DynaBytes, Inc.
Java-Based Data Driven Test Automation Project

Contents

DynaBytes, Inc.

Java-Based Data Driven Test Automation Project

Introduction

This document describes the functional aspects of the test automation project at DynaBytes Inc.

This project follows a test automation philosophy coined as 'Data Driven Testing' that separates the details of test automation from the test automation script / code.

This approach is motivated by:

1. *Cost of automation tools in general.* The jDDT project is built using open source code and plugins and bears no licensing fees.
2. *The increase in the complexity of applications and their user interface (UI results in a commensurate increase in the coding of aumation tools.* The jDDT approach avoids coding but does not exclude it. This opens up the use of jDDT to QA staff without coding experience.
3. *Familiar Territory* – The jDDT tool is based on Selenium, a widely used UI automation framework that facilitates creation of tools on top of it along with open source plugins developed for it.
4. *Test Automation Features* – The jDDT tool provides several compelling features designed and implemented by test automation expert(s).
5. *Maintenance.* Traditional QA Automation uses at least one coded element per test. Considering most current applications may yield billions of test cases, maintaining the coded test harness is a daunting task all by itself. The jDDT approach uses a single code base to run any number of tests. The maintenance is shifted to excel spreadsheets (or other input source) from a code-base (that has to be retested itself whenever an element in it has changed.)

The Data Driven Test Automation project described here (jDDT) is an attempt to return the creation and maintenance of testing to the domain of the QA staff.

This is achieved thru abstraction of the interaction of a user with an application and the UI of the application by separating the details of the testing from the testing code. These enable expansion of the test harness with little or no expansion of the test software.

Under this paradigm, adding test cases to the test harness is done by adding worksheets in spreadsheet(s) and rows in worksheets, html, JSON or XML files instead of coding more modules.

Non-Silverlight web site(s) is the initial 'sandbox' for jDDT.

Java, the scripting environment used by jDDT, Selenium (from thought works®) is the framework upon which jDDT is based, Maven is the dependency management tool used by jDDT and the (free) JetBrains Intellij IDE is used by the developer(s). The project can be easily converted to Eclipse format. Any open source code management platform (github, etc.) can be used as version control.

Two types of data source are currently supported, namely, File and Inline.

File formats currently supported are:

- **Excel spreadsheet**. Each row in a given spreadsheet is equivalent to a test step in the test automation paradigm. Each spreadsheet can represents a test scenario but the test designer is not forced to use this paradigm – the test harness can be constructed in many ways. JDDT 'knows' how to handle .xls and .xlsx spreadsheets.
- **Html File**. Proprietary structure. Each <tr></tr> element represents a test step. Each <td/> element represents a test item property (all should be included.)

Inline:

- **JSON File**. Proprietary structure containing a list of JSON objects representing TestItem instances.
- **Tab Delimited Text File**. Each line in a given tab delimited file is equivalent to a test step in the test automation paradigm. The first row is assumed to be field header line and is ignored. Lines starting with # (hash mark) are considered comments and are ignored (as are empty lines.)
- **XML File**. Proprietary structure. Each <TestIem></TestItem> element represents a test step. Each <TestCase></TestCase> element represents a scenario.
- **Test Strings Generator class**. For users who prefer coding on external file as test item provider, a special type of class is used to provide / generate test item strings (from which test item objects are constructed.) The input source for this type starts with Inline followed by class name and method name. Example: Inline!`DemoTestStringsGenerator!`root.

  Example: The following class (`DemoTestStringsGenerator`) has one method named root with 5 steps each of which refer to another test case.

```
public class DemoTestStringsGenerator extends InlineTestStringsProvider {
   public DemoTestStringsGenerator() {
   }
   // ================================================================================================================
   //                               Test Item Strings Generating Methods
   // ================================================================================================================
public void root(ArrayList<String[]> list) {
  list.add(new String[] {"DDTDemo01", "newTest", "", "", "", "", "InputSpecs=Inline!DemoTestStringsGenerator!calculator", "Run the Calculator tests"});
  list.add(new String[] {"DDTDemo02", "newTest", "", "", "", "", "InputSpecs=Inline!DemoTestStringsGenerator!chainingFinders", "Run the ChainFinders tests"});
  list.add(new String[] {"DDTDemo03", "newTest", "", "", "", "", "InputSpecs=Inline!DemoTestStringsGenerator!cssFinders", "Run the CssFinders tests"});
  list.add(new String[] {"DDTDemo04", "newTest", "", "", "", "", "InputSpecs=Inline!DemoTestStringsGenerator!frameSwitching", "Run the Frame Switching tests"});
  list.add(new String[] {"DDTDemo05", "newTest", "", "", "", "", "InputSpecs=Inline!DemoTestStringsGenerator!nonBrowserPassingTests", "Run constant verifications
"});
  list.add(new String[] {"DDTDemo06", "newTest", "", "", "", "", "InputSpecs=Inline!DemoTestStringsGenerator!nonBrowserFailingTests", "Run constant verification -
failures"});
  list.add(new String[] {"DDTDemo07", "generateReport", "", "", "", "", "Description=Report For Inline Test Provider", "Generate the report for the demo inline
items provider"});
  stringifyTestItems(list);
 }
}
```

A test step can be an atomic activation or verification of the application or execution of a sub-script (much like a programmer's subroutine.)

Abstraction of Interaction with Software Application – The jDDT Approach

A user's interaction with an application (for now, leaving out touch screen platforms) is initiated by either a keyboard or a pointing device (mouse.) The application under test (AUT) responds to the signals coming from either to produce specific behavior that is reflected in the UI and, thus, can be subject to activation & verification.

UI Testing can, thus, be thought of as a series of Activation and Verification steps. Both, Activation and Verification steps typically apply to some UI components.

Thus, the UI aspect of test automation revolves around locating elements, activating elements, interrogating elements and comparing the result of such interrogation with expected values.

Non-UI, aspects of test automation must also be addressed by any automation paradigm to enhance the value of such tools (for example, is there a file named 'xyz.doc' in some folder, etc.)

The jDDT driver reads text input from the source (excel spreadsheet, xml file, html or JSON file), parses it to its test harness semantics and carries the test semantics step after step.

The semantics of test steps are encoded as described below. Each of the attributes in the abstraction is a property of an instance of a test step represented by an instance of the TestItem class in the jDDT implementation. In the case of XML input, a property in a <TestItem></TestItem> element. In the case of HTML input, <tr/> elements represent test steps with properties in <td/> elements.

Please note:

- Some of the attributes used below have no interaction or verification semantics but are used for reporting, documentation or other aspects that facilitate interpretation of test results.
- Some of these properties are generic and some of these are specific to the jDDT parent framework (Selenium) and may not make sense in the context of other test automation tools.
- Not all properties are used by each of the steps.
- The term 'Element (or element) describes an Element on a web page (body, div, tag, input, option, etc.)
- As of this writing, the ordrer of columns in the excel spreadsheet matters and one must use the order prescribed below. Neither the XML, nor the HTML input bear this restriction.

ID

Used to identify a given test step. Test ID is used for documentation purposes only and may correspond to some step in a formal, manual, test plan.

The 'Id' attribute of a given test step is mainly used to identify it in the result set in order to facilitate interpretation of the test result (what happened where / when.) For example: If the Id column of three steps in the login sequence is `"Login01", "Login02", "Login03"` respectively, the results report will indicate these 'as is'. If you want to have the test driver automatically sequence the steps within a test case, denote those with the following (corresponding) notation: `"Login#", "Login#", "Login#"`. This will result in reporting those steps as: `"Login.001", "Login.002", "Login.003"` and

you will not have to worry about sequencing those steps correctly in the data stream upon inserting data rows, repositioning data rows, deleting datarows, etc.

Action

Action specifies / encodes the action to be taken by the jDDT test driver. This is the only mandatory attribute for all test items.

The jDDT driver maps the Action component of the input to a Java object / method and thus, must be spelled accurately (but, optionally, case 'insensitive'.)

Conceptually speaking, each action within jDDT is equivalent to a verb in jDDT's dictionary. The collection of jDDT Actions defines the vocabulary / behavior repertoire of the tool.

Some actions are UI related (Click, Select, Type) and some are not (Wait, runCommand, generateReport, newTest).

LocType

The LocType attribute designates the type of locator used in order to locate an Element on a Web Page.

This is blank for (or ignored by) non-UI test steps.

It is advised to choose the most descriptive locator type available (performance is a secondary consideration.)

Valid options are:

Id	This is the preferred type to use when element has a unique id
Name	This is the preferred type to use when element has no unique id but has a unique name.
Css	This is used in lieu of id or name when the css selector is the most descriptive of all others.
ClassName	Used in lieu of id and name in a situation where the Class Name of a web element will yield consistent results.
LinkText	Used to locate a link element by its text.
PartialLinkText	Used to locate a link element using part of its text.
TagName	Used to locate an element by its tag name
Xpath	Used in cases where other locators are neither unique nor provide an improved location. EVERY element has an xPath locator but xPath locators may behave inconsistently in pages where the contents of the page may change dynamically.

The content of LocType is NOT case sensitive.

When multiple ("chained") locators are used for a UI test item, the various locators are separated by "`" character (without the double quotes, and with no spaces.) The deleimited items correspond to those listed in the LocSpecs property (example: id`xpath`css.) Multiple LocType usage must correspond to commensurate multiple LocSpecs usage.

LocSpecs

LocSpecs is used in conjunction with LocType to locate an Element on a Web Page.

Used for UI steps (left blank/ignored by test steps that are not related to UI components.)

The LocSpecs property contains the value associated with the locator type specified in LocType.

When multiple ("chained") locators are used for a UI test item, the various locator specs are separated by "`" character (without the double quotes, and with no spaces.) The deleimited items correspond to those listed in the LocType property.

Suppose an Element on a page has a unique Id of 'save-button'

LocType will be 'id' (without the quotes) and LocSpecs will be 'save-button' (without the quotes).

QryFunction

The QryFunction property is used for verification of elements. It specifies a way to get some value from a located element (or a web page.) In some cases, this property is used in conjunction with the Data token QueryParam (see below.)

This is left blank (or ignored by) test steps that are not related to UI components.

GetAttribute — Gets the value of an element's attribute whose name is provided in the Data token **QueryParam**

Example: Suppose a web element like the following is on some page:

```
<img title="Title" alt="myTitle" src="A/B/C/xyz.png">
```

For a test item where

QryFunction: 'GetAttribute'

Data contains: QueryParam=alt

The value is 'myTitle' (without the quotes.)

Example: Suppose you want to verify the number of rows in a table where the table can be located by:

LocType: id

LocSpecs: myTable and the element located is the <tbody> element of a table.

Use the following:

QryFunction: GetAttribute

Data: QueryParam=childElementCount; SaveAs=TableSize

After the step successfully completed, the number of rows in that table is assigned to a variable named TableSize and can be referenced as {TableSize} in a test step for verification purposes.

	An interrogation plugin like Firefox' FireBug facilitates the determination of specifying these values.
GetClass	Gets the element's class name.
GetCssValue	Gets the Css Value of an element's CSS property whose name is provided in a Data token name QueryParam. This works the same way as the GetAttribute above with respect to the parameter to the property to be used with getCssValue() Suppose the queryParam evaluages to "expanded", the result of the query will be the result of `element.getCssValue("expanded")`
GetSize	Gets the size property of an element.
GetTagName	Gets an element's tag name.
GetText	Gets an element Text property (this is the most frequently used option.)
GetTitle	Gets the title of a web page or a Title element on a page.
IsDisplayed	Returns 'true', 'false' or blank for the isDisplayed property of an element.
IsEnabled	Returns 'true', 'false' or blank for the isEnabled property of an element.
IsSelected	Returns 'true', 'false' or blank for the isSelected property of an element.

Active

The Active attribute is used to activate and de-activate steps. Any value other than "no" is interpreted by the jDDT driver as an active test step.

Deactivating steps may be a great time-saver during construction of a test harness when one needs to test only a portion of the harness.

Data

The Data attribute is used for passing parameters / context to the various Vocabulary instances and is verb-specific. The parameters / context affects the behavior of the DDTTestRunner instance that execute test steps, the Verifier class that verifies the page or its element(s) or the Query class that interrogates web page(s) or web elements on web pages.

The various usages of this attribute are provided in the detailed description of the vocabulary and in the section below.

The Data attribute is formatted as a doubly-delimited string specifying various pieces of information relevant to the Action specified for a test item.

For example, the sendKeys verbs expects a variable "tokenized" as "Value" (this is the text to enter in a text box.) So, to enter "some text" in a text box the Data attribute will look like "Value=some text" (without the quotes.) Here, the token 'Value' is separated by "=" from its assigned value that the jDDT driver will use. When multiple"tokenized" variables are expected by some Acrion, these are separated by semicolon. For example: "Value={orgName};CompareMode=Contains;Option=IgnoreCase".

As of this writing, the following have been implemented:

ActualValue	Used with the verify verb to provide the actual value for a non-UI verification of some values. The token 'value' is the expected value of the verification.

Append	Used with the sendKeys verb to indicate whether or not to append text in a given textbox or overwrite the existing text (if any.) The default mode is to replace the existing value. Append=true (or Append=yes) will append the specified text to the existing.
Browser	Used with the createWebDriver verb to indicate the name of the browser to use as WebDriver (e.g. CHROME, FIREFOX, IE)
CellsToFind	Used by table verification verbs to indicate the number of cells to find when the default of 1 is not used.
Class	Used with various verification verbs to indicate the class of the actual value when it is not a string (e.g, date, amount, long. Etc.)
Col	Used by table verification to restrict the search to a given column in a table.
CompareMode	Used to indicate how to compare the actual value to the expected value during verification. Options are specific to the type of data being compared. Amount – used to compare amounts (see the comparison modes for numerics below) Boolean – used for functions of binary (true / false) value IsDisplayed, isEnabled, isSelected. Date – used for date verification when comparing dates (not their string representation) Is, equals, equal, ge, greaterThanOrEqualsTo, greaterThanOrEqualTo, >=, =>, onOrAfter, gt, greaterThan, >, after, le, lessThanOrEqualsTo, lessThanOrEqualTo,<=, =<, onOrBefore, lt, lessThan, <, before, ne, notEqual, notEquals, !=, notOn, between (*) Numeric – various numeric comparison are handled as expected using the Integer, Decimal, Long types Is, equals, equal, =, ==, ge, greaterThanOrEqualsTo, greaterThanOrEqualTo, >=, =>, gt, greaterThan, >, le, lessThanOrEqualsTo, lessThanOrEqualTo, <=, =<, lt, lessThan, <, ne, notEqual, notEquals, !=, between (*) *String* (text) – the default verification, use any of the following (case insensitive) Is, equals, equal, contains, contain, notContains, notContain, =, startsWith, startWith, endsWith, endWith, isLowerCase, isUpperCase, Match, Matches. Between (*) (*) For the 'between' option, the value specified in the Value string should appear like two values of that type with the string .and. in between.

