependency>
<dependency>
  <groupId>org.stringtree</groupId>
  <artifactId>stringtree-json</artifactId>
  <version>2.0.10</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>3.0.1-FINAL</version>
</dependency>
</dependencies>

<build>
  <resources>
    <resource>
      <directory>src/main/groovy</directory>
    </resource>
    <resource>
      <directory>src/main/resources</directory>
    </resource>
  </resources>
  <testResources>

```

```

<testResource>
  <directory>src/test/groovy</directory>
</testResource>
<testResource>
  <directory>src/test/resources</directory>
</testResource>
</testResources>

<pluginManagement>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.0.2</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.4.3</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>2.4.3</version>
    </plugin>
    <plugin>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.2</version>
    </plugin>
    <plugin>
      <artifactId>maven-source-plugin</artifactId>
      <version>2.0.4</version>
    </plugin>
    <plugin>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>2.4</version>
    </plugin>
    <plugin>
      <artifactId>maven-site-plugin</artifactId>
      <version>2.0-beta-7</version>
    </plugin>
    <plugin>
      <groupId>org.codehaus.gmaven</groupId>
      <artifactId>gmaven-plugin</artifactId>
      <version>${gmaven-version}</version>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jxr-plugin</artifactId>
      <version>2.1</version>
    </plugin>
  </plugins>
</pluginManagement>

<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
      <source>${java-version}</source>
      <target>${java-version}</target>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
      <includes>
        <include>**/*_UT.java</include>
        <include>**/*Testcase.java</include>
      </includes>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-jar-plugin</artifactId>
    <executions>
      <execution>
        <goals>
          <goal>test-jar</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>

```

```

</plugin>
<plugin>
  <artifactId>maven-source-plugin</artifactId>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>jar</goal>
        <goal>test-jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <artifactId>maven-javadoc-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <groupId>org.codehaus.gmaven</groupId>
  <artifactId>gmaven-plugin</artifactId>
  <configuration>
    <providerSelection>1.7</providerSelection>
    <targetBytecode>${java-version}</targetBytecode>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>generateStubs</goal>
        <goal>compile</goal>
        <goal>generateTestStubs</goal>
        <goal>testCompile</goal>
      </goals>
    </execution>
  </executions>
</plugin>
</plugins>
</build>
<reporting>
  <plugins>
    <plugin>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <reportSets>
        <reportSet>
          <reports>
            <report>report-only</report>
          </reports>
        </reportSet>
      </reportSets>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jxr-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
<properties>
  <java-version>1.6</java-version>
  <groovy-version>1.7.0</groovy-version>
  <gmaven-version>1.2</gmaven-version>
  <selenium-version>1.0.1</selenium-version>
  <selenium-server-version>1.0.1</selenium-server-version>
  <tellurium-engine-version>te3</tellurium-engine-version>
  <tellurium-version>0.7.0</tellurium-version>
  <javac-debug>true</javac-debug>
</properties>
</project>

```

Appendix G. Sample Tellurium Project Configuration

TelluriumConfig.groovy

TelluriumConfig.groovy is Tellurium default configuration file and it can be loaded from project root or classpath

```
tellurium{
    //embedded selenium server configuration
    embeddedserver {
        //port number
        port = "4444"
        //whether to use multiple windows
        useMultiWindows = false
        //whether to trust all SSL certs, i.e., option "--trustAllSSLCertificates"
        trustAllSSLCertificates = true
        //whether to run the embedded selenium server. If false, you need to manually set up a selenium server
        runInternally = true
        //By default, Selenium proxies every browser request; set this flag to make the browser use proxy only
        //for URLs containing '/selenium-server'
        avoidProxy = false
        //stops re-initialization and spawning of the browser between tests
        browserSessionReuse = false
        //enabling this option will cause all user cookies to be archived before launching IE, and restored
        //after IE is closed.
        ensureCleanSession = false
        //debug mode, with more trace information and diagnostics on the console
        debugMode = false
        //interactive mode
        interactive = false
        //an integer number of seconds before we should give up
        timeoutInSeconds = 30
        //profile location
        profile = ""
        //user-extension.js file
        userExtension = ""
    }
    //event handler
    eventhandler{
        //whether we should check if the UI element is presented
        checkElement = false
        //whether we add additional events like "mouse over"
        extraEvent = false
    }
    //data accessor
    accessor{
        //whether we should check if the UI element is presented
        checkElement = true
    }
    //the bundling tier
    bundle{
        maxMacroCmd = 5
        useMacroCommand = false
    }
    //the configuration for the connector that connects the selenium client to the selenium server
    connector{
        //selenium server host
        //please change the host if you run the Selenium server remotely
        serverHost = "localhost"
        //server port number the client needs to connect
        port = "4444"
        //base URL
        baseUrl = "http://localhost:8080"
        //Browser setting, valid options are
        // *firefox [absolute path]
        // *iexplore [absolute path]
        // *chrome
        // *iehta
        browser = "*chrome"
        //user's class to hold custom selenium methods associated with user-extensions.js
        //should in full class name, for instance, "com.mycom.CustomSelenium", org.tellurium.test.MyCommand
        customClass = ""
        //browser options such as
        // options = "captureNetworkTraffic=true, addCustomRequestHeader=true"
        options = ""
    }
}
datadriven{
```

Sample Tellurium Project Configuration

