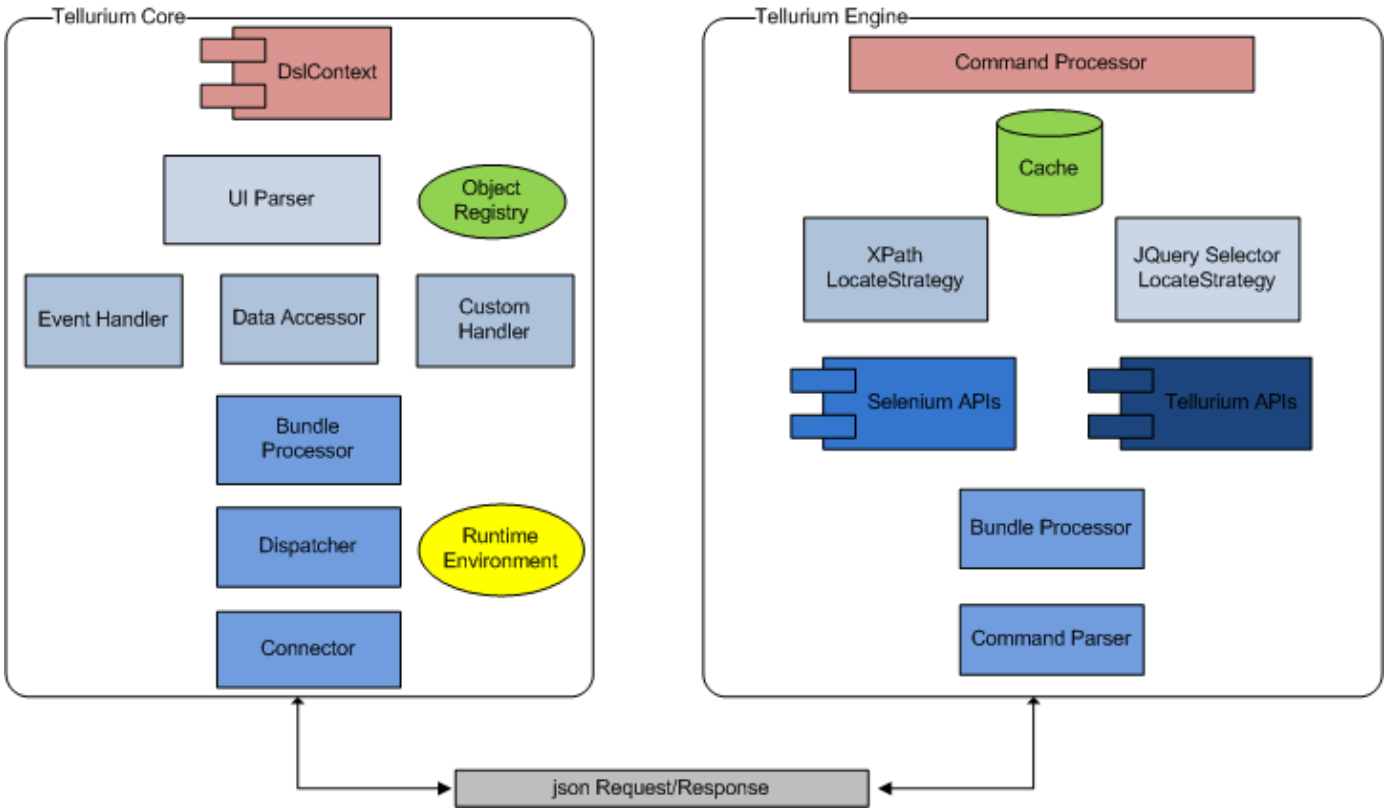
Table of Contents

Table of Contents

Introduction

Tellurium 0.7.0 RC1 release is out now. Over 150 issues have been closed. The issues include new features, bugs fixings, and other enhancements requested by users. There are some fundamental changes in Tellurium 0.7.0 compared with Tellurium 0.6.0 such as the group locating algorithm, UI module caching, Macro command, jQuery-based new APIs, and i18n support. To make your life easier, We list all the changes here so that you can update your existing Tellurium testing project accordingly.

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



The main changes include

- The bundle tier to support Macro Command
- New Tellurium API in Engine
- UI Module based Caching
- Runtime Environment

The details are covered in the subsequent sections.

Changes in Tellurium Core

Package Name

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 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

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.

Group Locating has some fundamental impacts on Tellurium and this can be explained by an example.

For instance, you have the following html on the page that you want 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", na
    }
```

Assume the html was changed recently and you still use the 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", na
    }
```

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: "
SubmitButton(uid: "Submit", clocator: [tag: "input", type: "submit", value: "logon", name: "
```

What will happen without the new group locating algorithm? You 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 trouble to find the individual UI element such as "ProblematicForm.Username.Input", 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.

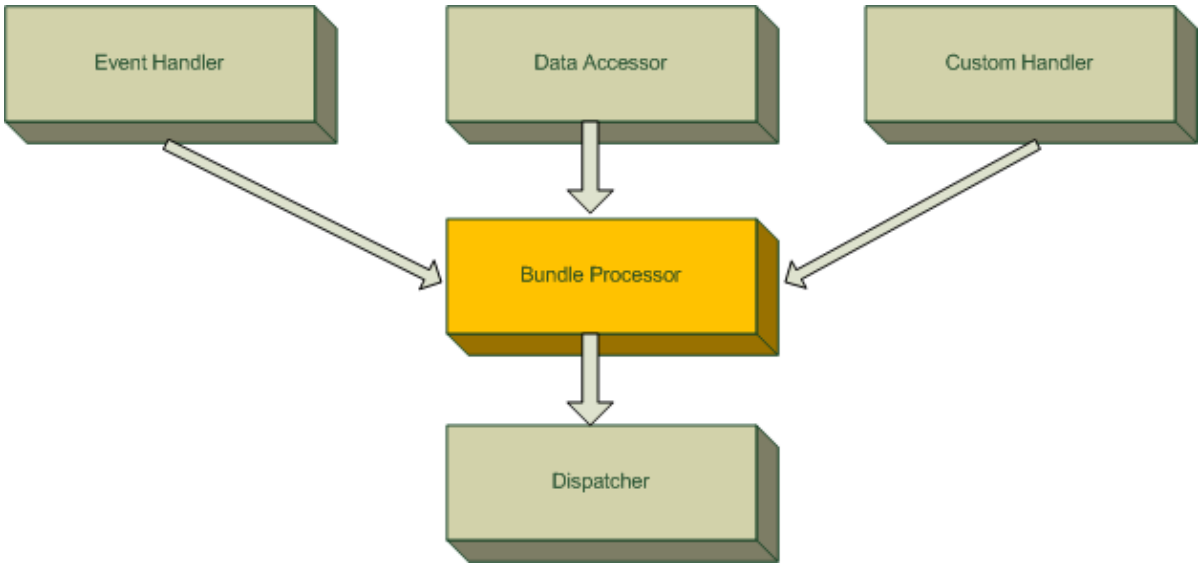
```
UI Module Validation Result for ProblematicForm

-------------------------------------------------------

        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" ty


        --- Element ProblematicForm.Password.Input -->
```

```
        Composite Locator: <input name="j" type="password"/>

        Closest Matched Element: <input size="12" value="" name="j_password" maxlength="25" ty

-------------------------------------------------------
```

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 = false
}
```

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"
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.

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.

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();
```

Be aware that the caching mechanism is still under development.

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()
```

Be aware, the New Tellurium APIs are still under development and are not fully tested. Use them at your discretion.

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);
```

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.

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");
    }
```

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.