	For example “200.and.300” (without the quotes – for numeric) or “05/10/2012.and.06/10/2012” (without the quotes for date.)
Description	Used by the generateReport verb. Provides the description of the report (will appear in the Subject of the email message.)
EmailBody	Used by the generateReport verb. Provides ‘individual’ text to appear in the body of the test result email message.
FileName	Used by various verbs (runCommand, RunJS) to name a script (shell script), batch file or exe file name to be invoked (see Params below)
InputSpecs	Used by the **newTest** verb to specify the type of test item provider as well as the actual source. This is a “!” delimited string with up to three components (type!source!stepsContainer). Example: `InputSpecs=Inline!DemoTestStringsGenerator!calculator`
ItemNo	Used by drop-down option selection logic to find a particular option in a drop-down or list element. Origin: 1.
ItemText	Used by various drop-down / option verbs to indicate the option function relates to the text, not value, aspect of option element.
ItemValue	Used by various drop-down / option verbs to indicate the option function relates to the value, not text, aspect of option element.
JSCode	Used by the RunJS action that executes Java Script code. The value of this token is a java script code with “\n” (new line) standing for statement terminator (;).
Option	Used by various string verification verbs to indicate option like IgnoreCase (Option=IgnoreCase).
PageTitle	Used by the verifyWebDriver and ensurePageLoaded verbs to name the page being verified.
ParamX	Used by the runCommand verb to specify parameters to the invoked command (Param1=this;Param2=that;Param3=etc.)
Ptp	Used by ANY verb. This is used to specify a Post Test Policy. The following have been implemented thus far: QTOF Quit TestCase On Fail. QTOP Quit TestCase On Pass. QTU Quit TestCase Unconditionally. QSOF Quit TestSession On Fail. QSOP Quit TestSession On Pass. QSU Quit TestSession Unconditionally. SOF Skip Steps On Fail. For example: SOF2 means: Skip 2 steps on failure. (*) SOP Skip Steps On Pass. For example: SOP5 means: Skip 5 steps on success. (*) SUN Skip Steps Unconditionally. For example: SUN4 means: Skip 4 steps unconditionally. (*) (*) These policies constitute a rudimentary mechanism for control of flow of testing where steps may be invoked or skipped based on previous step(s) result.
QueryParam	Used by various string verification verbs to name a parameter to either of two element interrogation functions in QryFunction (GetAttribute and GetCssValue). When QryFunction is GetAttribute, QueryParam is the attribute to use in the element query logic

(element.GetAttribute({QueryParam}).
When QryFunction is GetCssValue, QueryParam is the property to use in the element query logic (element.GetCssValue({QueryParam}).

Response — Used by the handleAlert verb to specify whether the alret popup should be accepted or rejected ("accept" or "reject").

Row — Used by various table verification verbs to limit the search to a particular row in the table (origin 1)
RowRange Used by various table verification verbs to indicate the range of rows to consider when searching for cells: RowRange=5-10

SaveAs — Used by various element finding and verifying verbs to save the actual value of a property or function of an element in the jDDT verbs table for subsequent use. For example, when a particular element is found and its getText query function yielded some string (say, with the semantics of Company Name, the specification of SaveAs=TheCompany will set a variable named TheCompany the value of that string. This can later be used for subsequent comparison with the Value={ thecompany} notation to add reuse value to the testing repertoire of jDDT.

SkipToken — Used to provide 'run time' / 'on the fly' step skipping mechanism.
This mechanism is used when several paths through the test sequence use a common sequence within which some step(s) should be executed / skipped conditionally.

Suppose, a given sequence of steps like the following (logically described):

ActionTests

TestNavigation	(sequence of tests for menu navigation)
SetTokens	a ***setVars*** step with "SkipTokens=none" (sans the quotes) in the Data attribute. This step 'sets' the variable SkipTokens to "none".
TestOrder	(sequence of tests for order processing – uses a FinalizeAction sequence – should execute the skippable test step.)
TestPayment	(sequence of tests for payment processing – uses a FinalizeAction sequence – should NOT execute the skippable test step – i.e., should skip it.)
Report	(generate the report for the test cases above.)

Suppose both the FinalizeAction test sequence executes some reusable tests including a step that should be skipped by the TestPayment sequence and executed by the TestOrder sequence.

Finalize

Finalize1	(sequence of tests used by all test cases)
Finalize2	(sequence of tests used by all test cases)

	Skippable	(a test / sequence of tests to be skipped conditionally) This step includes “SkipToken=SkipMeConditionally”(sans the quotes) in the Data attribute.
	Finalize3	(sequence of tests used by all test cases)
	The Order sequence should execute the skippable test and is not shown here because it takes no special action.	
	The Payment sequence can be described logically as:	
	TestPayment	
	Pmt1	(payment test 1 – does some testing of a payment)
	Pmt2	(payment test2 – does some more testing of a payment)
	setVars	“SkipTokens=SkipMeConditionally” in the Data property This sets the SkipTokens string to a value that will cause skipping of the Skippable test(s) because the SkipToken tag in the Data property of the skippable test is contained in the SkipTokens ‘variable’.
	FinalizeAction	(calls the FinalizeAction sequence of tests where the skippable step will be skipped)
StripWhiteSpace	Indicates to the Verifier logic whether or not to strip white spaces prior to comparing strings.	
TabOut	Used by the sendKeys verb to indicate whether to tab out of an input control (needed by some pages to invoke some java script.	
Tag	Used by various table verification verbs to add flexibility to finding element nested within table cells. The table finding logic will look only at elements with the named tag specified (e.g. Tag=td)	
Type	Indicates the type of scrolling to use with the **scrollWebPage** Action.	
	ScrollBy	Scrolls the web page by a specific X, Y (pixel-wise)offset – both x and y must be specified as integers.
	ScrollTo	Scrolls the web page to a specific X, Y coordinate (pixel-wise) both x and y must be specified as integer or ‘max’
	ScrollIntoView	Scrolls the web page to a specific web element – the locType and locSpecs attributes must be specified.
Url	Used by the createWebDriver and navigateToPage verbs to specify the URL to navigate to.	
Value	Used verbs to specify Action specific value. For example Value=Some Choice Text with Action of sendKeys will enter the text “Some Choice Text” (without the quotes) into the (presumed enabled) text box located by that step.	
	Note: When comparison mode is ‘between’, Value should appear like two values of that type with the string .and. in between. For example “200.and.300” (without the quotes – for numeric) or “05/10/2012.and.06/10/2012” (without the quotes for date.)	

WaitInterval	Optionally used with various element finding verbs to specify the interval between element location attempts. A setting of 100 milliseconds is used as a default so this need not specified at all and provides an extra granularity for handling dynamic (ajax) pages. Used in conjunction with WaitTime.
WaitTime	Optionally used with various element finding verbs to specify the number of seconds to wait for a page or an element to render. This should only be used when it is needed to override the global value that is used in the ddt.properties file. Used in conjunction with WaitInterval that specifies how many milliseconds to wait between attempts to locate the element (page).
TotalWaitTime	Used with verbs that use repeated polling of an element (WaitUntil). Specify the total number of seconds to wait for a an element to appear. This should only be used in dynamic web pages where the same element keeps changing (a carouselle for instance. Used in conjunction with WaitInterval that specifies how many milliseconds to wait between attempts to locate the element (page).
X	Used by the setPageSize verb to indicate the desired width of the page (in pixels) Used by the scrollWebPage verb to indicate the horizontal scroll position (in pixels) – Applies to ScrollBy and ScrollTo types.
Y	Used by the setPageSize verb to indicate the desired height of the page (in pixels) Used by the scrollWebPage verb to indicate the vertical scroll position (in pixels) – Applies to ScrollBy and ScrollTo types.

Description

This property is used for documentation purposes alone and is self explanatory. While this property does not have any verification value, it is extremely useful when properly used in order to track errors as well as scope of testing

.

Action Vocabulary

This section describes the current 'Action Vocabulary' (AKA 'verbs') of jDDT.

The term 'Action Vocabulary' is synonymous with the repertoire of capabilities supported by the jDDT framework.

Each entry in the 'Action Vocabulary' corresponds to a method in the Vocabulary class of the jDDT driver.

Each example is provided in both formats, namely, spreadsheet (xls or xlsx) and XML. For the sake of brevity, the HTML format is omitted.

The following measures were taken for brevity and space considerations:

1. The Active column has been omitted from all examples for brevity. When wishing to skip a given test step, just place "no" (without the quotes) in the Active column of that step. This can be very useful for saving time when constructing test cases by deactivating stable / tested portion of the test harness.
2. Irrelevant columns were omitted to enhance clarity.
3. Some tokens of the Data property have been omitted from many verbs wher they may be used.
   For example, overriding of the global WaitTime and WaitInterval values or setting the test item's Ptp (Post Test Policy) which may apply to any test were omitted from some (or all) of the verbs explained below
4. HTML and Inline Inputs are omitted.

Please note that the Action must be spelled accurately in the Action property of the input source but is not case sensitive.

Following is a sample html input with two steps (all attributes should be specified even when empty):

```
<!DOCTYPE html>
<html>
   <head lang="en">
      <meta charset="UTF-8">
      <title>DDT Demo Root</title>
   </head>
   <body>
      <table>
         <tbody>
            <tr>
               <td class="id">DDTDemo01</td>
               <td class="action">newTest</td>
               <td class="locType"></td>
               <td class="locSpecs"></td>
               <td class="qryFunction"></td>
               <td class="active"></td>
               <td class="data">FileName=DDTRoot.xls;WorksheetName=Calculate</td>
```

```
                <td class="description">Run the Calculator tests</td>
             </tr>
             <tr>
                <td class="id">DDTDemo02</td>
                <td class="action">newTest</td>
                <td class="locType"></td>
                <td class="locSpecs"></td>
                <td class="qryFunction"></td>
                <td class="active"></td>
                <td class="data">FileName=DDTRoot.xls;WorksheetName=ChainingFinders</td>
                <td class="description">Run the Chaining Finders tests</td>
             </tr>
          </tbody>
       </table>
    </body>
 </html>
```

Following is a sample JSON input with two steps. Only used attributes need be specified. Please note that the string "TestItems" is mandatory & case sensitive:

```
{
"TestItems": [
   {
   "id":"JSONTest#",
   "action": "NewTest",
   "data": "InputSpecs=File!DDTRoot.xlsx!Root",
   "description": "Run the first calculation test scenario"
   },
   {
   "id":"JSONTest#",
   "action": "Click",
   "locType": "id",
   "locSpecs": "submitButton",
   "description": "Click the Submit Button"
   }]
}
```

BranchOnElementValue

BranchOnElementValue provides a mechanism for controlling the flow of testing based on the contents of a particular web element on a page. This is mostly useful in presence of a dynamic web page when the locator of a given web element and the value of any of the verifieable attributes is one of a known list of values.

This action makes use of the following attributes:

Action	BranchOnElementValue
LocType	Specifies the type of locator to use Example: id.
LocSpecs	Specifies the value of locator to use. Example: submit-button
QryFunction	The element interrogation function to be used for locating the element to be evaluated.
Data	The following tokens are used to specify what to look for and where to handle it:
Iterations	Number of iterations to attempt to find a match for any of the options (4 in the example below)
TotalWaitTime	Total time to wait for some match (across possible options).
WaitTimeInterval	Number of milliseconds to wait between iterations while waiting for one of the options to match on.
Branches	Input bar (|) delimited specification(s) indicating where to find steps for handling a given match – when an option was matched. In the example below various worksheets are specified in the demo.xlsx spreadsheet (File!Demo.xlsx!Bards|File!Demo.xlsx!Spock|File!Demo.xlsx!Journey|File!Demo.xlsx!Einstein|File!Demo.xlsx!Electoloom|File!Demo.xlsx!Kettle)
Values	Bar (|) delimited list of values to match on. In the example below the following strings are specified “The Bard|Spock|Journey|Einstein|Electroloom|Kettle”
Queries	Bar (|) delimited list of element interrogation functions for extracting the value to match on. In the example below GetText is used for all options.
Comparisons	Bar (|) delimited list of comparison modes to used for verifying the desired value. In the example below “contains” is used for all options.

Excel Example

In the example below, the element with id of “item-span” may contain Bard or Spock or Journey or Einstein or Kettle. The element.getText() function is to be used and if the string contains one of the values specified, the corresponding worksheet is to be launched.

Id	Action	LocType	LocSpecs	QryFunction	Data
BranchExample	**BranchOnElementValue**	**id**	Item-span	**getText**	**Iterations=4;TotalWaitTime=50;WaitInterval=1000;Branches=File!Demo.xlsx!Bards|File!Demo.xlsx!Spock|File!Demo.xlsx!Journey|File!Demo.xlsx!Einstein|File!Demo.xlsx!Electoloom|File!Demo.xlsx!Kettle;Values=The Bard|Spock|Journey|Einstein|Electroloom|Kettle;Queries=GetText|GetText|GetText|GetText|GetText|GetText;Comparisons=Contains|Contains|Contains|Contains|Contains|Contains**

XML Example

```
<toBeCompletedLater>
```

Click

Click provides a generic mouse click on a UI element.

This action makes use of the following attributes:

Action	click
LocType	Specifies the type of locator to use Example: id.
LocSpecs	Specifies the value of locator to use. Example: submit-button

Excel Example

ID	Action	LocType	LocSpecs	Description
LOG06	Click	Id	submit-button	Click the Submit button

XML Example

```
<TestItem ID="LOG06" Action="click" LocType="id" LocSpecs="submit-button" Description="click the submit button"/>
```

ClickCell

ClickCell provides a generic mouse click on a UI element of type cell (tag td) within a table. This action commences with the findCell logic from the top of the table. When the searching mechanism applies to more than one cell, the first cell is used and gets clicked.