```
    dataprovider{
        //specify which data reader you like the data provider to use
        //the valid options include "PipeFileReader", "CSVFileReader" at this point
        reader = "PipeFileReader"
    }
}
//this section allows users to define the internationalization required
//if this section is removed, we take the default locale
//from the system
//enter only one locale at a time, and use this only if you want to explicitly
//set the locale, preferable way is to comment out this section
i18n{
    //locale = "fr_FR"
    locale = "en_US"
}
test{
    execution{
        //whether to trace the execution timing
        trace = false
    }
    //at current stage, the result report is only for tellurium data driven testing
    //we may add the result report for regular tellurium test case
    result{
        //specify what result reporter used for the test result
        //valid options include "SimpleResultReporter", "XMLResultReporter", and "StreamXMLResultReporter"
        reporter = "XMLResultReporter"
        //the output of the result
        //valid options include "Console", "File" at this point
        //if the option is "File", you need to specify the file name, other wise it will use the default
        //file name "TestResults.output"
        output = "Console"
        //test result output file name
        filename = "TestResult.output"
    }
    exception{
        //whether Tellurium captures the screenshot when exception occurs.
        //Note that the exception is the one thrown by Selenium Server
        //we do not care the test logic errors here
        captureScreenshot = true
        //we may have a series of screenshots, specify the file name pattern here
        //Here the ? will be replaced by the timestamp and you might also want to put
        //file path in the file name pattern
        filenamePattern = "Screenshot?.png"
    }
}
}
uiobject{
    builder{
        //user can specify custom UI objects here by define the builder for each UI object
        //the custom UI object builder must extend UiObjectBuilder class
        //and implement the following method:
        //
        // public build(Map map, Closure c)
        //
        //For container type UI object, the builder is a bit more complicated, please
        //take the TableBuilder or ListBuilder as an example

        //example:
        Icon="org.tellurium.builder.IconBuilder"
    }
}
widget{
    module{
        //define your widget modules here, for example Dojo or ExtJs
        included="dojo, extjs"
        included=""
    }
}
}
```

JSON String

Tellurium can also loaded configuration from a JSON String. The JSON String in pretty format is as follows.

```
{
  "tellurium": {
    "test": {
      "result": {
        "reporter": "XMLResultReporter",
```

```
        "filename": "TestResult.output",
        "output": "Console"
    },
    "exception": {
        "filenamePattern": "Screenshot?.png",
        "captureScreenshot": false
    },
    "execution": {
        "trace": false
    }
},
"accessor": {
    "checkElement": false
},
"embeddedserver": {
    "port": "4444",
    "browserSessionReuse": false,
    "debugMode": false,
    "ensureCleanSession": false,
    "interactive": false,
    "avoidProxy": false,
    "timeoutInSeconds": 30,
    "runInternally": true,
    "trustAllSSLCertificates": true,
    "useMultiWindows": false,
    "userExtension": "",
    "profile": ""
},
"uiobject": {
    "builder": { }
},
"eventhandler": {
    "checkElement": false,
    "extraEvent": false
},
"i18n": {
    "locale": "en_US"
},
"connector": {
    "baseUrl": "http://localhost:8080",
    "port": "4444",
    "browser": "*chrome",
    "customClass": "",
    "serverHost": "localhost",
    "options": ""
},
"bundle": {
    "maxMacroCmd": 5,
    "useMacroCommand": true
},
"datadriven": {
    "dataprovider": {
        "reader": "PipeFileReader"
    }
},
"widget": {
    "module": {
        "included": ""
    }
}
}
```

Appendix H. Sample Ant Build Script

build.properties

```
### Change javahome to your own Java Home ###
javahome = /usr/java/current

##### javac settings #####
# For javac -- can leave alone in most cases
javac.compiler=javac1.5
javac.deprecation = on
javac.optimize = on
javac.debug = true
javac.fork=no

dist.dir = ${basedir}/dist
```

build.xml

```
<?xml version="1.0"?>

<project name="tellurium-testng" default="compile-test" basedir=".">

    <property name="dir.project" value="${basedir}" />

    <property file="build.properties" />

    <property name="dir.source" value="${dir.project}/src" />
    <property name="dir.source.tellurium" value="${dir.source}/main/groovy" />
    <property name="dir.source.test" value="${dir.source}/test/groovy" />