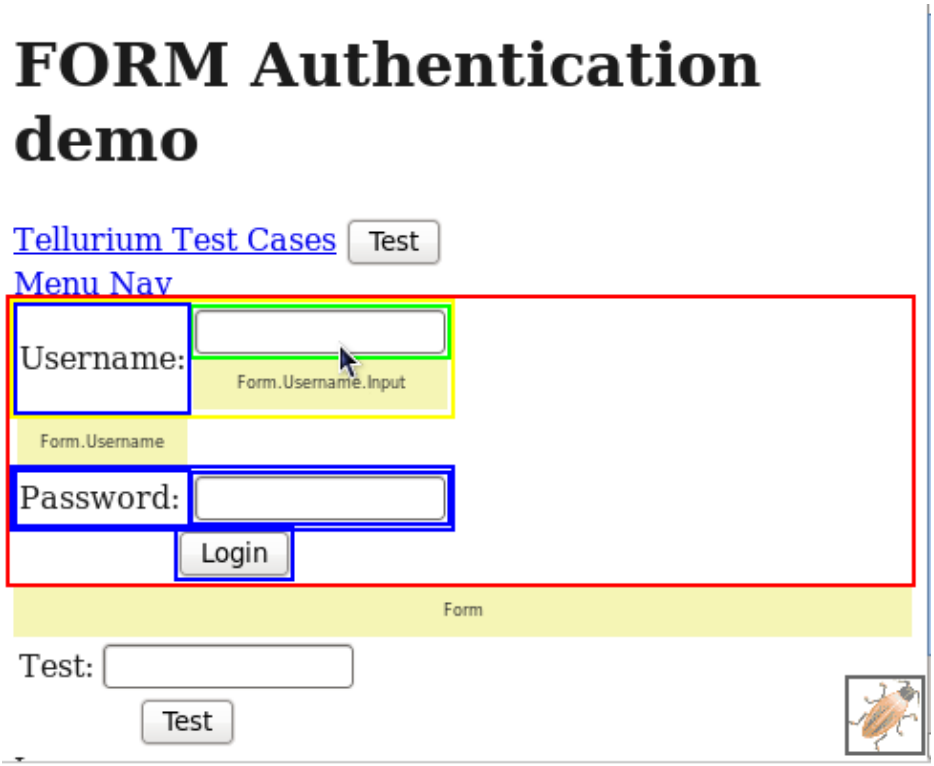
```
    public show(String uid, int delay);
```

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 will be outlined and if user hives over the UI module, the UIDs of the selected UI element's and its ancestors' will be shown in a tooltip as shown in the following figure.

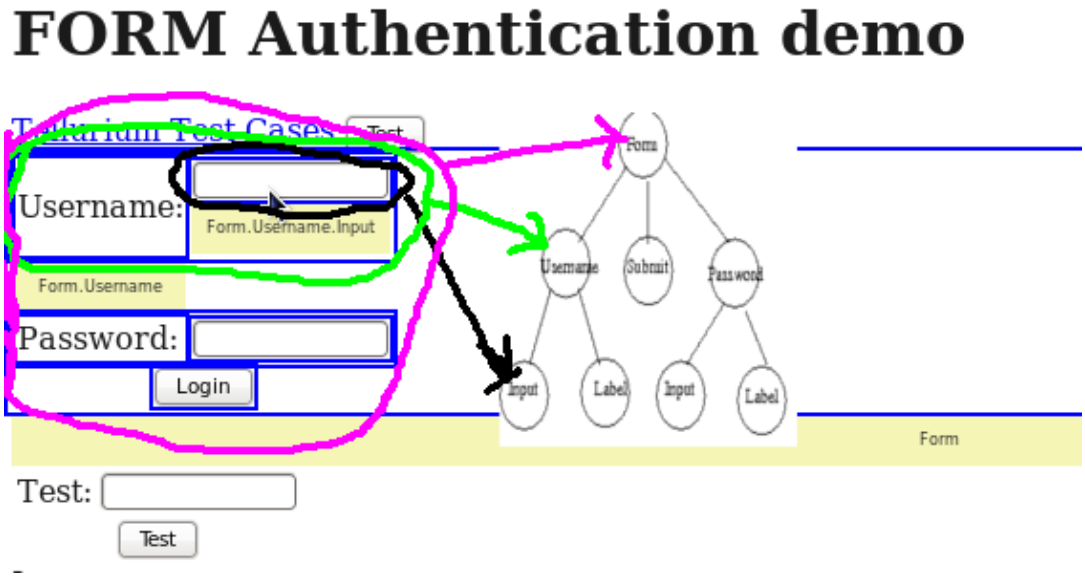


You may be a bit confused by the multiple UID labels. The above UI module is defined as

```
ui.Form(uid: "Form", clocator: [tag: "form"]){
        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", na
    }
```

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 have different colors. How the layout maps to each individual UI element can be illustrated more clearly by the following figure.



In the meanwhile, Tellurium Core also exposes the following two methods for users to start showing UI and clean up UI.

```
public void startShow(String uid);

public void endShow(String uid);
```

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>
        <tr>
            <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
```

I18N

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`

Internationalization support in Tellurium

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 , â ¦ , 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 , â ¦ , 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 extension to user defined tests

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

1. 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

1. Now use the `getMessage` function to internationalize strings

Simple Example

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'Ã©tablissement
GoogleBooksListGroovyTestCase.Category=La catÃ©gorie est {0}
GoogleBooksListGroovyTestCase.ConnectSeleniumServer=Se relier au serveur de sÃ©lÃ©nium
```

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" s
        //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)
      }
}
```

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
            reader = "ExcelFileReader"
        }
    }
```

Standard Table

The Standard Table is used for table with multiple "tbody" sections. The Table layout is as follows.

```
  table
      thead
         tr
           th
           ...
           th
      tbody
         tr
           td
           ...
           td
         ...
      tbody (Could have multiple ones)
         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".
- *fct*: 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".

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");
    }
```

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"> 
                <label>Economy</label>
            </li>
            <li>
                <input type="radio"> 
                <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"> 
                <label>Economy</label>
            </li>
            <li>
                <input type="radio"> 
                <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"> 
                <label>Economy</label>
            </li>
            <li>
                <input type="radio"> 
                <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"> 
                <label>Economy</label>
            </li>
            <li>
                <input type="radio"> 
                <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:

        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");
//        useCssSelector(true);            //uncomment this line if you choose to use css selec
        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.D
        assertEquals("08:00", departureTime);
        String arrivalTime = jlm.getText("SailingForm.Section[1].Option[1].Details.ShipInfo.Arr
        assertEquals("11:45", arrivalTime);
    }
```

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.

Groovy and GMaven

Groovy has been upgraded to the latest version 1.7.0.

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

For GMaven, we use the latest version 1.2 and removed the Maven Antrun plugin.

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.

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

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

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 = false
//    }
```

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$frame
        at org.telluriumsource.bootstrap.TelluriumSupport.addSupport(TelluriumSupport.groovy:17
        at org.telluriumsource.test.java.TelluriumTestNGTestCase.setUpForTest(TelluriumTestNGTe
        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 Configu
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAcces
        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
        at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteAr
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSi
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSi
        at org.telluriumsource.config.TelluriumConfigurator.checkConfig(TelluriumConfigurator.g
```

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="${user.home}/.m2/repository/[organisation]/[module]/[revision]/[module]-[revision](-
    override="false" />
  <resolvers>
    <chain name="downloadGrapes">
      <filesystem name="cachedGrapes">
        <ivy pattern="${user.home}/.groovy/grapes/[organisation]/[module]/ivy-[revision].xml"/>
        <artifact pattern="${user.home}/.groovy/grapes/[organisation]/[module]/[type]s/[artifac
      </filesystem>
      <filesystem name="local-maven-2" m2compatible="true" local="true">
        <ivy pattern="${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/repositorie
      <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
      <ibiblio name="kungfuters.snapshot" root="http://maven.kungfuters.org/content/repositorie
      <ibiblio name="kungfuters.release" root="http://maven.kungfuters.org/content/repositories
    </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
```

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();
```

Misc

Exposed the following Selenium APIs to DslContext.

- `void addScript(String scriptContent, String scriptTagId)`
- `void removeScript(String scriptTagId)`

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 Caching

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-1.4.1.js
|-- jquery-cookies-2.1.0.js
|-- jquery-simpletip-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-uialg.js
|-- tellurium-uibasic.js
|-- tellurium-uiextra.js
|-- tellurium-uimodule.js
|-- tellurium-uiobj.js
```

```
|-- tellurium-uisnapshot.js
|-- tellurium.js
|-- tooltip
|   `-- simpletip.css
|-- user-extensions.js
|-- utils.js
`-- xmlextras.js
```

where

- jquery-1.4.1.js: jQuery is updated to the latest version 1.4.1.
- 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-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 Tellurium UI snapshot
- tellurium-cache.js: Tellurium Engine caching for UI modules and locators
- telurium-extension.js: Extra Tellurium APIs for Selenium
- tellurium-api.js: New Tellurium APIs based on jQuery
- utils.js: Utility functions