This action makes use of the following attributes:

Action	clickCell

LocType	Specifies the type of locator to use Example: xpath.	
LocSpecs	Specifies the value of locator to use. Example: /html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/div/div/table/tbody	
	NOTE: The locator type and specs should point to the <tbody> (or <table>) element containing the cell as this is where the cell location logic is anchored.	
	When a cell in a table has fixed locator and specs, one may use the 'click' Action with a direct reference to the cell.	
QryFunction	Specifies how to query each cell during the search logic.	
Data	Specifies the value to look for during cell searching logic and the name of the tag (typically, td) to locate columns within a row and, potentially, the row, rowrange and column to restrict the search to.	
	The following properties can add verb-specifi c semantics to this Action:	
	Col	Limits the search for cell to a particular column in the table (origin 1). Example: Col=3
	CompareMode	Specifies how to compare the element's value to the expected value to find a cell. Example: CompareMode=StartsWith.
	QueryParam	When QryFunction is GetAttribute, QueryParam is the attribute to use in the element query logic (element.GetAttribute({QueryParam}). When QryFunction is GetCssValue, QueryParam is the property to use in the element query logic (element.GetCssValue({QueryParam}).
	Row	Limits the search for cell to a particular row the table (origin 1). Example: Row=3
	RowRange	Limits the search for cell to a particular (contiguous) range of rows in the table (origin 1). Example: RowRange=2-10
	Tag	Indicates the tag used to get items within the cell (in case of complex table cells containing child elements) Example: Tag=div
	Value	Expected value to use in order to locate cell. Example: Value=Johnson & Johnson
	WaitTime	Used to override the default wait time – rarely used in this instance (same for WaitInterval)

Excel Example (new lines are used for each Data attribute for visual clarity)

ID	Action	LocType	LocSpecs	QryFunction	Data
Step6	clickCell	xpath	/html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/div/div/table/tbody	GetText	Value={Notes}; Col=2; Tag=td; CompareMode=Contains

XML Example

```
<TestItem ID="Step6" Action="clickCell" LocType="id" LocSpecs="table_body" QryFunction="GetText"
Data="Value={Notes};Col=2;Tag=td;CompareMode=Contains" Description = "Find and click Cell in column 2 of the table"/>
```

CreateWebDriver

This action is used to create a web driver object opened on a particular URL.

This action makes use of the following attributes:

Action — createWebDriver

Data — Specifies the browser (if one wishes to override the value in the ddt.properties file) and the URL to open the web driver on..

The following properties can add verb-specifi c semantics to this Action:

Browser — Indicates what browser to use (FIREFOX, IE, CHROME).
Example: Browser=CHROME

URL — Specifies the URL to open the browser on.
Example:http://www.google.com

Excel Example

ID	Action	Data
Step6	createWebDriver	Browser=CHROME;URL=http://www.mycompany.com

XML Example

```
<TestItem ID="Step6" Action="createWebDriver" Data="Browser=CHROME;URL=http://www.mycompany.com" Description = "Open Browser URL"/>
```

EnsurePageLoaded

Verifies the expected page is loaded.

This action makes use of the following attributes:

Action — ensurePageLoaded

Data — Specifies the value of the page title in either “PageTitle” or “Value” attribute.

The following properties can add verb-specifi c semantics to this Action:

PageTitle	Indicates the title of the page to verify loading of (use either this or Value). Example: PageTitle=Welcome To Our Company
Value	Indicates the title of the page to verify loading of (use either this or PageTitle) Example: PageTitle=Welcome To Our Company
WaitTime	Used to override the default wait time. Example: WatiTime=10
WaitInterval	Used to override the default wait interval. Example: WatiInterval=50

Excel Example

ID	Action	Data
Step6	**ensurePageLoadedl**	**Value=Welcome Partner;WaitTime=8;WaitInterval=300**

XML Example

```
<TestItem ID="Step6" Action="ensurePageLoaded" Data="PageTitle={PageTitle};WaitInterval=200 " Description = "Ensure page {pagetitle} loaded"/>
```

FindCell

FindCell provides a generic finding mechanism of a cell (tag td) within a table. This action is used by other table cell actions. When the searching logic applies to more than one cell and only one cell is being 'requested' the first cell found gets returned.

This action makes use of the following attributes:

Action	findCell
LocType	Specifies the type of locator to use Example: xpath.
LocSpecs	Specifies the value of locator to use. Example: /html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/div/div/table/tbody **NOTE:** The locator type and specs should point to the <tbody> (or <table>) element containing the cell as this is where the cell location logic is anchored. When a cell in a table has fixed locator and specs, one may use the 'click' Action with a direct reference to the cell.
QryFunction	Specifies how to query each cell during the search logic.

Data	Specifies the value to look for during cell searching logic and the name of the tag (typically, td) to locate columns within a row and, potentially, the row, rowrange and column to restrict the search to. The following properties can add verb-specifi c semantics to this Action:	
	CellsToFind	Indicates the number of cells to find for the specified search criteria. Example: CellsToFind=20
	Col	Limits the search for cell to a particular column in the table (origin 1). Example: Col=3
	CompareMode	Specifies how to compare the element's value to the expected value to find a cell. Example: CompareMode=StartsWith.
	Opt	Specifies an optional way to consider when using the CompareMode (such as IgnoreCase=tru) Example: IgnoreCase=true
	QueryParam	When QryFunction is GetAttribute, QueryParam is the attribute to use in the element query logic (element.GetAttribute({QueryParam}). When QryFunction is GetCssValue, QueryParam is the property to use in the element query logic (element.GetCssValue({QueryParam}).
	Row	Limits the search for cell to a particular row the table (origin 1). Example: Row=3
	RowRange	Limits the search for cell to a particular (contiguous) range of rows in the table (origin 1). Example: RowRange=2-10
	Tag	Indicates the tag used to get items within the cell (in case of complex table cells containing child elements) Example: Tag=div
	Value	Expected value to use in order to locate cell. Example: Value=Johnson & Johnson
	WaitTime	Used to override the default wait time – rarely used in this instance (same for WaitInterval)

Excel Example (new lines are used for each Data attribute for visual clarity)

The semantics of this example is "look for a cell in a table that starts at the locator indicated here, use the GetText function of each child element with tag 'td' to find the actual value indicated in the Value token of the Data property and based on the CompareMode (and, if provided, Option) determine if the cell qualifies". In the example shown, the contents of a previously set variable named 'Selection' is the expected value.

ID	Action	LocType	LocSpecs	QryFunction	Data

Step6	**findCell**	**xpath**	**/html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/div/div/table/tbody**	**GetText**	**Value={selection} Tag=td;Option=IgnoreCase;CompareMode=Contains**

XML Example

```
<TestItem ID="Step6" Action="findCell" LocType="id" LocSpecs="table_body" QryFunction="GetText"
Data="Value={Notes};Col=2;Tag=td;CompareMode=Contains" Description = "Find Cell in column 2 of the table"/>
```

FindElement

FindElement provides a generic finding mechanism of an element within a web page. This action is used by other element-related actions.

This action makes use of the following attributes:

Action	findElement
LocType	Specifies the type of locator to use Example: id.
LocSpecs	Specifies the value of locator to use. Example: submit-button
WaitTime	Used to override the default wait time. Example: WaitTime=12
WaitInterval	Used to override the default wait interval. Example: WaitInterval=80

Excel Example

The semantics of this example is “look for an element at the locator indicated here, use the GetText function of the element to find the actual value indicated in the Value token of the Data property and based on the CompareMode (and, if provided, Option) determine if the element is the desired one. In the example shown, the contents of a previously set variable named ‘ButtonName’ is the expected value and exact comparison (is) should be used.

ID	Action	LocType	LocSpecs	Data
Step6	**findElement**	**id**	**submit-button**	**WaitTime=3**

XML Example

```
<TestItem ID="Step6" Action="findElement" LocType="id" LocSpecs="submit-button" Data="WaitTime=3" Description = "Find the submit
button"/>
```

FindOption

findOption provides a generic finding mechanism of an option element within a list of <option/> elements. Please note that the query mechanism for this verb is determined by the Data token ItemText, ItemValue or ItemNo indicating whether to look for the expected value (typically, Data token Value) in an option's Item Text or Item Value – or otherwise to select item number ItemNo.) When specifying the search logic, use exactly one of the three options.

This action makes use of the following attributes:

Action	findOption
LocType	Specifies the type of locator to use Example: xpath.
LocSpecs	Specifies the value of locator to use. Example: /html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/div/div/ **NOTE:** The locator type and specs should point to the <select> element containing the list of option elements to which the option element is anchored.
Data	The following properties can add verb-specifi c semantics to this Action:

CompareMode	Specifies how to compare the element's value to the expected value to find a cell. Example: CompareMode=StartsWith.
ItemNo	If used, specifies the option number to select. Mutually exclusive with ItemText and ItemValue. Example: ItemNo=2.
ItemText	Specifies that the option should be searched by the item's text (not value). Mutually exclusive with ItemNo and ItemValue. Example: ItemText=Alaska.
ItemValue	Specifies that the option should be searched by the item'svalue (not text). Mutually exclusive with ItemNo and ItemText. Example: ItemValue=AK.
WaitTime	Used to override the default wait time. Example: WaitTime=12
WaitInterval	Used to override the default wait interval. Example: WaitInterval=80

Excel Example

The semantics of this example is "look for an option element in a drop down that starts at the locator indicated here, use either the ItemText or ItemValue function of each child element to find the actual value indicated in the Data property and based on the CompareMode (and, if provided, Option) determine if the item qualifies. In the example shown, the contents of a previously set variable named 'StateCode' is the expected value to be found.

ID	Action	LocType	LocSpecs	Data
Step6	findOption	xpath	/html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/div/div/select	ItemValue={StateCode};CompareMode=Is

XML Example

```
<TestItem ID="Step6" Action="findOption" LocType="id" LocSpecs="select_list" QryFunction="GetText"
Data="ItemValue={StateCode};CompareMode=Is" Description = "Find the {StateCode} option in the state dropdown"/>
```

GenerateReport

This action causes generation of the report as of this point in the testing session. Any given testing session may generate report at any step in the process. Typically, this is done at the end of some functional testing point (say, the Order Creation set of scenarios.)

This action makes use of the following attributes:

Action — generateReport.

Data — The following properties can add verb-specifi c semantics to this Action:

- Description — Names the report.
  Example: Description=Results for the Order Creation tests.
- EmailBody — Provides an individual email body text to be sent to the recipients of the results by email (if any.)
  Example: Description=Results for the Order Creation tests.

Excel Example

ID	Action	Data
Step6	generateReport	Description=Test Results for the {MenuOption} menu tests;EmailBody=Jack, please forward to all stakeholders in your dept.

XML Example

```
<TestItem ID="Step6" Action="generateReport" Data="Description=Test Results for the {MenuOption} menu tests;EmailBody=Jack, please
forward to all stakeholders in your dept." Description = "Generate the report for the {MenuOption} menu tests"/>
```

HandleAlert

This action provides for handling of alerts during interaction with the application.

This action makes use of the following attributes:

Action — handleAlert.

Data — The following properties can add verb-specifi c semantics to this Action:

CompareMode	Indicates to jDDT verification mechanism how to compare the expected value (see Value below) to the actual text of the alert. Example: CompareMode=Contains.
Option	Indicates some option to be considered by the verification logic when validating the text of the Alert message. For example: Option=ignorecase.
Response	The response to the alert. This should be either "accept" or "Reject" (case-insensitive) For example: Response=Reject.
Value	This is the text provided by the alert popup (verified by jDDT)

Excel Example

ID	Action	Data
Step6	**handleAlert**	**Response=Reject;Value=Are you sure that;CompareMode=StartWith;Option=IgnoreCase**

XML Example

```
<TestItem ID="Step6" Action="handleAlert" Data="Response=Reject;Value=Are you sure that;CompareMode=StartWith;Option=IgnoreCase"/>
```

Hover

Hover provides a generic element hovering functionality. This is useful when element of a given page exposes some UI / properties / behavior only after it is hovered upon by a mouse.

This action makes use of the following attributes:

Action	Hover
LocType	Specifies the type of locator to use Example: id.
LocSpecs	Specifies the value of locator to use. Example: submit-button
WaitTime	The number of seconds to hover on the element

Excel Example

ID	Action	LocType	LocSpecs	Data	Description
LOG06	**Hover**	**Id**	**submit-button**	**WaitTime=2**	**Hover over the Submit button for two seconds**

XML Example

```
<TestItem ID="LOG06" Action="hover" LocType="id" LocSpecs="submit-button" data="WaitTime=2" Description="click the submit button"/>
```

Maximize

This action provides for maximizing a web page.

This action makes use of the following attributes:

Action	maximize

Excel Example

ID	Action	Data
Step6	maximize	

XML Example

```
<TestItem ID="Step6" Action="maximize"/>
```

NavigateToPage

This action provides for page navigation from the current page.

This action makes use of the following attributes:

Action	navigatetoPage	
Data	The following properties can add verb-specifi c semantics to this Action:	
	URL	The URL to navigate to. Example: Url=http://mycompany.com/mypage/or/anotherpage

Excel Example

ID	Action	Data
Step6	navigateToPage	URL=https://SomeCompany.Com/SomePageOr/AnotherPage

XML Example

```
<TestItem ID="Step6" Action="navigateToPage" Data="Response=Reject;Value=Are you sure that;CompareMode=StartWith;Option=IgnoreCase"/>
```

NewTest

This action provides for running a group of test steps as a new test at any given step. This is where much of the reusability power of jDDT becomes evident.

This action makes use of the following attributes:

Action	newTest
Data	The following properties can add verb-specifi c semantics to this Action:
InputSpecs	A colon-delimited string used to specify the input type (“file” or “inline” as of this writing) as well as the container (file or inline class name) and section (worksheet or inline method name) Example: InputSpecs=File:Enter Order.xlsx:EnterOrder1. Example: InputSpecs=File:Root.xml. Example: InputSpecs=Inline:DefaultInlineItemsGenerator:root.
	Please note that worksheet names are case sensitive and “enter Order” is not the same as “Enter Order”

Excel Example

ID	Action	Data
Step6	newTest	InputSpecsFile:{$DataFolder}Demo.xlsx:Root

XML Example

```
<TestItem ID="Step6" Action="newTest" Data=" InputSpecsFile:{$DataFolder}Demo.xlsx:Root"/>
```

Quit

This action causes the existing web driver (browser) to close and quit. The web session is terminated.

This action makes use of the following attributes:

Action	quit

Excel Example

ID	Action	Description
Step6	quit	Close the broqwser and the web session

XML Example

```
<TestItem ID="Step6" Action="quit" Description="Close the broqwser and the web session"/>
```

RefreshSettings

This action forces a refresh of the project settings values to their original values as reflected in the project’s ddt.properties file. This is useful when the ddt.properties file was changed as could be the case when switching settings within a given test session. One example is testing the same harness on two different browsers during the same session.

This action makes use of the following attributes:

Action refreshSettings

Excel Example

ID	Action	Description
Step6	refreshSettings	Reset project settings to the values in the ddt.proprties file

XML Example

```
<TestItem ID="Step6" Action="refreshSettings" Description="Reset project settings to the values in the ddt.proprties file"/>
```

RunCommand

This action provides for running an operating system executable, batch file, shell script, etc. at any given test step. This avails jDDT with all sorts of functionality outside of the Selenium realm (e.g. test environment setup and cleanup, etc.) Parameters to the file being invoked can be provided using the Data attribute Param1, Param2, … ParamX (as many parameters as the invoked file expects.) Make sure though that parameters are executive (it is inappropriate to use Param3 without using Param1 and Param2.)