    <property name="dir.build" value="${dir.project}/out" />
    <property name="dir.build.tellurium" value="${dir.build}/production" />
    <property name="dir.build.test" value="${dir.build}/test" />

    <property name="dir.lib" value="${dir.project}/lib" />

    <path id="lib.path">
        <fileset dir="${dir.lib}">
            <include name="*.jar" />
            <exclude name="*-src.jar" />
            <include name="*.class" />
        </fileset>
    </path>

    <!-- Match runtime libraries -->
    <patternset id="pattern.libs">
        <include name="**/*.jar" />
        <exclude name="**/*-src.jar"/>
        <!--exclude name="**/*junit.jar"/-->
    </patternset>

    <path id="junit.classpath">
        <fileset dir="${dir.lib}">
            <include name="junit*.jar"/>
        </fileset>
    </path>

    <path id="tellurium.classpath">
        <path refid="lib.path" />
        <pathelement location="${dir.build.tellurium}" />
    </path>

    <path id="test.classpath">
        <path refid="tellurium.classpath" />
```



```

        <path refid="junit.classpath" />
        <pathelement location="${dir.build.test}" />
    </path>

    <path id="project.classpath">
        <path refid="test.classpath" />
    </path>

    <taskdef name="junit"
        classname="org.apache.tools.ant.taskdefs.optional.junit.JUnitTask"
        classpathref="junit.classpath" />

    <target name="clean">
        <echo message="Cleaning ..." />
        <delete dir="${dir.build.tellurium}" />
        <delete dir="${dir.build.test}" />
    </target>

    <target name="init">
        <echo message="Initializing project..." />
        <tstamp>
            <format property="time.formatted" pattern="MM/dd/yyyy hh:mm:ss a"
                unit="hour" />
        </tstamp>
        <mkdir dir="${dir.build}" />
        <mkdir dir="${dir.build.tellurium}" />
        <mkdir dir="${dir.build.test}" />
    </target>

    <macrodef name="compile-java">
        <!-- required attributes -->
        <attribute name="srcdir" />
        <attribute name="destdir" />
        <attribute name="excludes" default="" />

        <!-- these defaults can be changed using properties -->
        <attribute name="compiler" default="${javac.compiler}" />
        <attribute name="debug" default="${javac.debug}" />
        <attribute name="optimize" default="${javac.optimize}" />
        <attribute name="deprecation" default="${javac.deprecation}" />
        <attribute name="fork" default="${javac.fork}" />

        <!-- these defaults can only be overridden explicitly by a task -->
        <attribute name="encoding" default="UTF-8" />
        <attribute name="includeAndRunTime" default="no" />
        <attribute name="failonerror" default="false" />

        <!-- this element sucks up all elements when the macro is used -->
        <element name="javac-elements" implicit="yes" />

        <!-- the macro body -->
        <sequential>
            <javac srcdir="@{srcdir}"
                excludes="@{excludes}"
                destdir="@{destdir}" compiler="@{compiler}"
                debug="@{debug}"
                optimize="@{optimize}"
                deprecation="@{deprecation}"
                fork="@{fork}"
                encoding="@{encoding}" failonerror="@{failonerror}" />
        </javac>
        </sequential>
    </macrodef>

    <taskdef name="groovyc" classname="org.codehaus.groovy.ant.Groovyc"
        classpathref="lib.path"/>

    <target name="compile-tellurium" depends="init">
        <echo message="Compiling java..." />
        <groovyc srcdir="${dir.source.tellurium}"
            destdir="${dir.build.tellurium}"
            <classpath refid="lib.path" />
        <!--javac source="1.5" target="1.5" debug="on" /-->
    </groovyc>
    <javac srcdir="${dir.source.tellurium}"
        destdir="${dir.build.tellurium}"
        <classpath refid="tellurium.classpath" />

```

```
</javac>
</target>

<target name="compile-test" depends="clean, compile-tellurium">
  <echo message="Compiling test.." />
  <groovyc srcdir="${dir.source.test}" destdir="${dir.build.test}">
    <classpath refid="tellurium.classpath" />
    <include name="**" />
  </groovyc>
  <javac srcdir="${dir.source.test}"
    destdir="${dir.build.test}">
    <classpath refid="test.classpath" />
  </javac>
</target>