Locate Strategies

Moved the CSS selector locate strategy and the cache aware locate strategy from Core to the Engine. This is done by adding the following lines to the method `BrowserBot.prototype._registerAllLocatorFunctions` in the selenium-browserbot.js file:

```
    this.locationStrategies['jquery'] = function(locator, inDocument, inWindow) {
        return tellurium.locateElementByCSSSelector(locator, inDocument, inWindow);
    };

    this.locationStrategies['uimcal'] = function(locator, inDocument, inWindow) {
        return tellurium.locateElementWithCacheAware(locator, inDocument, inWindow);
    };
```

CSS Selector Support

Actually, the CSS selector support was added in Tellurium 0.6.0 and we changed the name "jQuery Selector" to "CSS Selector" as suggested by Dylan.

New APIs

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);
```

UI Module Caching

On the Tellurium Core side, all UI modules are converted into a JSON object. That is why you can see the Tellurium UI objects must implement the following method,

```
abstract JSONObject toJSON();
```

For most UI objects, you only need to implement this method simply as follows if we take the UrlLink object as an example,

```
    public JSONObject toJSON() {

      return buildJSON() {jso ->
        jso.put(UI_TYPE, "UrlLink")
      }
    }
```

For more complicated implementation, please refer to the List object.

Tellurium Core automatically push the UI module definition to the Engine by calling the following method.

```
public void useUiModule(String json);
```

Once the UI module is cached. All locating procedure will be on the DOM sub-tree the UI module represents to reuse the locators and make the locating very fast.

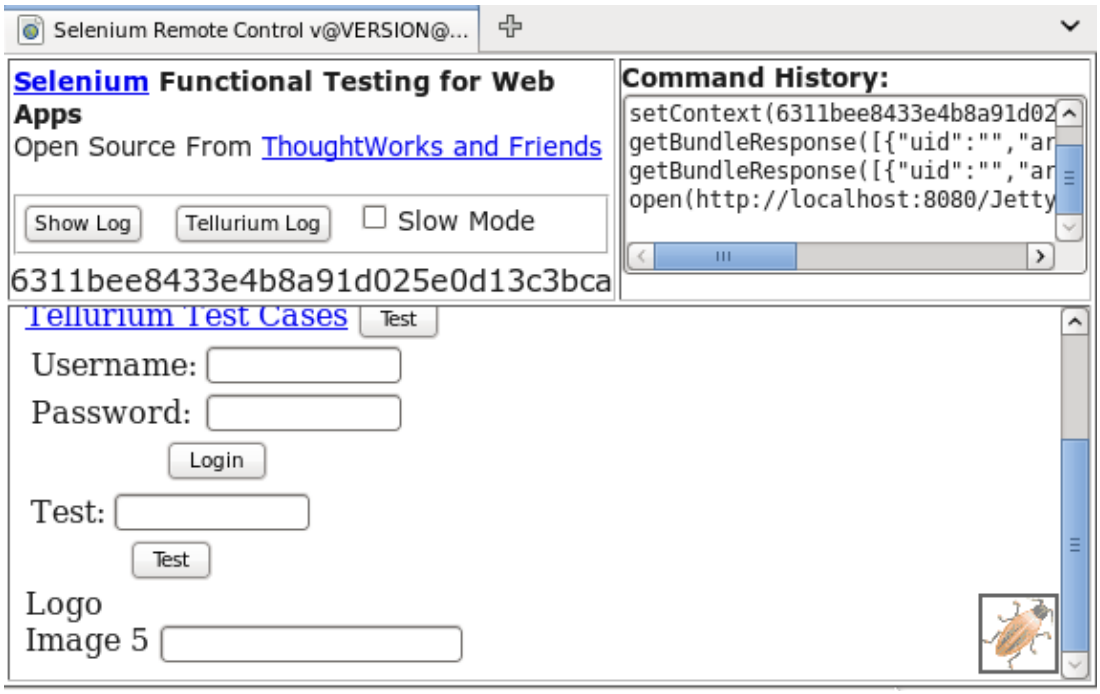
Again, this feature is still under developement. Please don't use it for the timebeing.

jQuery

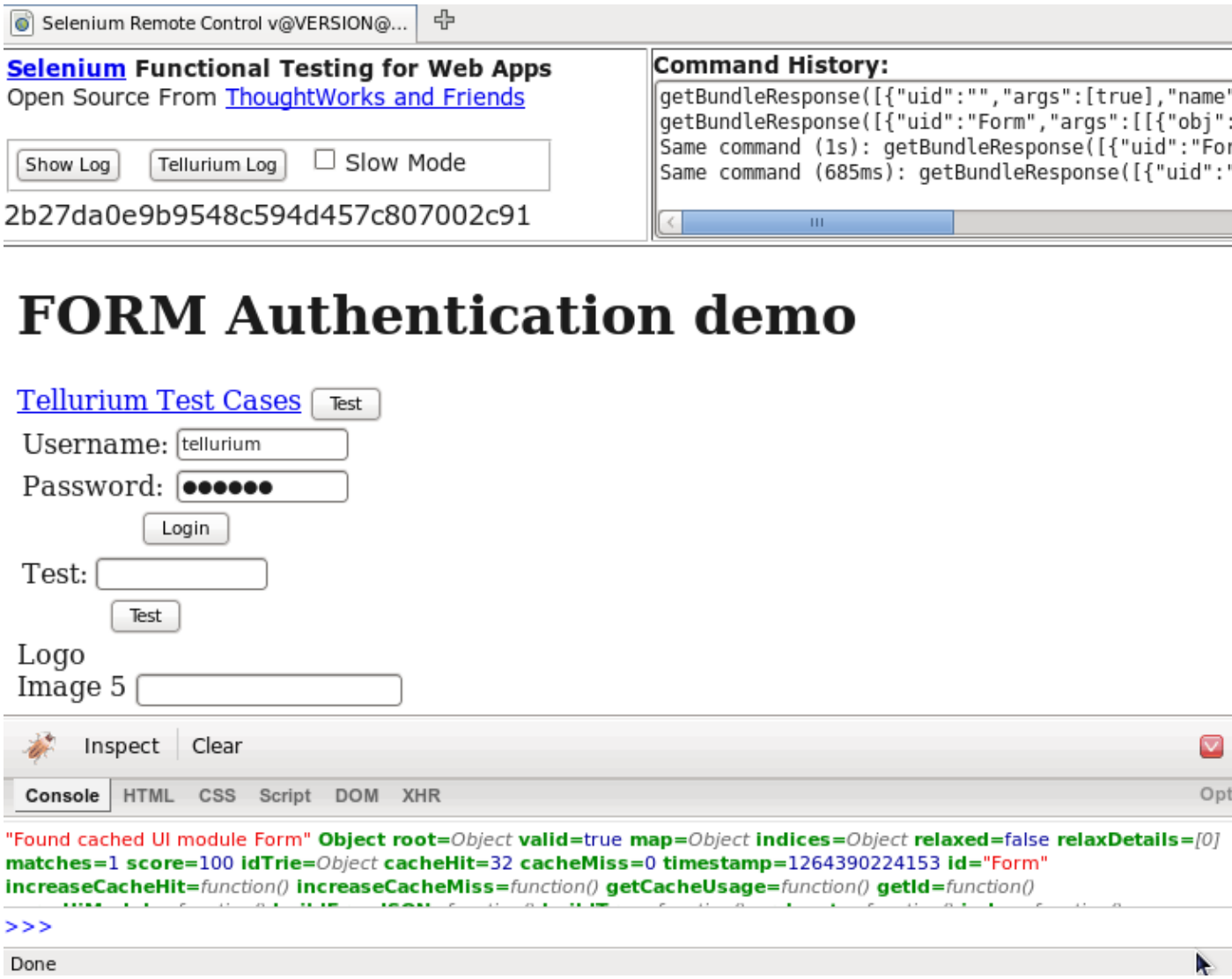
The jQuery in Engine has been upgraded from 1.3.2 to 1.4.1.

Engine Logging

Tellurium Engine uses Firebug Lite 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);
```