This action makes use of the following attributes:

Action runCommand

Data The following properties can add verb-specifi c semantics to this Action:

- FileName — Name of the file to be invoke. This must reference a file that the operating system can run. You can use %script% to indicate the path to the project's Scripts folder. You can use the variable {$ScriptsFolder} for that purpose as well
  For example: FileName=%script%setup.bat.
- Param1 — Optionally – provides the first parameter to the file being invoked
  For example: Param1=-a. The –a option should be meaningful to the Setup.bat module.
- Param2 — Optionally – provides the second parameter to the file being invoked
  For example: Param2=-b. The –b option should be meaningful to the Setup.bat module.

Excel Example

ID	Action	Data
Step6	runCommand	FileName={ScriptsFolder}Setup.bat;Param1=-a;Param2=-b

XML Example

```
<TestItem ID="Step6" Action="runCommand" Data="FileName=($ScriptsFolder}Setup.bat;Param1=-a;Param2=-b"/>
```

RunJS

This action provides for running a Java Script that is either provided in the Data attribute or is contained and a specified file.

This action makes use of the following attributes:

Action	runJS	
Data	The following properties can add verb-specifi c semantics to this Action:	
	FileName	(optionally) Name of a file containg valid java script code. For example: FileName=%script%myFunction.js
	JSCode	Optionally – a string of java script code where instances of "\n" are replaced by ";" (statement terminator). For example: JSCode= window.scrollTo(150,300)\n

Excel Example

ID	Action	Data
Step6	runJS	FileName={ScriptsFolder}MyCode.js
Step7	runJS	JSCode=window.scrollTo(150,300)\n

XML Example

```
<TestItem ID="Step6" Action="runJS" Data="FileName={ScriptsFolder}myFunction.js"/>
<TestItem ID="Step7" Action="runJS" Data="JSCode window.scrollTo(150,300)\n"/>
```

SaveElementProperty

saveElementProperty is used to save the value of some property of a web element in the variables dictionary. This enable the test creator to compare values of a given property during verification or any other purpose (data entry, etc.) Please note that the query mechanism for this verb is determined by the Data token ItemText or ItemValue – therefore, these are omitted from the setup provided below.

This action makes use of the following attributes:

Action	saveElementProperty
LocType	Specifies the type of locator to use Example: id.
LocSpecs	Specifies the value of locator to use. Example: grant-type-selection-dropdown

Data	The following properties can add verb-specifi c semantics to this Action:	
	CompareMode	Specifies how to compare the element's value to the expected value to find a web element. Example: CompareMode=Contains.
	SaveAs	Mandatory – provides a name to save the value of the property for future reference. Example: SaveAs=theFont.

Excel Example

The semantics of this example is "look for an option element in a drop down that starts at the locator indicated here, use either the ItemText or ItemValue function of each child element to find the actual value indicated in the Data property and based on the CompareMode (and, if provided, Option) determine if the item qualifies. The first qualified item is clicked in order to be selected. In the example shown, the contents of a previously set variable named 'TypeCode' is the expected value.

ID	Action	LocType	LocSpecs	Data
Step6	saveElementProperty	id	order-number	SaveAs=Order-Number;CompareMode=Is

XML Example

```
<TestItem ID="Step6" Action="saveElementProperty " LocType="id" LocSpecs="order-number" Data="SaveAs=Order-Number;CompareMode=Is"
Description = "Save the order number value in the Order-Number variable"/>
```

SaveWebDriverProperty

SaveWebDriverProperty is used to save the value of some property of a web driver (page) in the variables dictionary. This enables the test creator to compare values of a given web driver property during verification.

This action makes use of the following attributes:

Action	saveElementProperty	
LocType	Specifies the type of locator to use Example: id.	
LocSpecs	Specifies the value of locator to use. Example: grant-type-selection-dropdown	
Data	The following properties can add verb-specifi c semantics to this Action:	
	CompareMode	Specifies how to compare the element's value to the expected value to find a web element. Example: CompareMode=Contains.

	SaveAs	Mandatory – provides a name to save the value of the property for future reference. Example: SaveAs=theFont.

Excel Example

The semantics of this example is “save the title of the current page and save it in a variable named MainPageTitle.

ID	Action	QryFunction	Data
Step6	saveWebDriverProperty	getTitle	SaveAs=MainPageTitle

XML Example

```
<TestItem ID="Step6" Action="saveWebDriverProperty " QryFunction=”GetTitle” Data="SaveAs=MainPageTitle" Description = "Save the page title"/>
```

ScrollWebPage

scrollWebPage provides a web page scrolling mechanism.

Three types of scrolling are supported, namely:

- ScrollBy This type is used to scroll the page by a fixed number of pixels offset from the present position.
- ScrollTo This type is used to scroll the page to a particular coordinate specified in pixels wher either the x or the y coordinate can be ‘max’;
- ScrollIntoView This type is used to scroll the page to a particular element in order to ensure it is in view. The

This action makes use of the following attributes:

Action	scrollWebPage	
LocType	Specifies the type of locator to use when using the ScrollIntoView option (then mandatory, otherwise, ignored.) Example: id	
LocSpecs	Specifies the value of locator to use when using the ScrollIntoView option (then mandatory, otherwise, ignored.) Example: grant-type-selection-dropdown	
Data	The following properties can add verb-specifi c semantics to this Action:	
	Type	Specifies the type of scroll to perform. Example: Type=ScrollBy. Example: Type=ScrollTo. Example: Type=ScrollIntoView.

X	Used to specify either horizontal (pixel) offset or specific horizontal pixel value. Example: X=20.
Y	Used to specify either vertical (pixel) offset or specific vertical pixel value. Example: Y=20.

Excel Example

The semantics of step5 is “scroll the web page all the way down”.

The semantics of step6 is “scroll the web page to the element so specified”.

ID	Action	LocType	LocSpecs	Data	Description
Step5	scrollWebPage			Type=ScrollTo;X=0;Y=Max	Scroll the web page all the way down
Step6	scrollWebPage	id	grant-type-selection-dropdown	Type=ScrollIntoView	Scroll the web page to the element

XML Example

```
<TestItem ID="Step5" Action="scrollWebPage" Data="Type=ScrollTo;X=0;Y=Max" Description="Scroll the web page all the way down"/>
<TestItem ID="Step6" Action="scrollWebPage" LocType="id" LocSpecs="grant-type-selection-dropdown" Data="Type=ScrollIntoView"
Description = "Scroll the web page to the element"/>
```

SelectOption

selectOption provides a generic selection mechanism of an option element within a list of <option/> elements. This verb makes use of the findOption verb so its invocation is quite similar. Please note that the query mechanism for this verb is determined by the Data token ItemText or ItemValue – therefore, these are omitted from the setup provided below.

This action makes use of the following attributes:

Action	selectOption
LocType	Specifies the type of locator to use Example: id.
LocSpecs	Specifies the value of locator to use. Example: grant-type-selection-dropdown **NOTE:** The locator type and specs should point to the <select> element containing the list of option elements to which the option element is anchored.
Data	The following properties can add verb-specifi c semantics to this Action:
CompareMode	Specifies how to compare the element’s value to the expected value to find a cell. Example: CompareMode=StartsWith.

ItemNo	If used, specifies the option number to select. Mutually exclusive with ItemText and ItemValue. Example: ItemNo=2.
ItemText	Specifies that the option should be searched by the item's text (not value). Mutually exclusive with ItemNo and ItemValue. Example: ItemText=Alaska.
ItemValue	Specifies that the option should be searched by the item'svalue (not text). Mutually exclusive with ItemNo and ItemText. Example: ItemValue=AK.
WaitTime	Used to override the default wait time. Example: WaitTime=12
WaitInterval	Used to override the default wait interval. Example: WaitInterval=80

Excel Example

The semantics of this example is "look for an option element in a drop down that starts at the locator indicated here, use either the ItemText or ItemValue function of each child element to find the actual value indicated in the Data property and based on the CompareMode (and, if provided, Option) determine if the item qualifies. The first qualified item is clicked in order to be selected. In the example shown, the contents of a previously set variable named 'TypeCode' is the expected value.

ID	Action	LocType	LocSpecs	Data
Step6	selectOption	id	grant-type-selection-dropdown	ItemValue={TypeCode};CompareMode=Is

XML Example

```
<TestItem ID="Step6" Action="selectOption" LocType="id" LocSpecs="grant-type-selection-dropdown"
Data="ItemValue={TypeCode};CompareMode=Is" Description = "Select the {TypeCode} option in the grant type dropdown"/>
```

SendKeys

sendKeys provides a generic data entry mechanism into text boxes.

This action makes use of the following attributes:

Action	sendKeys
LocType	Specifies the type of locator to use Example: id.

LocSpecs	Specifies the value of locator to use. Example: notes-text-box	
Data	The following properties can add verb-specific semantics to this Action:	
	Append	Indicates whether to overwrite the current contents of the text box (the default behavior) or to append text to the current contents. Example: Append=true.
	Value	Specifies the value (test) to enter into the text box. Example: Value=Please verify the tax id ({taxId}) for organization {orgName}
	TabOut	Used to override the default behavior indicated in the ddt.properties file of the project. If this evaluates to true – the jDDT driver will tab out of the text box field after entering data into it. Example: TabOut=false

Excel Example

ID	Action	LocType	LocSpecs	Data
Step6	sendKeys	id	notes-text-box	Value= Please verify the tax id ({taxId}) for organization {orgName};Tabout=false

XML Example

```
<TestItem ID="Step6" Action="sendKeys" LocType="id" LocSpecs="notes-text-box" Data="Value= Please verify the tax id ({taxId}) for organization {orgName};Tabout=false" Description = "Enter the indicated text in the Notes field"/>
```

SetPageSize

setPageSize is used to control the size of a web page. This action is applied to the current Web Driver as opposed to an element on a page.

This action makes use of the following attributes:

Action	setPageSize	
Data	The following properties can add verb-specific semantics to this Action:	
	x	the number of pixels to set the width of the page to. Example: x=1700.
	y	the number of pixels to set the height of the page to. Example: x=2700.

Excel Example

ID	Action	Data

Step6	setPageSize	X=1200;y=2400

XML Example

```
<TestItem ID="Step6" Action="setPageSize" Data="X=1200;y=2400" Description = "Set the page size to {x} by {y} pixels"/>
```

SetVars

setVars is used to set 'variables' whose values can be used during a test session. This feature is one of the ways to promote reusability. Following setting of a variable named(for instance) MyVar – say with a value of 'Some Value' – one may subsequently use {myvar} (case insensitive) to have the value of 'Some Value' replace that token. This can be applied to any of the properties of a test item. Any number of variables can be set at one time using the format of {VariableName}={VariableValue}

This action makes use of the following attributes:

Action	setVars
Data	The Data attribute for this verb is a series of variable name, '=', variable value as in: Var1=Value of Var 1;Amount=425.34;Etc=etc., etc., etc. With this example wherever {var1} is used, the value 'Value of var 1' will be used, wherever {etc} is used, the value of 'etc., ect., etc.' will be used.

Excel Example

ID	Action	Data
Step6	setVars	Var1=Value of Var 1;Amount=425.34;Etc=etc., etc., etc.

XML Example

```
<TestItem ID="Step6" Action="setars" Data="Var1=Value of Var 1;Amount=425.34;Etc=etc., etc., etc." Description = "Set the variables for the next bunch of tests"/>
```

SwitchTo

SwitchTo is used to have the web page switch to a different frame or web page when the application calls for it.

This action makes use of the following attributes:

Action	switchTo
Data	The following properties can add verb-specific semantics to this Action:

ItemType	The type of switching to do “frame” (the default) or “page”. Example: itemType=frame.
Value	The name of the frame or child web page to switch to. Example: value=The Other Frame.

Excel Example

ID	Action	Data
Step6	switchTo	ItemType=Frame;Value=The Next Frame

XML Example

```
<TestItem ID="Step6" Action="switchTo" Data="ItemType=Frame;Value=The Next Frame" Description = "Switch to the other frame"/>
```

TakeScreenShot

takeScreenShot is used to take a screen shot of the the current web page (a web page should be displayed for this verb to work). The image taken of the current web page is stored in the project;s Images folder and is named with the current time stamp appended to the TestItem id. For example, a screen shot taken at step MakeAGrant02 on March 20, 2014 around 2:13 PM may be named “MakeAGrant02 – 20140320-141352.png”

This action makes use of the following attributes:

Action	takeScreenShot

Excel Example

ID	Action
Step6	takeScreenShot

XML Example

```
<TestItem ID="Step6" Action="takeScreenShot"/>
```

Toggle

toggle provides a generic mechanism for toggling an element such as check box or radio button from its current state to the state indicated by the value of the Value token of the Data attribute. Any element that responds to the Toggled function can be the subject of using this verb.

This action makes use of the following attributes:

Action	toggle
LocType	Specifies the type of locator to use Example: id.

LocSpecs	Specifies the value of locator to use. Example: my-check-box	
Data	The following properties can add verb-specific semantics to this Action:	
	Value	Specifies whether to leave the element toggled or not. Example: Value=false

Excel Example

ID	Action	LocType	LocSpecs	Data
Step6	toggle	id	my-ceck-box	Value=true

XML Example

```
<TestItem ID="Step6" Action="toggle" LocType="id" LocSpecs="my-check-box" Data="Value=true" Description="Check on my-check-box"/>
```

Verify

verify provides a generic verification / comparison of two values that are not necessarily UI-bound. This can be useful when one saves some UI element's property before and after some action and then compare / verify the 'before' value to the 'after' value (for example, the number of rows in a table following creation of another record that is displayed in that table – the 'after' value of the row count should be larger than the 'before' count.

This action makes use of the following attributes:

Action	verify	
Data	The following properties can add verb-specifi c semantics to this Action:	
	Value	Stands for the expected value of the comparison / verification.
	ActualValue	Stands for the expected value of the comparison / verification.
	Class	Specifies the class of the objects to compare when the values are not strings (text.)
	CompareMode	The comparison mode between Value and ActualValue (see the comparison modes discussion.)

Excel Example

The semantics of this example is "look for an option in a drop-down that starts at the locator indicated here, use the isDisplayed function of each child element found by using either the ItemText or ItemValue specified to find the desired option considering the CompareMode (and, if provided, Option) determine if the cell qualifies. In the illustration below, the step succeeds if an option with itemValue of {TypeCode} is found and its isDisplayed function returns true. In the example shown, the contents of a previously set variable named 'TypeCode' is the value by which to find an option using ItemValue.