<target name="run-unit-tests" depends="compile-test">
  <junit fork="yes" forkmode="once" maxmemory="1024m"
    printsummary="yes" errorProperty="test.failed"
    failureProperty="test.failed">
    <classpath refid="project.classpath" />
    <formatter type="brief" usefile="false" />
    <formatter type="xml" />
    <batchtest todir="${dir.build.test}">
      <fileset dir="${dir.source.test}">
        <include name="**/org/tellurium/test/**"/>
      </fileset>
      <include name="**/org/tellurium/ddt/**"/>
    </batchtest>
  </junit>
  <fail if="test.failed" />
</target>

</project>
```

Appendix I. Sample Run DSL Script

Sample Groovy Grape Configuration

```
<ivysettings>
  <settings defaultResolver="downloadGrapes"/>
  <property
    name="local-maven2-pattern"
    value="${user.home}/.m2/repository/[organisation]/[module]/[revision]/
      [module]-[revision](-[classifier]).[ext]"
    override="false" />
  <resolvers>
    <chain name="downloadGrapes">
      <filesystem name="cachedGrapes">
        <ivy pattern="${user.home}/.groovy/grapes/[organisation]/[module]/ivy-[revision].xml"/>
        <artifact pattern="${user.home}/.groovy/grapes/[organisation]/[module]/
          [type]s/[artifact]-[revision].[ext]"/>
      </filesystem>
      <filesystem name="local-maven-2" m2compatible="true" local="true">
        <ivy pattern="${local-maven2-pattern}"/>
        <artifact pattern="${local-maven2-pattern}"/>
      </filesystem>
      <!-- todo add 'endorsed groovy extensions' resolver here -->
      <ibiblio name="kungfuters.3rdparty"
        root="http://maven.kungfuters.org/content/repositories/thirdparty/"
        m2compatible="true"/>
      <ibiblio name="codehaus" root="http://repository.codehaus.org/"
        m2compatible="true"/>
      <ibiblio name="ibiblio" m2compatible="true"/>
      <ibiblio name="java.net2" root="http://download.java.net/maven/2/"
        m2compatible="true"/>
      <ibiblio name="openqa" root="http://archiva.openqa.org/repository/releases/"
        m2compatible="true"/>
      <ibiblio name="kungfuters.snapshot"
        root="http://maven.kungfuters.org/content/repositories/snapshots/"
        m2compatible="true"/>
      <ibiblio name="kungfuters.release"
        root="http://maven.kungfuters.org/content/repositories/releases/"
        m2compatible="true"/>
    </chain>
  </resolvers>
</ivysettings>
```

rundsl.groovy

```
import groovy.grape.Grape;

Grape.grab(group:'org.telluriumsource', module:'tellurium-core', version:'0.7.0',
  classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.stringtree', module:'stringtree-json', version:'2.0.10',
  classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'caja', module:'json_simple', version:'r1',
  classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.seleniumhq.selenium.server', module:'selenium-server', version:'1.0.1-te3',
  classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.seleniumhq.selenium.client-drivers', module:'selenium-java-client-driver',
  version:'1.0.1', classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'org.apache.poi', module:'poi', version:'3.0.1-FINAL',
  classLoader:this.class.classLoader.rootLoader)
Grape.grab(group:'junit', module:'junit', version:'4.7',
  classLoader:this.class.classLoader.rootLoader)

import org.telluriumsource.dsl.DslScriptExecutor

@Grapes([
  @Grab(group='org.codehaus.groovy', module='groovy-all', version='1.7.0'),
```

```
@Grab(group='org.seleniumhq.selenium.server', module='selenium-server', version='1.0.1-te3'),
@Grab(group='org.seleniumhq.selenium.client-drivers', module='selenium-java-client-driver',
      version='1.0.1'),
@Grab(group='junit', module='junit', version='4.7'),
@Grab(group='caja', module='json_simple', version='rl'),
@Grab(group='org.apache.poi', module='poi', version='3.0.1-FINAL'),
@Grab(group='org.stringtree', module='stringtree-json', version='2.0.10'),
@Grab(group='org.telluriumsource', module='tellurium-core', version='0.7.0')
])

def runDsl(String[] args) {
    def cli = new CliBuilder(usage: 'rundsdl.groovy -[hf] [scriptname]')
    cli.with {
        h longOpt: 'help', 'Show usage information'
        f longOpt: 'scriptname', 'DSL script name'
    }
    def options = cli.parse(args)
    if (!options) {
        return
    }
    if (options.h) {
        cli.usage()
        return
    }
    if (options.f) {
        def extraArguments = options.arguments()
        if (extraArguments) {
            extraArguments.each {String name ->
                def input = [name].toArray(new String[0])
                DslScriptExecutor.main(input)
            }
        }
    }
}

println "Running DSL test script, press Ctrl+C to stop."

runDsl(args)
```