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-SNAPSHOT</version>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium.server</groupId>
    <artifactId>selenium-server</artifactId>
    <version>1.0.1-te2</version>
</dependency>
```

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

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

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

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

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

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id -DarchetypeArtifactI
```

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

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id -DarchetypeArtifactI
```

Reference Project

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

Create Custom UI objects

Tellurium supports custom UI objects defined by users. For most UI objects, they must extend the "UiObject" class and then define actions or methods they support. For container type UI objects, i.e., which hold other UI objects, they should extends the Container class. Please see Tellurium Table and List objects for more details.

In the tellurium-website project, we defined the custom UI object "SelectMenu" as follows:

```
class SelectMenu extends UiObject{
    public static final String TAG = "div"

    String header = null

    //map to hold the alias name for the menu item in the format of "alias name" : "menu item"
    Map<String, String> aliasMap

    def click(Closure c){
        c(null)
    }

    def mouseOver(Closure c){
        c(null)
    }

    def mouseOut(Closure c){
        c(null)
    }

    public void addTitle(String header){
        this.header = header
    }
```

```
    public void addMenuItems(Map<String, String> menuItems){
        aliasMap = menuItems
    }

    ......

}
```

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:

```
class SelectMenuBuilder extends UiObjectBuilder{
    static final String ITEMS = "items"
    static final String TITLE = "title"

    public build(Map map, Closure c) {
        def df = [:]
        df.put(TAG, SelectMenu.TAG)
        SelectMenu menu = this.internBuild(new SelectMenu(), map, df)
        Map<String, String> items = map.get(ITEMS)
        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:
        //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: *, column: 1", clocator: [:])

        //the summary + labels column consists of a list of UrlLinks
       List(uid: "row:*, 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 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", trailer: "/tbody/tr/th"], group: "
    //since the actual text is  Project&amp;nbsp;Home, we can use partial match here. Note "*" stan
    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

```
./rundsl.sh src/test/resources/org/telluriumsource/dsl/TelluriumPage.dsl
```

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', clas
Grape.grab(group:'org.stringtree', module:'stringtree-json', version:'2.0.10', classLoader:this
Grape.grab(group:'caja', module:'json_simple', version:'r1', classLoader:this.class.classLoader
Grape.grab(group:'org.seleniumhq.selenium.server', module:'selenium-server', version:'1.0.1-te2
Grape.grab(group:'org.seleniumhq.selenium.client-drivers', module:'selenium-java-client-driver'
Grape.grab(group:'org.apache.poi', module:'poi', version:'3.0.1-FINAL', classLoader:this.class.
Grape.grab(group:'junit', module:'junit', version:'4.7', classLoader:this.class.classLoader.roo

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-te2')
   @Grab(group='org.seleniumhq.selenium.client-drivers', module='selenium-java-client-driver',
   @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: 'rundsl.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)
```

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
```

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 = getCachedVariable("headernum")
            int expectedHeaderNum = getTableHeaderNum()
            compareResult(expectedHeaderNum, headernum)

            List<String> headernames = getCachedVariable("headernames")
            String[] issueTypes = getCachedVariable("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". "getCachedVariable" is used to get variables passed from previous tests and "compareResult" is used to compare the actual result with the expected result.

The 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 bui
    <Message>Issue: Configure IntelliJ to properly load one maven sub-project and not look for
    <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.

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", na
    }
  }

  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="http://books.google.com/books1" style="">Literature</A></P>

        <P><A href="http://books.google.com/books2" style="">Science fiction</A></P>

        <P><A href="http://books.google.com/books3" style="">Fantasy</A></P>

        <P><A href="http://books.google.com/books4" style="">Romance</A></P>

        <P><A href="http://books.google.com/books5" style="">Mystery</A></P>

        <P><A href="http://books.google.com/books6" style="">Fairy tales</A></P>
```

```
        <P><A href="http://books.google.com/books7" style="">Short stories</A></P>

        <P><A href="http://books.google.com/books8" style="">Poetry</A></P></DIV>
    <DIV class="sub_cat_section">
        <DIV class="sub_cat_title">Non-fiction</DIV>
        <P><A href="http://books.google.com/books9">Philosophy</A>
        </P>

        <P><A href="http://books.google.com/books10">Economics</A>
        </P>

        <P><A href="http://books.google.com/books11">Political science</A></P>

        <P><A href="http://books.google.com/books12">Linguistics</A>
        </P>

        <P><A href="http://books.google.com/books13">Mathematics</A>
        </P>

        <P><A href="http://books.google.com/books14">Physics</A>
        </P>

        <P><A href="http://books.google.com/books15">Chemistry</A>
        </P>

        <P><A href="http://books.google.com/books16">Biology</A>
        </P></DIV>
    <DIV class="sub_cat_section">
        <DIV class="sub_cat_title">Random subjects</DIV>
        <P><A href="http://books.google.com/books17">Toys</A>
        </P>

        <P><A href="http://books.google.com/books18">Solar houses</A></P>

        <P><A href="http://books.google.com/books19">Stories in rhyme</A></P>

        <P><A href="http://books.google.com/books20">Courtship</A>
        </P>

        <P><A href="http://books.google.com/books21">Mythology</A>
        </P>

        <P><A href="http://books.google.com/books22">Differential equations</A></P>

        <P><A href="http://books.google.com/books23">Latin inscriptions</A></P>

        <P><A href="http://books.google.com/books24">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") {
          TextBox(uid: "title", clocator: [tag: "div", class: "sub_cat_title"])
          List(uid: "links", separator: "p") {
            UrlLink(uid: "all", clocator: [:])
          }
        }
      }
    }
  }
}
```

Table

HTML Source:

```
<table id="xyz">
    <tbody>
    <tr>
        <th>one</th>
        <th>two</th>
        <th>three</th>
    </tr>
    </tbody>

    <tbody>
    <tr>
        <td>
            <div class="abc">Profile</div>
        </td>
        <td><input class="123"/><br/>

            <div class="someclass">
                <span class="x">Framework</span><br/>
                <a href="http://code.google.com/p/aost">Tellurium</a>
            </div>
        </td>
        <td>
            Hello World!
        </td>
    </tr>

    </tbody>
</table>
```

UI Module:

```
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
  }
}
```

Where to Find ?

0.7.0 SNAPSHOTs

Where to find the latest 0.7.0 snapshots?

They are in our Maven repo.

- Tellurium Core 0.7.0 Snapshot
- Custom Selenium Server with Tellurium Engine

If you use Maven, you need the following dependencies

```
        <dependency>
            <groupId>org.telluriumsource</groupId>
            <artifactId>tellurium-core</artifactId>
            <version>0.7.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.seleniumhq.selenium.server</groupId>
            <artifactId>selenium-server</artifactId>
            <version>1.0.1-te2-SNAPSHOT</version>
        </dependency>
```

Here is a sample POM file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.telluriumsource</groupId>
    <artifactId>tellurium-website</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>Tellurium Reference Project - Tellurium Website</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>

    <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>${tellurium-version}</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium.server</groupId>
    <artifactId>selenium-server</artifactId>
    <version>${selenium-version}-te2-SNAPSHOT</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>r1</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>
```

```
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>5.8</version>
            <classifier>jdk15</classifier>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.uncommons</groupId>
            <artifactId>reportng</artifactId>
            <version>0.9.8</version>
        </dependency>
        <dependency>
            <groupId>velocity</groupId>
            <artifactId>velocity-dep</artifactId>
            <version>1.4</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>
    <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>
                        <!-- The generateStubs goals are not yet working for enums and gene
                        <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>
        <tellurium-version>0.7.0-SNAPSHOT</tellurium-version>
        <javac-debug>true</javac-debug>
    </properties>

</project>
```