ID	Action	LocType	LocSpecs	QryFunction	Data
Step6	verifyOption	id	grant-type-selection-dropdown	isDisplayed	ItemValue={TypeCode};CompareMode=Is;Value=true

XML Example

```
<TestItem ID="Step6" Action="verifyOption" LocType="id" LocSpecs="grant-type-selection-dropdown"
Data="ItemValue={TypeCode};CompareMode=Is" Description = "Select the {TypeCode} option in the grant type dropdown"/>
```

VerifyElementSize

verifyElementSize provides a verification mechanism of the size of some property of web elements. Typically, the element's getText() property is used but one can use any other valid property. As this function uses the findElement, its invocation is rather similar to findElement. Please, take note that the compareMode token is applied to the size of the property, not the property itself! Thus, the compareMode, if omitted, will default to "int" (integer) as this suffices to address any element's practical size.

This action makes use of the following attributes:

Action	verifyElementSize
LocType	Specifies the type of locator to use Example: xpath.
LocSpecs	Specifies the value of locator to use. Example: /html/body/div/div/h1
QryFunction	Specifies how to query the element For example: GetText
Data	Specifies additional information to be used during the verification logic. The following properties can add verb-specifi c semantics to this Action:

CompareMode	Specifies how to compare the element's value to the expected value to find a cell. Example: CompareMode=le. Example: CompareMode=between.
QueryParam	When QryFunction is GetAttribute, QueryParam is the attribute to use in the element query logic (element.GetAttribute({QueryParam}). When QryFunction is GetCssValue, QueryParam is the property to use in the element query logic (element.GetCssValue({QueryParam}).
Value	Expected size of the element's property. Example: Value=20 Example (for 'between'): 10.and.20
WaitTime	Used to override the default wait time – rarely used in this instance (same for WaitInterval)

Excel Example (new lines are used for each Data attribute for visual clarity)

The semantics of this example is "look for an element on the page using the locator specified here and obtain an actual value for comparison using the GetText function of this element. Compare the size of this value to the expected value provided in the Value= token of the Data attribute using the CompareMode.

ID	Action	LocType	LocSpecs	QryFunction	Data
Step6	verifyElementSize	xpath	/html/body/div/div/div[3]/div/div[3]/div/div[2]/div/	GetText	Value=20; CompareMode=le;

XML Example

```
<TestItem ID="Step6" Action="verifyElementSize" LocType="id" LocSpecs="page-header" QryFunction="GetText"
Data="Value=20;CompareMode=le" Description = "Verify the size of the page header is less than 21"/>
```

VerifyOption

verifyOption provides a generic selection mechanism for verifying an element within a list of <option/> elements. This verb makes use of the findOption verb so its invocation is quite similar. Please note that the query mechanism for this verb is determined by the Data token ItemText or ItemValue – therefore, these are omitted from the setup provided below.

This action makes use of the following attributes:

Action	selectOption
LocType	Specifies the type of locator to use Example: id.
LocSpecs	Specifies the value of locator to use. Example: grant-type-selection-dropdown **NOTE:** The locator type and specs should point to the <select> element containing the list of option elements to which the option element is anchored.
QryFunction	Specifies how to query an element for values other that the options value or text (here: what function to use on an element) For example: isDisplayed
Data	The following properties can add verb-specifi c semantics to this Action:

CompareMode	Specifies how to compare the element's value to the expected value to find a cell. Example: CompareMode=StartsWith.
ItemText	Specifies that the option should be searched by the item's text (not value). Mutually exclusive with ItemNo and ItemValue. Example: ItemText=Alaska.

	ItemValue	Specifies that the option should be searched by the item'svalue (not text). Mutually exclusive with ItemNo and ItemText. Example: ItemValue=AK.

Excel Example

The semantics of this example is "look for an option in a drop-down that starts at the locator indicated here, use the isDisplayed function of each child element found by using either the ItemText or ItemValue specified to find the desired option considering the CompareMode (and, if provided, Option) determine if the cell qualifies. In the illustration below, the step succeeds if an option with itemValue of {TypeCode} is found and its isDisplayed function returns true. In the example shown, the contents of a previously set variable named 'TypeCode' is the value by which to find an option using ItemValue.

ID	Action	LocType	LocSpecs	QryFunction	Data
Step6	verifyOption	id	grant-type-selection-dropdown	isDisplayed	ItemValue={TypeCode};CompareMode=Is;Value=true

XML Example

```
<TestItem ID="Step6" Action="verifyOption" LocType="id" LocSpecs="grant-type-selection-dropdown"
Data="ItemValue={TypeCode};CompareMode=Is" Description = "Select the {TypeCode} option in the grant type dropdown"/>
```

VerifyWebDriver

verifyWebDriver provides a generic mechanism for verifying a web page or properties of a web driver.

This action makes use of the following attributes:

Action	verifyWebDriver	
QryFunction	Specifies the function to use in order to get the actual value for comparison against the expected value. The following functions are supported: GetTitle, GetCurrentUrl, GetPageSource, GetWindowHandle.	
Data	The following properties can add verb-specifi c semantics to this Action:	
	CompareMode	Specifies how to compare the element's value to the expected value to find a cell. Example: CompareMode=StartsWith.
	Option	Specifies an option to be used when verifying the web driver. Example: Option=ignoreCase.
	Value	Specifies the expected value to compare the actual value to. Example: Value=Dashboard.

Excel Example

The semantics of this example is “using the GetTitle function of the web page displayed (as an expected value), determine whether the it contains text that is referenced by the variable {pageTitle. In the example shown, the contents of a previously set variable named ‘PageTitle’ is the expected value to consider.

ID	Action	QryFunction	Data
Step6	verifyWebDriver	GetTitle	Value={pageTitle;CompareMode=Contains

XML Example

```
<TestItem ID="Step6" Action="verifyWebDriver" Data="Value={pageTitle;CompareMode=Containss" Description = "Verify the page title is {pageTitle}"/>
```

VerifyWebElement

verifyWebElement provides a generic verification mechanism of elements. As this function uses the findElement, its invocation is rather similar to findElement.

This action makes use of the following attributes:

Action	verifyWebElement
LocType	Specifies the type of locator to use Example: xpath.
LocSpecs	Specifies the value of locator to use. Example: /html/body/div/div/h1
QryFunction	Specifies how to query the element For example: GetText
Data	Specifies additional information to be used during the verification logic. The following properties can add verb-specifi c semantics to this Action:

CompareMode	Specifies how to compare the element’s value to the expected value to find a cell. Example: CompareMode=StartsWith.
Option	Indicates other considerations to be used during validation. Example: ption=IgnoreCase
QueryParam	When QryFunction is GetAttribute, QueryParam is the attribute to use in the element query logic (element.GetAttribute({QueryParam}). When QryFunction is GetCssValue, QueryParam is the property to use in the element query logic (element.GetCssValue({QueryParam}).
Value	Expected value to use in order to locate cell. Example: Value=Johnson & Johnson

WaitTime Used to override the default wait time – rarely used in this instance (same for WaitInterval)

Excel Example (new lines are used for each Data attribute for visual clarity)

The semantics of this example is "look for an element on the page using the locator specified here and obtain an actual value for comparison using the GetText function of this element. Compare this to the expected value provided in the Value={header} of the Data attribute using the CompareMode and Option (in the case below 'ignore case'.) In the example shown, the contents of a previously set variable named 'Header' is the expected value to consider.

ID	Action	LocType	LocSpecs	QryFunction	Data
Step6	**verifyWebElement**	**xpath**	**/html/body/div/div/div[3]/div/div[3]/div/div[2]/div/**	**GetText**	**Value={header}; CompareMode=Contains; Option=IgnoreCase**

XML Example

```
<TestItem ID="Step6" Action="verifyWebElement" LocType="id" LocSpecs="page-header" QryFunction="GetText"
Data="Value={header};CompareMode=Contains" Description = "Verify page header contains {header} ignoring element's case."/>
```

Wait

wait is used to halt the web driver before the next test step.

This action makes use of the following attributes:

Action takeScreenShot

Excel Example

ID	Action	Data
Step6	wait	WaitTime=6

XML Example

```
<TestItem ID="Step6" Action="wait" Data="WaitTime=8"/>
```

APENDIX I – Creating a jDDT Script

Overview

The most time consuming aspect of working with the jDDT framework is selecting and using the correct search criteria for control-centered commands (e.g., 'Click', 'EnterText', etc.) Selecting and using the correct search criteria is 'where the rubber meets the road' in terms of test automation.

As the UI is typically a hierarchically organized structure composed of controls and their descendant controls, the path leading to the control of interest in a given test step must follow this hierarchical structure. This is done via the LocSpecs attribute (column) of a test step. The hierarchy of UI controls and "scaffolding" becomes more complex as the UI complexity increases. "Scaffolding" refers to invisible UI structures that are used for UI organizational purposes (e.g. 'div'.) Frequently, the control of interest is several levels down the hierarchy that may pose automation challenges if the developer(s) did not bother to associate the control of interest with any unique control property (such as "name" or "id").

The Selenium WebDriver automation framework, upon which jDDT is based, uses locators in order to find a web element on a page.

The jDDT framework uses the LocType and LocSpecs for locating an element.

A simple example is provided below with relevant annotations.

Determining LocType and LocSpecs (Element Locator)

In lieu of a better UI / application to handle the selection of controls and their search criteria, Firefox' plugin 'Firebug' and / or 'Firepath' are used. The following assumes that Firebug has been installed and activated and is used to explore the application.

Using FireBug

1. Using Firefox, navigate to the proper page in the Application.
2. If not done so yet, click the Firebug icon at the top right of the page (a panel will open at the bottom of the page.)
3. Hover over an element of interest and right-click.
   A drop-down menu with options will appear.
4. Select the 'Inspect element with Firebug' option at the bottom.
   The Firebug panel will expand to include the portion represented by the element you hovered over.
5. Inspect the element and decide what locator to use. This should populate the LocType column of a given test step.
   If you observe an id that seems like a human named it – use it. If the id is something that is machine generated (id="prw5EF50400CE32A996120DD7C000080100") it may be a dynamically generated id that will change from iteration to iteration and should not be used.
   Remember that you can always use the xPath locator type.
6. Determine the value for the locator. This should populate the LocSpecs column of a given test step.
   You can just highlight it within FireBug or, if you want to use xpath or css then (for xpath)

a. Right-click the highlighted element in the Firebug panel (a drop-down of options will display)
b. Select the 'Copy Xpath' option
c. In your worksheet, type xpath in this row's LocType and paste (Ctrl-V) the copied string from Firebug into the LocSpecs column of the worksheet.

The locator is now defined.

ID	Action	LocType	LocSpecs	Data
Con03	findElement	xpath	/html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/	

XML Version

```
<TestItem ID="CON03" Action="findElement" LocType="xpath" LocSpecs="/html/body/div/div/div[3]/div/div[3]/div/div[2]/div/div/"/>
```

APENDIX II – External Settings – ddt.properties File

Overview

jDDT uses a number of settings are used to specify default behavior. These are found in a properties file named ddt.properties that resides in the project's 'Resources' file as per Maven traditional project settings.

For a project that resides in c:\jddt – this is where the pom.xml file is found), the resources folder is c:\jddt\src\main\resources.

Following is a description of the various settings available at this time.

Sample Content

Following is a sample content of a project (lines starting with exclamation marks are comments.) Lines beginning with exclamation mark (!) are comments.

```
! Default Properties for the DDT Automation project - this is the place to override those.
! %proj% will be replaced by the value of project folder which is derived from %user.dir%
DDTVersion = 1.2.3a
ResourcesFolder = %proj%src\\main\\Resources
ImagesFolder = %proj%Images
ReportsFolder = %proj%Reports
DataFolder = %proj%Data
ScriptsFolder = %proj%Scripts
! Folder where Inline Test Items Strings generating classes should be loaded from (if any)
ClassLoadFolder = c:\\javaDDTExt\\target\\test-classes
XslFileName = Automation.Xsl
ItemDelim =
ValidDelims = ;~|!@#$%^&*()_+
! The file that starts execution of test items - Valid input files: .xls, .xlsx, .xml
InputSpecs = File!CrownRoot.xlsx!Root
! Comma delimited names of Inline Test Strings providing Classes
InlineTestStringsProviders = DemoTestStringsGenerator
! The IE standalone executable
IEDriverFileName = %proj%src\\main\\Resources\\IEDriverServer.exe
IEPropertyKey = webdriver.ie.driver
ChromeDriverFileName = %proj%src\\main\\Resources\\ChromeDriver.exe
ChromePropertyKey = webdriver.chrome.driver
! possible values: FIREFOX, IE, CHROME, OPERA, HEADLESS
BrowserName = FIREFOX
! For use by the HEADLESS driver (or other drivers) - List of settable capabilities
DesiredCapabilityNames =
takesScreenshot,javascriptEnabled,databaseEnabled,locationContextEnabled,applicationCacheEnabled,applicationCacheEnabled,webStorageEnabled,acceptSsl
Certs,rotatable,nativeEvents,proxy,unexpectedAlertBehaviour,elementScrollBehavior
DesiredCapabilityValues = true,true,false,true,true,true,false,false,false,true,false,dismiss,0
```

```
! in seconds - used by the 'find engine' as default and (optionally) modified as needed on the fly in test items dealing with ajax pages or pages
slow to load.
WaitTime = 10
! in millis - the interval of polling for existence of elements on a page.
WaitInterval = 100
! Adjust the timezone of the test machine to that of the application's / server
TimeZoneAdjustment = 5
! For reporting purposes
ProjectName = Selenium Based DDT Automation
! The email provider is gmail - these values were working initially - change user name, password and recipients to cater to your needs
EmailAuthenticationRequired = true
EmailSender = retsettdd@gmail.com
EmailPassword = Kishkes01
! Comma delimited list of results email recipients
EmailRecipients = bmelamed@microedge.com
EmailHost = smtp.gmail.com
EmailPort = 587
! Is this installation running locally
IsLocal = true
! "Always" = Take Image on each UI verb, "Never" = Do not take image (except during TakeScreenShot verb), "OnFail" = Only on failed UI Verbs (and
TakeImagePolicy = OnFail
! When searching for cells through table - should each individual cell be reported?  If true - report results may be huge in large tables - use true
mostly for debugging purposes.
ReportEachTableCell = false
! A list of events to report for each test item
EventsToReport = INFO,PASS,FAIL,SKIP
! Report Verbosity Support
! A list of status values to include in report - only items whose status is one of those listed below are included in the report.
! The entire list (for now) is: PASS,FAIL,SKIP
StatusToReport = PASS,FAIL,SKIP
! A list of actions to exclude from reporting
DontReportActions = NewTest,GenerateReport,InitializeReport,SetVars
! A list of test step (TestItem instance) properties to include in test step report - used to control report verbosity on the test step level
! The list in its entirety is:
! status,action,locType,qryFunction,description,active,data,comments,errors,exceptionStack,exceptionCause,duration,events
! To be reported, property should be listed here and in TestItemReportTemplate
ReportElements = id,status,action,loctype,locspecs,qryfunction,description,active,data,comments,errors,exceptionStack,exceptionCause,duration,events
! Reporting template for test item (TestItem instance) - tokens are substituted by the corresponding contents (unused tokens removed)
! The first and last characters in the string are used as the open and close token delimiters.
! {id}{description}{action}{loctype}{locspecs}{comments}
TestItemReportTemplate = {id}{description}{action}{data}{comments}{errors}{loctype}{locspecs}
! User defined default comparison (Equals, Is, Contains, Matches, StartsWith, EndsWith, etc. - used when not specified on test item.
DefaultComparison = Equals
! Number of milliseconds to pause before a UI step - used to slow the TestItemRunner if needed
DefaultPause = 100
! The default date format (MM/dd/yyyy) - use to parse / compare a date or date output when the default is padded day and month (01/03/2014)
! Use M/d/yyyy when the default is un-padded day and month (1/1/2014)
DateFormat = MM/dd/yyyy
! The default time stamp format - use to set date or date output
TimeStampFormat = MM/dd/yyyy HH:mm:ss
! Indicates whether to tab out of data entry - this is needed on some web pages to trigger javascripts tied to leaving input fields
```

```
TabOut = true
! This might be a more desired method of comparing strings as one does not have to worry about white space that may be invisible.
StripWhiteSpace = true
! Indicate reporting style.
! Default is the original style (xml that gets transformed to html), Html produces an active web page the user can interact with
ReportingStyle = Default
```

Settings Explained

DDTVersion	Used for reporting purposes and overrides the mechanism for getting the version from the pom.properties file when it is not accessible.
ResourcesFolder	Points to the project's resources folder. The token %proj% will be replaced by the location of the project on the file system.
ImageFolder	Points to the project's folder where images (screen shots) reside. The token %proj% will be replaced by the location of the project on the file system.
ReportsFolder	Points to the project's folder where reports files reside. The token %proj% will be replaced by the location of the project on the file system.
DataFolder	Points to the project's folder where data input files reside. The token %proj% will be replaced by the location of the project on the file system.
ScriptsFolder	Points to the project's folder where script files (shell commands or batch files) reside. The token %proj% will be replaced by the location of the project on the file system.
ClassLoadFolder	A folder from which external Test Strings provider classes may be loaded
InlineStringsProviders	A comma delimited list of classes that can be loaded dynamically during the test session.
XslFileName	This is the name of the reporter's xsl file (the file that defines the transformation from xml report to the html report.)
ItemDelim	A property delimiter - obsolete
ValidDelims	A list of valid delimiters to be used when the (default) ';' (semicolon) delimiter cannot be used in a property (it is part of one or more properties) For example: Suppose the text for verification of some element contains ';', the Data property that would default to: Value=some text used for validation including the ';' character;CompareMode=Contains;Option=IgnoreCase Cannot be used (as it is included in the Value item. In this case, use one of the validDelims: Value=some text used for validation including the ';' character@CompareMode=Contains@Option=IgnoreCase

InputSpecs	This is a "!" delimited string indicating the type of test items provider, the container (file or inline items provider class name) and the section (spreadsheet or inline items provider method name.)
IEDriverFileName	This is the standalone executable to start the IE (Internet Explorer) browser when the default browser is IE. It is typically located at the project's Resources folder.
IEPropertyKey	This is the string that identifies IE in the java VM's system properties structure. Needed when testing under IE.
ChromeDriverFileName	This is the standalone executable to start the Google Chrome browser when it is the default browser. It is typically located at the project's Resources folder.
ChromePropertyKey	This is the string that identifies Chrome in the java VM's system properties structure. Needed when testing under Chrome.
BrowserName	The name of the browser to start (FIREFOX, CHROME, ID, OPERA)
DesiredCapabilityNames	Used to specify capabilities of HEADLESS web driver. This comma delimited string specifies capability names.
DesiredCapabilityValues	Used to specify capabilities of HEADLESS web driver. This comma delimited string specifies capability values.
WaitTime	This is the number of seconds the web driver should wait for an element to render. Works in conjuction with WaitInterval. You do not get penalized for placing a large number here, it only makes it safer when handling slow rendering or ajax pages.
WaitInterval	This is the number of milliseconds the web driver will wait between verifications of the rendering of an element. Works in conjunction with WaitTime.
TimeZoneAdjustment = 5	Adjust the timezone of the test machine to that of the application's / server.
ProjectName	Used as a defult string during report generation – this will be the name of the project for which results are reported.
EmailAuthenticationRequired	This determines whether the email server should authenticate the sender before sending an email (use 'true' or 'false') Typically, gmail requires authentication while a company email server may not.
EmailServer	This identifies the mail server (used for mailing of results.) Example: 10.10.1.2
EmailRecipients	This is a comma delimited list of recipients that will get an email message with attached test results in html format.
EmailSender	This is the email address of the sender of the email messages (the machine where the tests are running should have a mailbox for this user.
EmailPassword	This is the (unencrypted!) password of the email sender (needed when authentication is required.)
EmailHost	This is the email server (host) name (smtp.gmail.com or 10.10.1.2, etc.)

EmailPort	This is the port used by the email server (host) name (587 is default for smtp.gmail.com or 25 for outlook.)
IsLocal	Indicates whether the tester is running local (used when taking screen shots as the mechanism for taking screen shots is different for local vs. server based testing.)
TakeImagePolicy	This indicates whether to create a screen shots when a UI step failed ("OnFail" / slower processing), after any UI step ("Always")or never ("Never".)
ReportEachTableCell	Indicates whether or not to report on each table cell during a table traversal. If true, reporting during searches of cells of large tables may be quite voluminous. Typically 'false', use 'true' for debugging purposes.
EventsToReport	Indicates which events to report for an individual step execution (any of INFO, PASS, FAIL, SKIP). Multiple INFO events may apply for any step but only one of PASS, FAIL, or SKIP may apply to any given step.
StatusToReport	Indicates which step statuses to include in the results report. A test step may have status of FAIL, PASS or SKIP – some of which may not be desired for a given test session. For example, you may want to skip the SKIP steps.
DontReportActions	A comma delimited names of Action instances to exclude from the report (as they have no testing semantics, only meaning to the test driver.) For example, newTest, wait, generateReport, etc. may have no value in terms of testing an application and would reduce the report size if excluded from it.
ReportElements	A comma delimited names of test item (step) properties that can be reported on. This is one of the mechanism used to control report contents (verbosity). Elements from this list may be used in the report template.
TestItemReportTemplate	A list of tokens indicating the order of elements to be reported in the test session result. Each of the tokens is denoted by it being surrounded with curly braces {id} stands for the test item's Id property {description} – the test item's description property, etc. The placement (or absence) of such tokens in this element define the structure of a reported test item in the test results. For debugging purposes you may want to have more items while for regular reports you may want to have only a few elements.
DefaultComparison	This is the default string comparison to be used. If (on a validation step) the "CompareMode=" element is missing, the comparison specified here (Equals, Contains, etc.) will be used.
DefaultPause	This (integer milliseconds) value is used to slow down the test driver – if needed. Larger values may be useful during debugging and harness testing runs to ensure the test harness runs as expected.
DateFormat	This is the format the application under test defaults its date display to. Use MM/dd/yyyy in the US to indicate day an month format is padded with 0 (01/09/2014) as opposed to unpadded format M/d/yyyy (1/9/2014).
TimeStampFormat	This is the format the application under test defaults its time stamp display to. In the US, this would typically be: MM/dd/yyyy HH:mm:ss.
TabOut	This indicates to the test driver whether it should tab out of controls after entering data in. The default is true. This may be needed for the application to trigger java script logic such as field validations, etc. Need to consult developers prior to setting this. This can be overridden on a test step level.

StripWhiteSpace		This setting enables one to compare values without concerns of white spaces. When the value of this setting is 'true', verifications will be made only between non-blank characters of the expected and actual values.
ReportingStyle		Used to specify the default reporting style. Used to provide for more than one reporting style.
	Default	**This results in an xml and a corresponding html file that is (ptionally) emailed to mail recipient(s)**
	Extent	**This results in an html file that is (ptionally) emailed to mail recipient(s). The html file is generated by the ExtentReport project (**Here is the link**).**

APENDIX III – Running Tests

Overview

There are several ways to run the jDDT tests, namely, from the command line, using a batch file and as an automated test using Java Development IDE (Intellij or Eclipse).

Running Tests From The Command Line

1. Test Mode.
   Navigate to the folder where the project (pom.xml) is located
   Type
   `mvn test`

If the commands were typed correctly, he tests will run and results will be emailed to the recipients list indicated in the TestRunnerSettings.xml file.

Running Tests Using a Batch File

This option is the most likely to be used as it avoids typing at the command line and enables easy scheduling of a test task.

Create a batch file that:

1. Changes directory to the project folder (for example: 'cd \javaddt' – without the quoes)
2. Issue the 'mvn test' commend (without the quotes.)

For a project that resides on C:\jddt this will be:

```
cd \javaddt
Mvn test
```

Running Tests as an Automation Test Developer

The developer runs tests by using Java Development IDE (Intellij or Eclipse), with the project opened and the class to be run highlighted.

A green ('run') button is available to start running the test harness using the default input file specified in the ddt.properties file that resides in the project's resources folder.

APENDIX IV – jDDT & Reusability

Reusability is, potentially, a most significant contributor to any automated test tool and one of the main themes in jDDT design.

The jDDT project has the following mechanisms to increase reusability:

1. The newTest action.
   The newTest action can be used to define a set of steps that can be reused, typically, with use of some variables.
   For example, if a web application's UI is organized as tabs and subtabs under the tabs, most likely the application have been also programmed with reusability built in (DOM structure of one tab / sub tab instance is the same as others.) This, in conjunction with variable setting one set of test items to navigate to any tab / sub-tab combination.
2. The setVars action.
   The setVars action is used to define test-session variable that can be used repeatedly following its definition.
   For example (omitted irrelevant columns) let's use a login script as a reusable group of tests with three variables (URL, User, Password.)
   Step POC 03 on some worksheet set variable as follows:

ID	Action	Data	Description
POC03	setVars	URL=https://demo-site.com/;User=Joe;password=Demo123	Set up the login credentials
POC04	newTest	FileName=DemoRoot.xlsx;WorksheetName=Login	Login with credentials of User={user}, Password={password}

Enabling the construction of a login script like the following (again, omitting irrelevant steps for brevity) – using a worksheet name Login in workbook named DemoRoot:

ID	Action	LocType	LocSpecs	QryFunction	Data
Log01	createWebDriver				URL={url};ptp=QSOF
Log02	verifyWebDriver			getTitle	Value=Company Portal;CompareMode=Contains;ptp=QSOF
Log03	sendKeys	Id	edit-name		Value={User}
Log04	sendKeys	Id	edit-pass		Value={Password}
Log05	click	id	edit-submit		
Log06	verifyWebDriver			GetTitle	Value=Dashboard;CompareMode=Equals;WaitTime=10;ptp=QSOF
Log07	saveElementProperty	ClassName	welcome_user	GetText	SaveAs=WelcomeUser

Log01	Creates a web driver that opens up the page provided by the URL variable set in POC03 and denoted as {url}. The createWebDriver verb looks for the URL token for this purpose. Note the ptp (Post Test Policy) of Quit Session On Fail – no reason to continue session if web site is not available.
Log02	verifyWebDriver – this verifies the welcome page to the site. Again, note the ptp – no use to continue the session if the page is not loaded.
Log03	Enter the value of {user} – in this case it will be 'joe' in the user name edit box located by id of edit_name.
Log04	Enter the value of {password} – in this case it will be 'Demo123' in the user password edit box located by id of edit_pass.
Log05	Click the submit button – no variables needed nor used here.
Log06	Verify that the Dashboard page is loaded allowing 10 seconds for the page to load, also with a Quit Session Post Test Policy.
Log07	Use the save ElementProperty action to save a string with the variable name of WelcomUser that can be referenced in subsequent verification steps as {WelcomeUser}. This form of variable setting is most useful in creating reusable steps that avoid hard coding of test steps.

APENDIX V – Verifications – Further Details

Verification of Non-String Values

By default, jDDt assumes verifications are String verifications. In this category the special strings 'true' and 'false' connote Boolean values.

Since all web pages are represented as a DOM which is constructed of strings, this suffices in most cases.

However, in some cases, verification of numeric string (including amounts / currency) or strings representing date values are useful especially when the essence of the verification involves collating order of non-string objects. This is significant when the collating order of the implied object is different from the collating order of the string representation of that object (9 is less than 1000 but its collating order is greater. Similarly, 01/01/2010 falls after 09/09/2009 but its collating order suggests the opposite.)

In such cases, jDDT provides for these instances via the inclusion of the "class" token in the Data property of a TestItem instance when the test item is related toverification.

For example:

Suppose some web page has two elements rendered, one as an amount (located by id=Some-Amt) and one as a date (located by id=Some-Date) and for both elements the query function of GetText returns the element's text which, for the respective elements is: "-12345.0" and "02/02/2010".

Steps Test05 and Test06 below are used to verify that the decimal value of -12345.0 is greater than -12345.0001 and the date represented by "02/02/2010" falls after the date represented by "**03/03/2000**". In both cases, what 'does the trick' is the specification of the respected class (Decimal and Date respectively.) Also, not the use of Option to specify the date format that should be used to convert the date to obtain a date value.

In the row with Id of Test07, the 'between' comparison is demonstrated where the contents of the located element is expected to be a decimal between 2,000 and 3,000

Id	Action	LocType	LocSpecs	QryFunction	Data
Test05	**verifyWebElement**	**id**	**Some-Amt**	**GetText**	**Value=-12,345.0001;Class=Decimal;CompareMode=gt**
Test06	**verifyWebElement**	**id**	**Some-Date**	**GetText**	**Value=03/03/2000;Class=Date;CompareMode=after;Option=MM/dd/yyyy**
Test07	**verifyWebElement**	**id**	**Some-Amt**	**GetText**	**Value=2,000.and.3,000;Class=Decimal;CompareMode=between**

Verification of Date-Dependent Elements

Most applications have some date related business rules that render elements relateive to the continuously changing system date.