0.7.0 RC1

For 0.7.0-RC1, you can find Tellurium core and custom selenium server from the following URLs, respectively.

http://maven.kungfuters.org/content/repositories/releases/org/telluriumsource/tellurium-core/0.7.0-RC1/

http://maven.kungfuters.org/content/repositories/thirdparty/org/seleniumhq/selenium/server/selenium-server/1.0.1-te2-

If you Tellurium Maven archetypes, you can use the following command to create a Tellurium 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-RC1 \
    -DarchetypeRepository=http://maven.kungfuters.org/content/repositories/releases
```

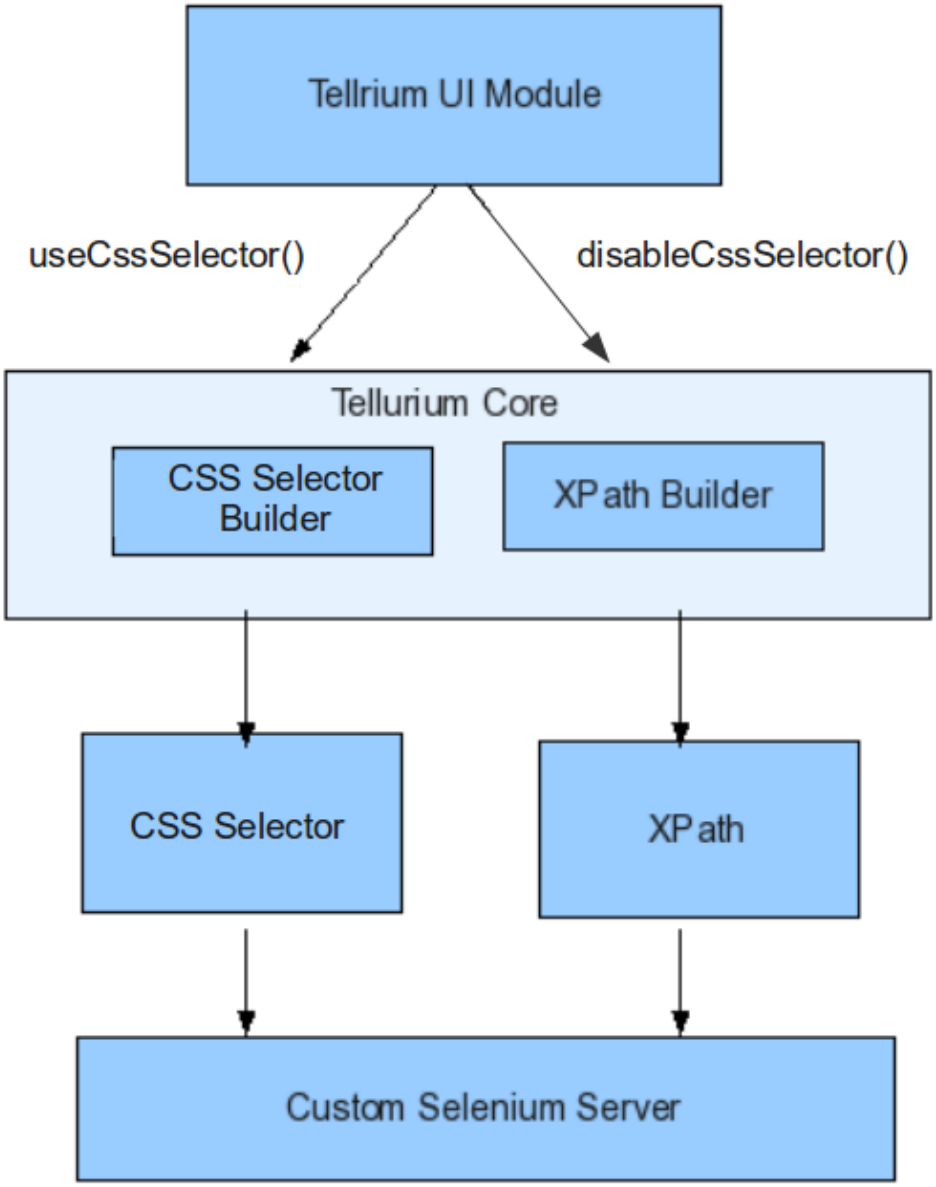
Similarly, use the following Maven command for a Tellurium TestNG project,

```
mvn archetype:create -DgroupId=your_group_id -DartifactId=your_artifact_id \
    -DarchetypeArtifactId=tellurium-testng-archetype \
    -DarchetypeGroupId=org.telluriumsource -DarchetypeVersion=0.7.0-RC1 \
    -DarchetypeRepository=http://maven.kungfuters.org/content/repositories/releases
```

Advanced Topics

The Santa Algorithm

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. This procedure can be illustrated by the following diagram.



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);
```

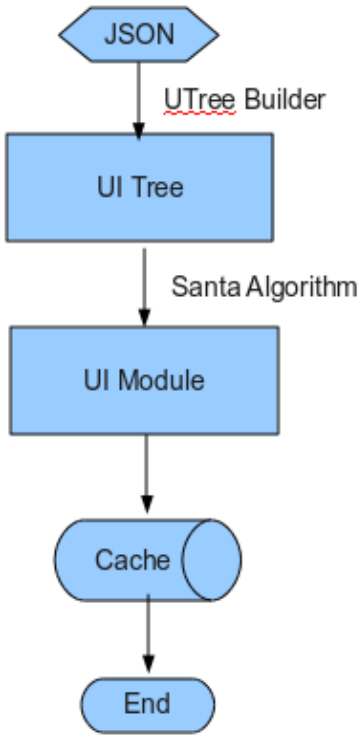
Why is the algorithm named **Santa**. This is because I have completed most of the design and coding work during the Christmas season in 2009. It is like a gift for me from Santa Claus.

Problem

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.

System Diagram

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":"Text
   {"obj":{"uid":"Input","locator":{"tag":"input","attributes":{"name":"j_username","type":"tex
   {"obj":{"uid":"Password","locator":{"tag":"tr"},"uiType":"Container"},"key":"Form.Password"}
   {"obj":{"uid":"Label","locator":{"direct":true,"text":"Password:","tag":"td"},"uiType":"Text
   {"obj":{"uid":"Input","locator":{"tag":"input","attributes":{"name":"j_password","type":"pas
   {"obj":{"uid":"Submit","locator":{"tag":"input","attributes":{"name":"submit","value":"Login
```

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 module at Tellurium Engine is defined as follows.

```
function UiModule(){
    //top level UI object
    this.root = null;

    //whether the UI module is valid
    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 = new Hashtable();

    //If the UI Module is relaxed, i.e., use closest match
    this.relaxed = false;

    //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 look Id operation in group locating
    this.idTrie = new Trie();

    //Snapshot Tree, i.e., STree
    this.stree = null;
}
```

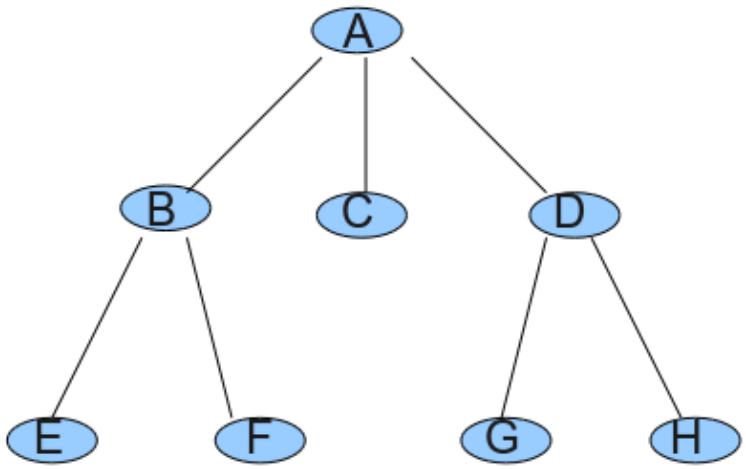
From above, you can see the UI module has two indices for fast access. One is UID to UI object mapping and the other one is the UID to DOM reference mapping.

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

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.

Main Flow

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
```

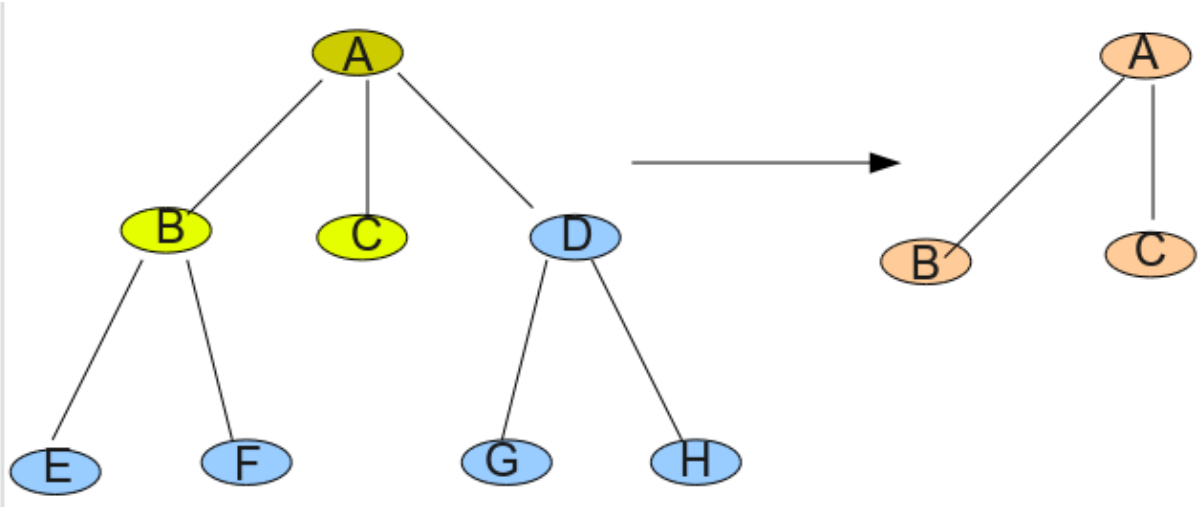
```
        this.squeue.push(ust);
    }
    while(this.oqueue.size() > 0){
        var uiobj = this.oqueue.pop();
        uiobj.locate(this);
    }

    //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.

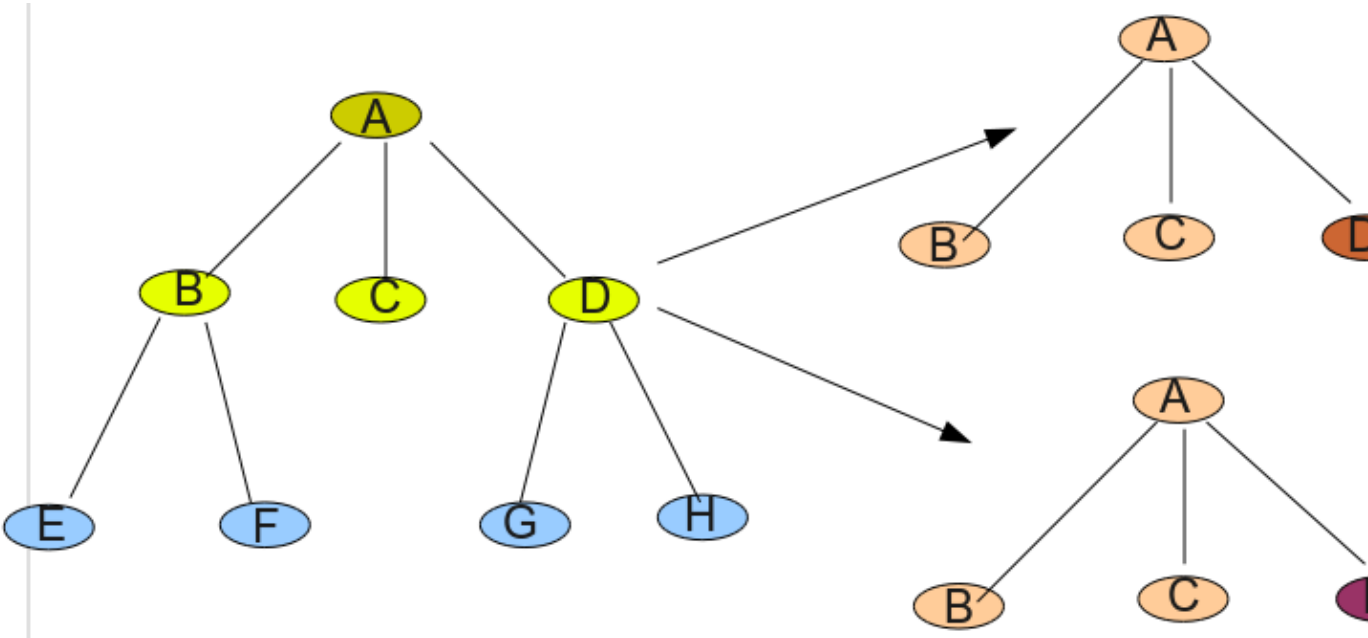
```
UiAlg.prototype.locate = function(uiobj, snapshot){
    //get full UID
    var uid = uiobj.fullUid();
    var clocator = uiobj.locator;

    //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

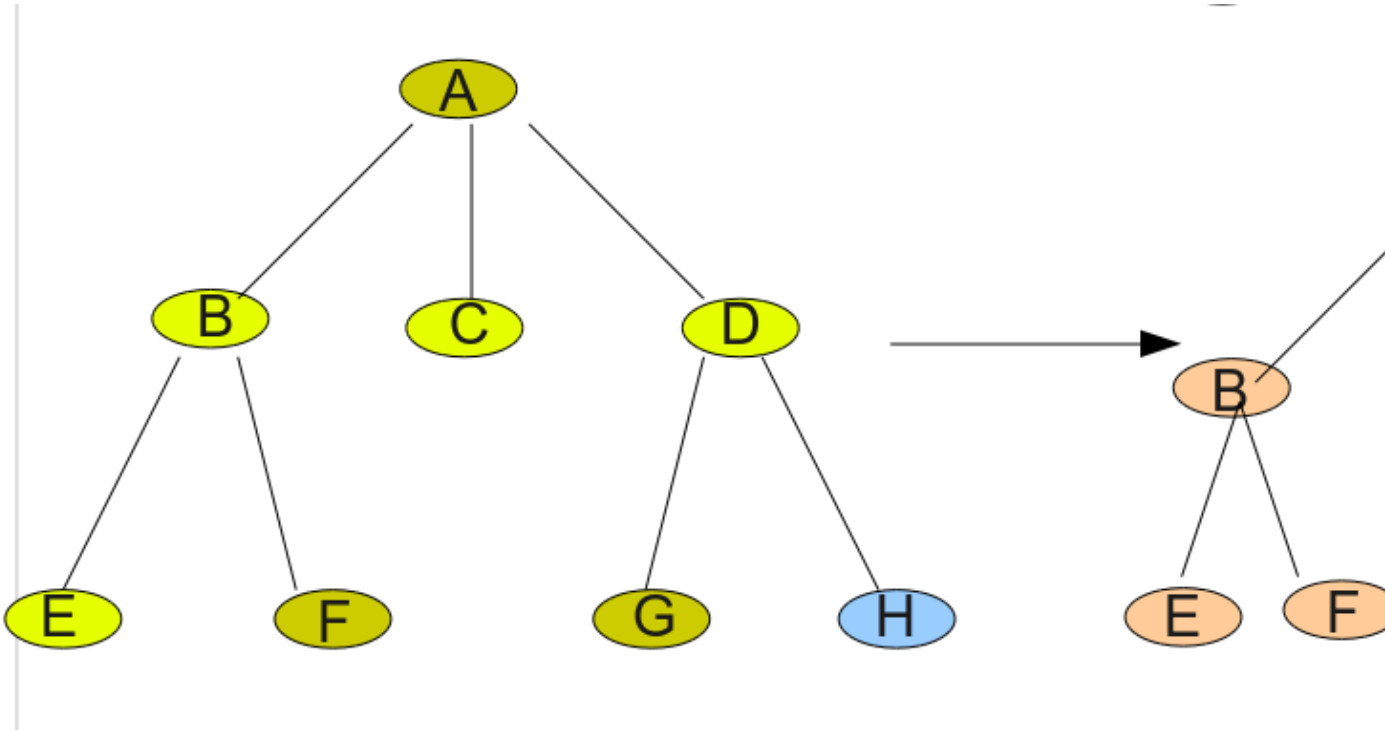
Santa is basically a branch and trim search procedure on the runtime DOM. Assume at some point, the Santa algorithm has located UI elements A, B, and C. A snapshot has been generated as shown in the following diagram.