For example, an order entry application defaults the Order_Date element to today's value, or, following creation of an appointment, a follow-up event is automatically scheduled for two weeks hence . These simple business rules are tricky to test for with a harness that is supposed to work unchanged as the value of "today's date" or "two weeks hence" changes daily.

Another example, suppose the same order entry application has a customer page that shows the monthly total for a client with the title of {MonthName}, {Year} total for {CustomerName} – more specifically, in Januray, 2014 the page of the ABC Company would display "**January, 2014 total for ABC Company**". Obviously, every month the page title will change and should be verified by the same test.

iDDT avails a facility to handle such cases in a very flexible manner by enabling the user to set a host of variables related to a given "reference-time-stamp".

Upon start, jDDT sets up the (reserved) system variables described below with respect to the system date. This enables verification of values related to the current date and time. Additionally, the Action "setVars" has a special case enabling one to change the default reference time stamp (forward or backwards.)

For example, suppose the application defaults a date field to two weeks from "today's date" (for example, follow-up date). In that case, one uses the setVars verbs as follows (unused columns omitted)

Id	Action	Data
Test07	**setVars**	**datevars=%Date+14Days%**

jDDT uses the format of %Date{sign}{number}{unit}% to indicate the number of units to move the reference time stamp forward or backwards.

The string must begin with "%date" (case insensitive) and must terminate with "%".

In between, the {sign} can be either + or -, followed by {units} which can be minutes, hours, days, weeks, months, years (case insensitive). Using %data% without any other specification, resets the date variables to represent the current date and time.

As a result of this, the variables map of jDDT contains the values summarized in the following table (these variable names should be reserved for that purpose).

The values are locale dependent and represent the US Eastern time zone.

Please note that the hours, minutes, seconds represent some fictitious time during which the command was, supposedly, issued.

Variable Name	On March25, 2014	After issuing dtvars=%Date+8Days%
$defaultDate	**03/25/2014**	04/02/2014'
$shortDate	**3/25/14**	'4/2/14'
$mediumDate	**Mar 25, 2014**	Apr 2, 2014'
$longDate	**March 25, 2014**	April 2, 2014
$fullDate	**Tuesday, March 25, 2014**	Wednesday, April 2, 2014
$hours	**03**	03
$minutes	**10**	10

Variable Name	On March25, 2014	After issuing dtvars=%Date+8Days%
$seconds	**36**	36
$milliseconds	**880**	880
$hours24	**15**	15
$timeStamp	**03:10:36**	03:10:36
$shortYear	**14**	14
$longYear	**2014**	2014
$shortMonth	**3**	4
$paddedMonth	**03**	04
$shortMonthName	**Mar**	Apr
$longMonthName	**March**	April
$weekYear	**2014**	2014
$shortDay	**25**	2
$paddedDay	**25**	02
$yearDay	**84**	92
$shortDayName	**Tue**	Wed
$longDayName	**Tuesday**	Wednesday
$ampm	**PM**	PM
$era	**AD**	AD
$zone	**EDT**	EDT

Considering the scenario mentioned above, verification of the page titled something like "**{monthName}, {year} total for {companyName}**" can be verified by specifying the string of **Value={$longMonthName}, {$longYear} total for {companyName};CompareMode=Contains;Option=ignoreCase**. This assumes that the value of {companyName} variable has been assigned previously.

APENDIX VI – Extending jDDT

jDDT can be extended by 'java speaking' programmers following the protocol described below.

There are two way, at present, to extend jDDT.

New Verb

This is accomplished by creating a new Verb subclass.

To extend jDDT with a new verb, do the following:

Using any IDE or text editor create:

1. A corresponding class in the Verb.java file.

For example… At the time of this writing, the 'Drag and Drop' functionality has not been catered to yet.

To implement Drag and Drop functionality:

a. Create a method (say) dragAndDrop in the Vocabulary class. This method, sets up the corresponding Verb and invokes it.

```
/**
 * Drag & drop logic using the context set in testItem and the standard basicDoIt() logic.
 */
public static void dragAndDrop(TestItem testItem) throws Exception{

   try {
      Verb.DragAndDrop verb;
      verb = (Verb. DragAndDrop) setupFromTestItem(testItem, new Verb. DragAndDrop ());
      Vocabulary.basicDoIt(testItem, verb);
   }
   catch (Exception e) {
      if (!testItem.hasException())
         testItem.setException(e);
      return;
   }
}
```

b. Create a Verb and override the doIt() method (that, as needed, may use existing element finding, interrogation and activation logic).
When doing so, consider:

c. Consider existing protocol.
The typical protocol is to carry whatever process under try/catch block with the following in mind:

   i. Errors in input parameters (or other errors that are not due to exception) are posted to the verb instance using Verb's basicAddError() method.
   ii. Exceptions caught in processing are posted to the testItem using Verb's setException() method.
   iii. Successful processing and / or comments that could be helpful in the reporting phase are posted to the testItem using Verb's basicAddComment() method.

iv. Explore existing methods in the Vocabulary class (and corresponding Verb subclass) and how they make use of / get support from existing jDDT classes.
v. Having built and compiled the project to ensure the code compiles, test it by creating a test item using the new function name (dragAndDrop) in the Action attribute of some test step.
vi. With respect to parameterization of your action, follow these guidelines:
vii. If a token that already exists can be used by your logic, use it (do not invent a new one) – say WaitTime or CompareMode, etc.
viii. If a token that does not exist yet is needed (say, dragPixelsH – a numeric indicating how many pixels to drag an element horizontally, dragPixelsV – a numeric indicating how many pixels to drag an element vertically, etc. – feel free to role your own and add a meaningful comment)
ix. Follow existing methodology inasmuch as possible to facilitate factoring, maintenance, etc.

2. A new line in the Verb.initializeVerbs() method as follows:

```
verbs.put("DragAndDrop".toLowerCase(), new DragAndDrop());
```

Code example (incomplete) of a doIt() method for the DragAndDrop verb:

```
public static class DragAndDrop extends Verb {

   public void doIt() throws VerbException {
      debug(this);

   basicValidation(this, true);
   if (this.hasErrors())
      return;

    // Validations if needed – typically, analyzing parameter(s) provided in the verb's getContext() hashtable.
    // (say, how much to drag, where to drop, etc.)
    // Find element and recovery –
    // Inasmuch as possible, use the current methodology for finding an element
    // Use FindElementto find the element of interest
    // If the element was found it can be accessed using the verb's getElement() instance.
   try {

      if (!(getElement() instanceof WebElement))
         FindElement.findElement(this);

      if (hasErrors())
          return;

      if ((getElement() instanceof WebElement)) {
         if (getElement().isEnabled()) {
                 // code whatever is needed to get the drag and drop effect
```

```
                    // Use addComment to add a comment about the success – this will appear in the report (a settings option)
                    Verb.basicAddComment(this, "Element Dragged and Dropped!");
                } // If element enabled
                else
                    Verb.basicAddError(this, "Element not enabled - dragAndDrop() failed");

            } // if webElement located
            else
                Verb.basicAddError(this, "Failed to find Web Element - Element not dragAndDropped!");
            } // Try
            catch (Exception e) {
                addException(e);
            }
        } // dragAndDrop
```

The example above illustrates the 'standard' way for implementing a web UI verb. However, in the same manner, one can extend jDDT for any purpose such as (for example) restoring a database from a snapshot or backup file. In cases like that, the code will look differently as there will not be any considerations related to web driver or web element.

Still, the protocol of passing parameters via the delimited string(s) in testItem.dataProperties and the setting of comments and errors in the TestItem instance should work the same way and will dove-tail properly with the DDTTestRunner instance.

External Method

Another option for extending jDDT is to roll up the sleeves and code a method in the DDTExternal class.

Methods in the DDTExternal class are called by the Verb subclass RunExternalMethod.

The DDTExternal class methods have a particular protocol compatible with the Verb paradigm of jDDT.

Example 1: Trivial Pass

The code:

```
/**
 * Context is a Hashtable of <string, Object> associations with methods to get and set properties.
 * Adding a comment, without an error means something was done successfull!
 * This is a static method but an instance method will do as well
 * @param context
 */
public static void trivialPass(DDTTestContext context) {
   context.addComment("Hey, I passed!");
```

```
    return;
}
```

The entry in a spreadsheet (only relevant columns shown)

Id	Action	Data	Description
Demo#	RunExternalMethod	Class=DDTExternal;type=DDTTestContext;method=trivialPass	Try a (trivial) passing External method

Example 2: Trivial Fail

The code:

```
/**
 * Context is a Hashtable of <string, Object> associations with methods to get and set properties.
 * Adding an error (possible, an exception) means something failed!
 * This is a static method but an instance method will do as well
 * @param context
 */
public static void trivialFail(DDTTestContext context) {
    context.addError("Ooops, I failed!");
}
```

The entry in a spreadsheet (only relevant columns shown)

Id	Action	Data	Description
Demo#	RunExternalMethod	Class=DDTExternal;type=DDTTestContext;method=trivialFail	Try a (trivial) failing External method

APENDIX VII – Using jDDT in a Selenium Project

jDDT can be used in your own Selenium based project following the protocol described below.

To use jDDT, in your own project, do the following:

1. Using Maven, install the JavaDDT to your local repository
   From the command prompt, change directory to the JavaDDT project (where the pom file resides) and issue the following command:
   ```
   mvn clean compile package install –DskipTests = true
   ```
2. Use Maven to manage your own java project.
3. Add the appropriate JavaDDT dependency in that project to reference the JavaDDT project you just created.
   ```
   <dependency>
     <groupId>com.DynaBytes.Automation</groupId>
     <artifactId>JavaDDT</artifactId>
     <version>1.1</version>
   </dependency>
   ```
4. Create metod(s) that explore the various JavaDDT objects (Locator, Verifier, Query, Reporter)
   Following is an example:

```
public void demoLoginStyle2(DDTTestContext testContext) throws Exception {

DDTTestRunner runner = new DDTTestRunner();
DDTReporter reporter = new DDTReporter();
DDTTestContext testContext = new DDTTestContext();

DDTTestRunner.setElementsMap(new Hashtable<String, WebElement>());
DDTTestRunner.setVarsMap(new DDTTestContext());

String url ="https://urlToLoginPageOfCompany.org/";
int commentNo = 0;
int errorNo = 0;

   reporter.reset();
       addAnotherComment(testContext, "Starting Login Style 2 with FIREFOX");

       System.out.println("");
       System.out.println("====================================");
       System.out.println("=========== START Style 2 ===========");
       System.out.println("========= Using TestContext =========");
       System.out.println("====================================");

try {
      String browserName = "FIREFOX";
      Driver.BrowserName browserType = Driver.asBrowserName(browserName);
            Driver.set(browserType);

      DDTTestContext tc = new DDTTestContext();
```

```
tc.setProperty("stepid", "CRDemo01");
tc.setProperty("url", url);
tc.setProperty("browser", "FIREFOX");
tc.setProperty("compareMode", "Contains");
tc.setProperty("qryFunction", "getTitle");
Verb.CreateWebDriver verb0 = new Verb.CreateWebDriver();
verb0.setContext(tc);

verb0.doIt();

System.out.println(verb0.toString());
reporter.addDDTest(new DDTReportItem(verb0));

tc.clear();
tc.setProperty("stepid", "CRDemo02");
tc.setProperty("value", "Donor Portal");
tc.setProperty("compareMode", "Contains");
tc.setProperty("qryFunction", "getTitle");
Verb.EnsurePageLoaded verb = new Verb.EnsurePageLoaded();
verb.setContext(tc);

verb.doIt();
reporter.addDDTest(new DDTReportItem(verb));

System.out.println(verb.toString());

tc.clear();
tc.setProperty("stepid", "CRDemo03");
tc.setProperty("locType", "id");
tc.setProperty("locSpecs", "edit-pass");
tc.setProperty("value", "Demo2014");
Verb.TypeKeys verb1 = new Verb.TypeKeys();
verb1.setContext(tc);

verb1.doIt();
reporter.addDDTest(new DDTReportItem(verb1));

System.out.println(verb1.toString());

tc.clear();
verb1.clear();
tc.setProperty("stepid", "CRDemo04");
tc.setProperty("locType", "id");
tc.setProperty("locSpecs", "edit-name");
tc.setProperty("value", "Jackie_K");
verb1.setContext(tc);

verb1.doIt();
reporter.addDDTest(new DDTReportItem(verb1));

System.out.println(verb1.toString());
```

```
        tc.clear();
        tc.setProperty("stepid", "CRDemo05");
        tc.setProperty("locType", "id");
        tc.setProperty("locSpecs", "edit-submit");
        Verb.Click verb2 = new Verb.Click();
        verb2.setContext(tc);

        verb2.doIt();
        reporter.addDDTest(new DDTReportItem(verb2));

        System.out.println(verb2.toString());

        tc.clear();
        tc.setProperty("stepid", "CRDemo06");
        tc.setProperty("qryFunction", "GetTitle");
        tc.setProperty("Value", "Dashboard");
        tc.setProperty("CompareMode", "=");
        tc.setProperty("WaitTime", "10");
        Verb.VerifyWebDriver verb3 = new Verb.VerifyWebDriver();
        verb3.setContext(tc);

        verb3.doIt();
        reporter.addDDTest(new DDTReportItem(verb3));

        System.out.println(verb3.toString());

        tc.clear();
        tc.setProperty("stepid", "CRDemo07");
        tc.setProperty("locType", "ClassName");
        tc.setProperty("locSpecs", "welcome_user");
        tc.setProperty("qryFunction", "GetText");
        tc.setProperty("SaveAs", "WelcomeUser");
        tc.setProperty("CompareMode", "is");
        Verb.SaveElementProperty verb4 = new Verb.SaveElementProperty();
        verb4.setContext(tc);

        verb4.doIt();
        reporter.addDDTest(new DDTReportItem(verb4));

        System.out.println(verb4.toString());

        Util.takeScreenImage(Driver.getDriver(), "C:/testingddt", "somesillyFileName");
        reporter.addDDTest(new DDTReportItem("PASS", "Screen Shot Taken."));

        addAnotherComment(testContext, "Login Style 2 Passed");

        System.out.println("\nScreen Shot Taken in C:/testingddt, file named 'somesillyFileName'\n");
        Driver.quit();
        reporter.generateReport("Login Style 2 Report","Testing of Login Style 2" );
}
```

```
catch (Exception e) {
      addAnotherError(testContext, "Login Style 2 generated exception " + e.toString());

      testContext.setProperty("exception2", e);
      System.out.println("Exception generated in Login Style 2: " + e.toString());
}
finally {
      testContext.setProperty("comments02", "Login Style 2 PASS");
      System.out.println("\n========== DONE Login Style 2 ==========\n");
}
   }
```

APENDIX VIII – Using Inline Test Strings Provider Classes

Overview

This appendix describes how to use a DDT feature that enables the use of 'Inline' Test Strings Provider classes.