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.

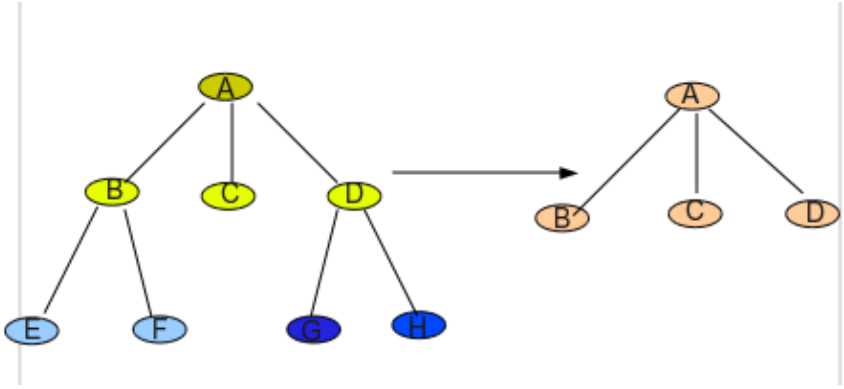
```
$found.eq(0).data("uid", uid);
```

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.

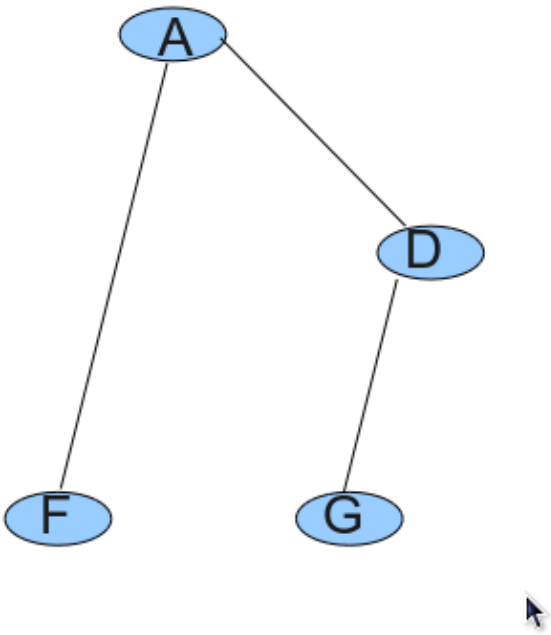
Look Ahead

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 obj
        //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.

```
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 jqs = 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".

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 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.

The 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 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': '#3030
                           'background-color': '#f5f5b5', 'border': '1', 'solid': '#DECA7E',
                           'font-family': 'sans-serif', 'font-size': '8', 'line-height': '12px
                         })
                    .html(conf.content)
                    .insertAfter(elem);
```

In additional 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

Example

To show the UI module visual effect, we still consider the the "FORM authentication demo" example. The test case is shown as follows.

```
public class FormExampleTestCase extends TelluriumMockTestNGTestCase {
    private static FormExampleModule fem;

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

```
        fem = new FormExampleModule();
        fem.defineUi();
        useTelluriumEngine(true);
        useTrace(true);
        useEngineLog(true);
    }

    @Test
    public void testShowUi(){
//        fem.show("Form", 10000);
        fem.startShow("Form");
        fem.endShow("Form");
    }

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

Visual Effects

First, you can download the above example by checking out the ui-examples reference project from subversion or download it from

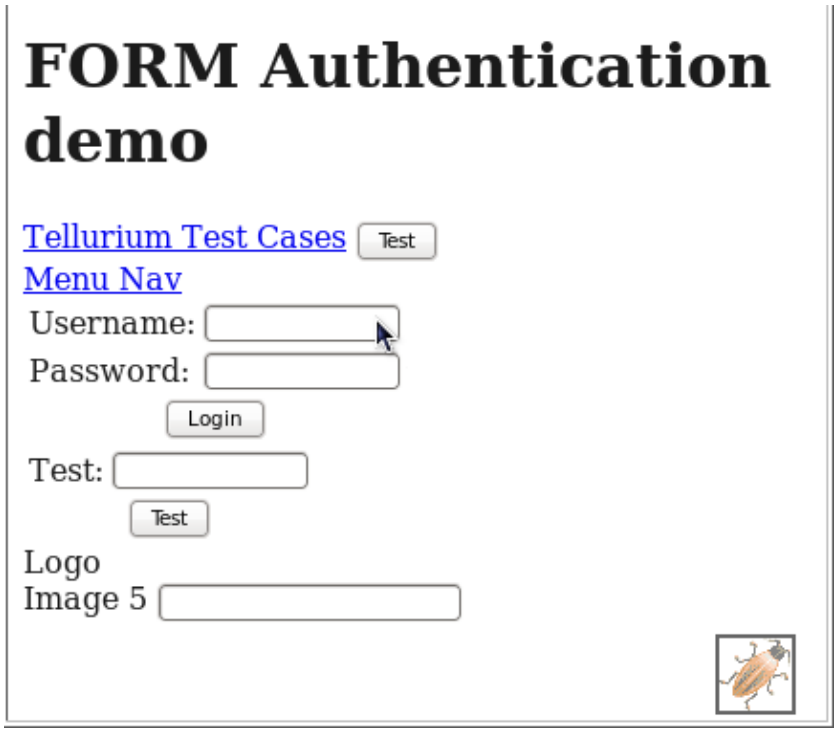
http://aost.googlecode.com/files/tellurium-examples-0.7.0-SNAPSHOT.tar.gz

Then, load the project into an IDE and put a breakpoint at the line

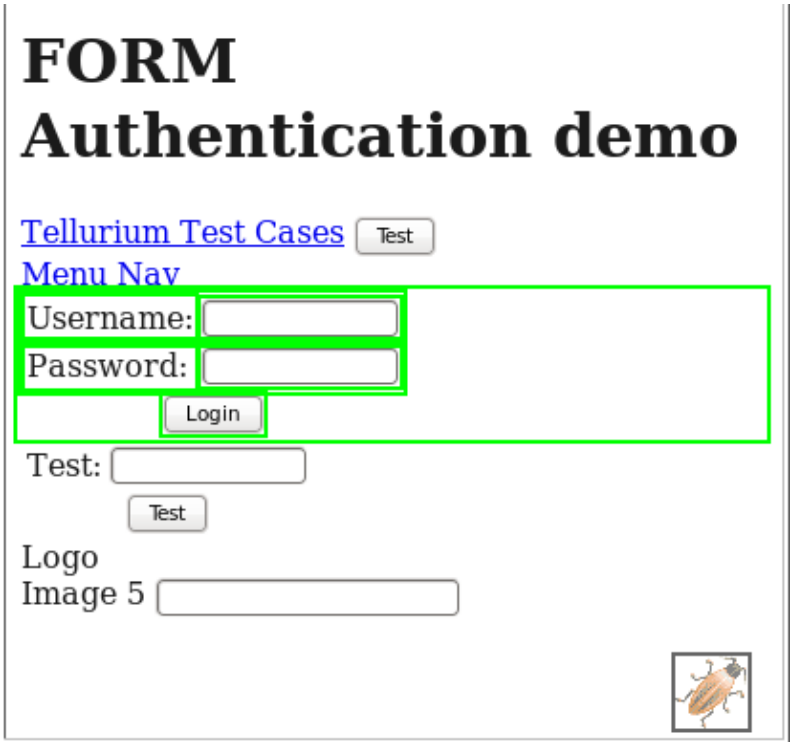
```
fem.startShow("Form");
```

After that, debug the test case `testShowUi()`.

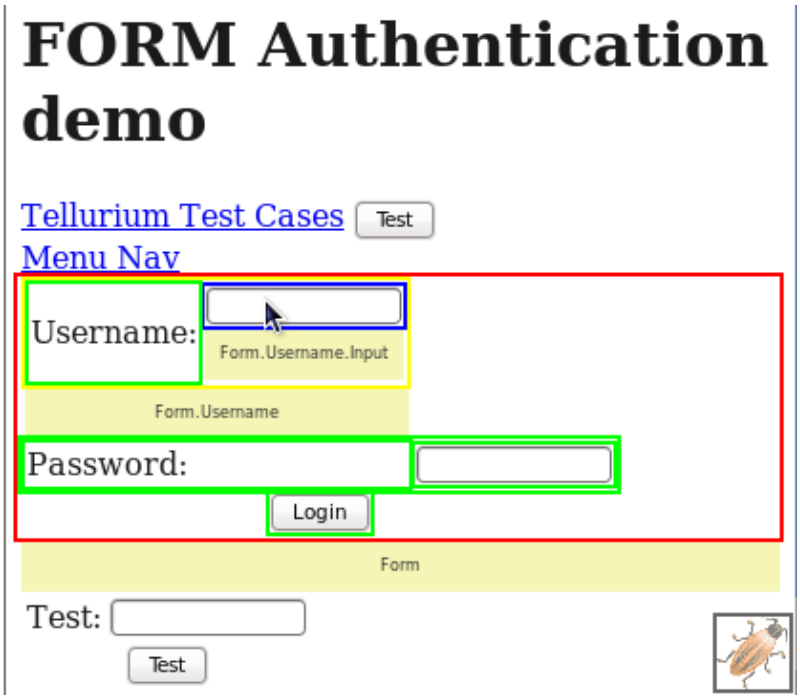
Before the `startShow` command, the UI looks as follows,



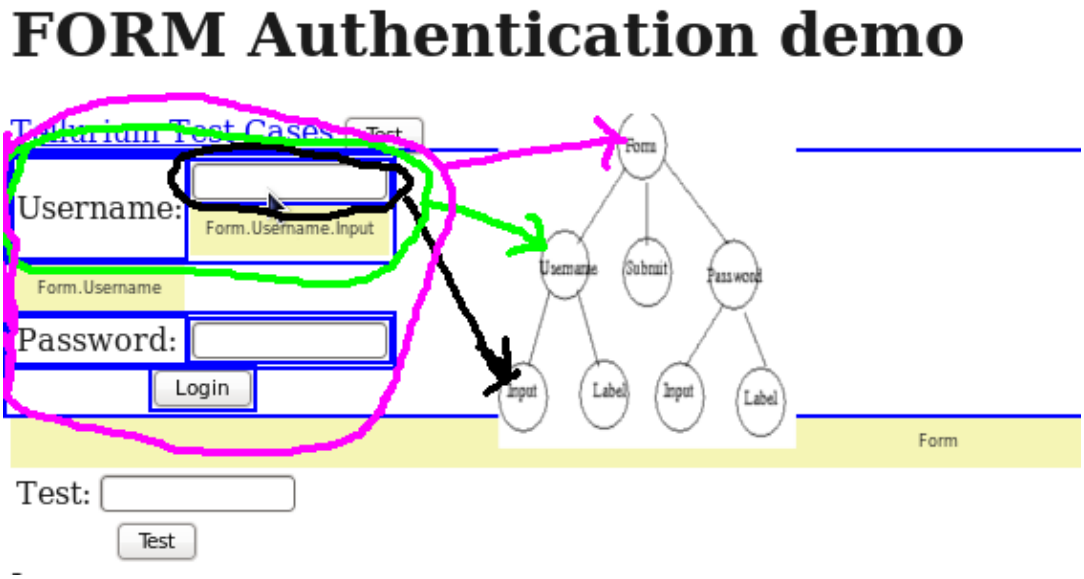
and after the `startShow` command, the UI module is outlined by the outline visitor.



If you mouse over the UI module, you will see the visual effects as follows.



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.



Once you call the following command

```
endShow("Form");
```

The visual effects are removed and the UI is restored to the original one.

How to Debug Tellurium Engine

For the past three months, I have worked on the new Tellurium Engine project heavily. A lot of new ideas, for example, the Santa algorithm and UI module visual effect, become a reality. Debugging Tellurium Engine is a critical task for me everyday. I cannot image I can finish all the tasks without a good debug technology.

Tellurium Engine is similar to Selenium core to drive browser events to simulate users actions on the web application. If you are familiar with Selenium core, you may find that it is very difficult to debug Selenium core. As a result, I took a different approach to do debugging and like to share my experience with you here.

Basic Idea

The basic idea is to write Javascript test cases to call Tellurium Engine and embed the tests in the header of the html page that I want to test. Then I can set a breakpoint in the Javascript code using Firebug and refresh the web page to run the tests. Some part of Tellurium Engine need to call Selenium core, I can mock that up.

Prerequisites

To use Firebug, I need to have Firebug installed to either the Firebug profile or the customized Selenium server. How to add the Firebug support is described in details in the wiki page Use Firebug and JQuery to Trace Problems in Tellurium Tests.

Details

Jetty

First, I installed a Jetty server and use the Jetty server to load up the web page I want to run tellurium Engine test on.

Example

Take the "FORM authentication" as an example.

Create Test Page

First, I added the following headers to it.

```
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"> </scr
          <script src="http://localhost:4444/selenium-server/core/scripts/json2.js"> </script>
          <script src="http://localhost:4444/selenium-server/core/scripts/utils.js"> </script>
          <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.
        <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.
        <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"> </scri
        <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 actual Javascript test file to test the Tellurium Engine. The above header loads up the Tellurium Engine code when the page is served by Jetty.

Engine Test Script

I 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":"Text
   {"obj":{"uid":"Input","locator":{"tag":"input","attributes":{"name":"j_username","type":"tex
   {"obj":{"uid":"Password","locator":{"tag":"tr"},"uiType":"Container"},"key":"Form.Password"}
   {"obj":{"uid":"Label","locator":{"direct":true,"text":"Password:","tag":"td"},"uiType":"Text
   {"obj":{"uid":"Input","locator":{"tag":"input","attributes":{"name":"j_password","type":"pas
   {"obj":{"uid":"Submit","locator":{"tag":"input","attributes":{"name":"submit","value":"Login
    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 uiid = new Uiid();
    var uinput = uim.walkTo(context, uiid.convertToUiid("Form.Username.Input"));
    var pinput = uim.walkTo(context, uiid.convertToUiid("Form.Password.Input"));
    var smt = uim.walkTo(context, uiid.convertToUiid("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 UI element DOM reference. After than, I test the *getHTMLSource* and *getUiByTag* Tellurium APIs.

As I said, Tellurium Engine needs to call Selenium Core code somewhere. I 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') { // IE, not ECMA
        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();
```

Logging

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 script in the custom selenium server so that you can still use the above logging wrapper.

Run the Test

To run the above test page, first, I 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`. I 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 I can start to debug the application.

What's Next ?

- More flexible UI templates, i.e., UID Description Language (UDL)
- Tellurium Widget revisit
- Trump IDE upgrade to match 0.7.0

Want to Contribute ?

We welcome contributions for Tellurium from various aspects. Right now, we are looking for new members to join our dev team. More details on How to Contribute.

Resources

- Tellurium Project website
- Tellurium on Twitter
- Tellurium User Group
- Tellurium Automated Testing Framework LinkedIn Group
- Tellurium User Guide
- Tellurium at Rich Web Experience 2009 by Jian Fang and Vivek Mongolu
- Tellurium 0.6.0 User Guide
- Tellurium video tutorial by Vivek Mongolu
- Tellurium - A New Approach For Web Testing
- 10 Minutes to Tellurium
- How to create your own Tellurium testing project with IntelliJ 9.0 Community Edition