This option is appropriate for folks who prefer to maintain the source of test data using java classes instead of external spreadsheets, xml or html files.

'Inline' Test Strings Provider classes are Java classes that extend the DDT class InlineTestStringsProvider class that, in turn, extends the abstract class TestStringsProvider.

A possible advantage of using Inline Test String Provider classes is that does not have to maintain a separate test-ware objects (spreadsheets or other files) as well as enjoying the search and other facilities of some Java IDE.

The original project includes an example of such class (SampleTestStringGenerator) that can be used as a template and a model (to follow or to avoid.)

The design of this feature is guided by the following:

- Keep is simple.
- Enabling non-technical staff to code such classes and methods.
- Separating the Test Strings Provider classes from the main body of code.

In order to use this feature, a simple java project should be constructed which includes a small subset of the DDT main project.

The Setup

This describes an example project that might be used for this purpose.

The sample project is rooted at the C:\JavaDDTExt folder (but, of course, could be set up on any network or other folder)

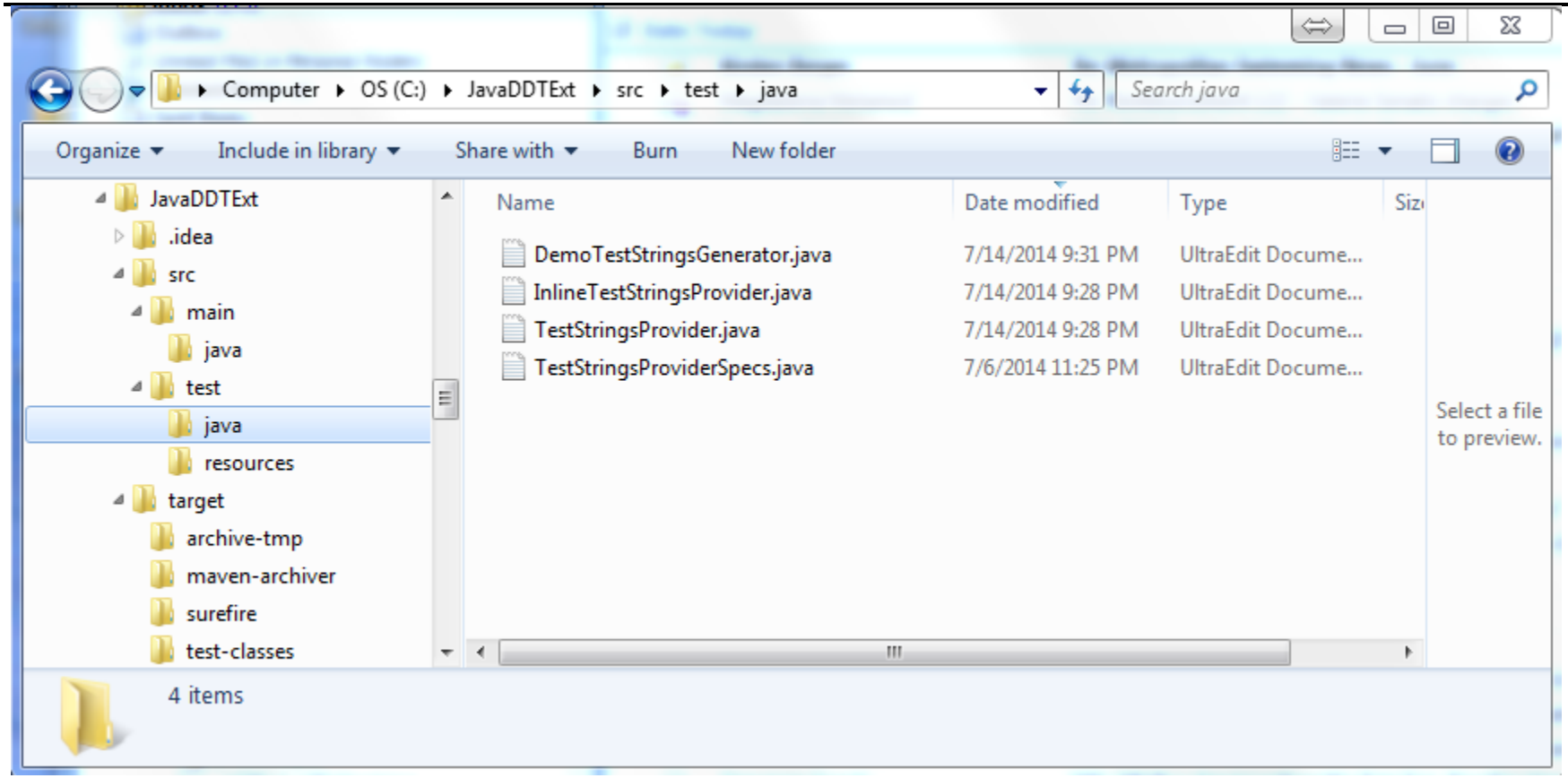
Folder Structure

This, too, is just an example which could be modified as per your desired standards.

The example described here is a project developed using IntelliJ Idea from JetBrains with Maven as the dependency manager.

Thus, one will notice the presence of the .idea folder under the project folder.

The project structure is shown below.



Project Contents

The important elements of the project are:

pom.xml

The pom.xml file for this project resides at the root of the project (C:\JavaDDTExt) and its content is:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>JavaDDTExt</groupId>
<artifactId>JavaDDTExt</artifactId>
<version>1.0</version>
<packaging>jar</packaging>

<dependencies>

    <dependency>
        <groupId>commons-lang</groupId>
        <artifactId>commons-lang</artifactId>
        <version>2.6</version>
    </dependency>

    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.1</version>
    </dependency>

</dependencies>

<build>
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.2</version>
        <!-- nothing here -->
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>2.2-beta-4</version>
        <configuration>
            <descriptorRefs>
                <descriptorRef>jar-with-dependencies</descriptorRef>
            </descriptorRefs>
            <archive>
                <manifest>
                    <mainClass>DDTestRunner</mainClass>
                    <addClasspath>true</addClasspath>
```

```
                    </manifest>
                </archive>
            </configuration>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
          </plugin>
        </plugins>
    </build>
```

Classes

DemoTestStringsGenerator	This is a sample class that can be used as a template. Each method provides test strings for some test case. The class extends the InlineTestStringsProvider class from which relevant methods are inherited.
InlineTestStringsProvider	This is the parent class providing the necessary machinery for Inline Test Strings generation. Some methods were removed from this class in order to avoid dependencies on other DDT classes to limit the size of the project. The InlineTestStringsProvider class is an extension of the abstract class TestStringsProvider.
TestStringsProvider	This is the root class of the DDT test strings provider.
TestStringsProviderSpecs	This class provides the machinery for specifying a Test Strings Provider from either, a delimited string or String[] array describing the type of provider (file or 'inline'), the source (worksheet or class name) and test item container (worksheet or method)

Example Test Strings Providing Method

Here is an example of a 'root' method of an Inline Test String Provider class (e.g. DemoTestStringsGenerator):

```
public void calculate(ArrayList<String[]> list) {
    // Template:
    //list.add(new String[] {"id", "action", "locType", "locSpecs", "qryFunction", "active", "data", "description"});
    list.add(new String[] {"Calculate01", "sendKeys", "Id", "number1", "", "", "Value={number1}", "Enter {number1} in the Number1 text box"});
    list.add(new String[] {"Calculate02", "sendKeys", "Id", "number2", "", "", "Value={number2}", "Enter {number2} in the Number2 text box"});
    list.add(new String[] {"Calculate03", "findElement", "Id", "function", "", "", "", "Find the {function} web element"});
    list.add(new String[] {"Calculate04", "click", "Css", "option[value='{Action}']", "", "", "", "Click the '{action}' option in the Action drop down list"});
    list.add(new String[] {"Calculate05", "click", "Id", "calculate", "", "", "", "Click the Calculate button"});
    list.add(new String[] {"Calculate06", "verifyWebElement", "Id", "answer", "{function}", "", "Value={Answer};compareMode={CompareMode}", "Find the {function}
    web element"});
    stringifyTestItems(list);
}
```

All Test Strings Provider methods are instance methods that add a String[8] (test item) to an ArrayList<String[]> object, one item at a time.

The last line of those methods (stringifyTestItems(list);) convert the ArrayList to String[][] array – the standard format of Test Strings used by the DDT driver.

Please note the comment template line at the start of a method – just for documentation purposes – indicating that the order of elements in the String[8] is crucial and must be adhered to.

Please note the occurrences of strings enclosed in squiggly brackets ({function} for example) – these connote usage of a previously set DDT variable so named.

The Target Folder

With such a project properly set, building the project (either by issuing an "`mvn clean compile package`" command or by the IDE, the Target folder of the project is populated with the class(es) we are after. In this case, the fully specified class path would be:

`C:\JavaDDTExt\target\test-classes\ DemoTestStringsGenerator.class`

Settings

Some settings are needed to be set properly in the ddt.properties file of the main JavaDDT project in order for this feature to work.

ClassLoadFolder A folder from which external Test Strings provider classes may be loaded.
In the case documented here, this entry would be (notice the escaping sequence '\\'):
`ClassLoadFolder = c:\\javaDDTExt\\target\\test-classes`

InlineStringsProviders A comma delimited list of classes that can be loaded dynamically during the test session.
In the case documented here (and assuming just one inline test strings provider class, this entry would be:
`InlineStringsProviders = DemoTestStringsGenerator`

If (and usually you would) you wish to have the root method (the one that the DDT Driver invokes first) in the same class, then the following entry should be set:

InputSpecs The specification of the provider type and specs that gets the test session going.
In the case documented here, and, assuming the initial method to get the test session is named 'root', this entry would be:
`InputSpecs = Inline!DemoTestStringsGenerator!root`

APENDIX IX – Workstation / Project Setup

jDDT was originally developed on a Windows workstation using **JetBrains IntelliJ Idea** with **Maven** as the dependency manager.

This section summarizes a typical setup by user named {user}.

1. Download and install the current stable Java SDK (use all defaults).
   The installation folder is referred to as {javaHome}.
2. Download and install the current version of **Maven** (use all defaults) into a folder of your choice on your workstation.
   This installation folder is referred to as the {m2home} below.
3. In the Control Panel > System and Security > System > Advanced System Settings, click the Environment Variables button and set the following:
   a. JAVA_HOME – set it to the value of {javaHome} above.
      Do this on the user and system level.
   b. M2_HOME – set it to the value of {m2home} above.
      Do this on the user and system level.
      NOTE: You should be logged in as the user who will actually work with JavaDDT on that machine as the {m2home} folder is (by default) tied to the user who installed Maven.
   c. Path – include %M2_HOME%\bin and %JAVA_HOME%\bin in the path.
      Do this on the user and system level.

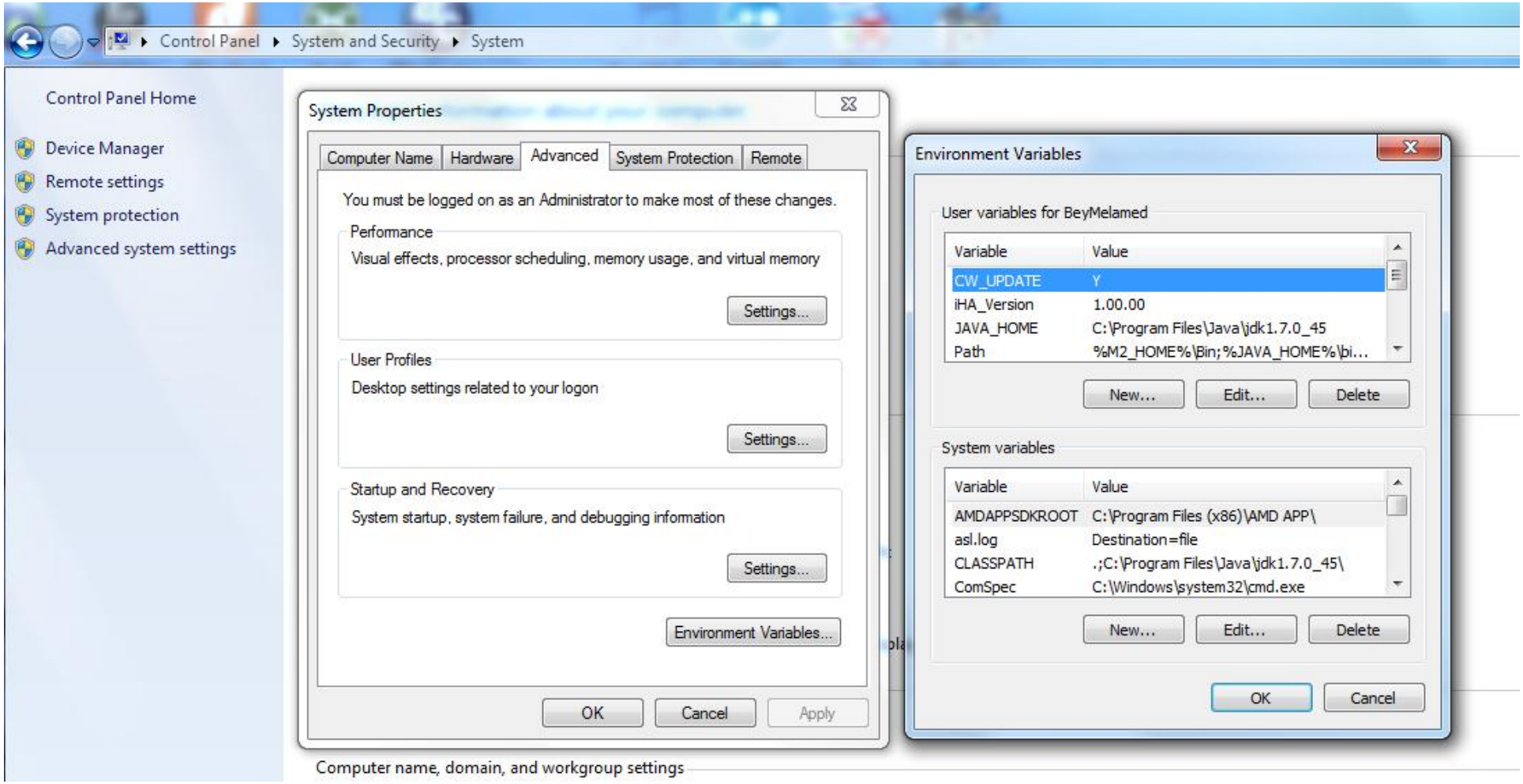
   See image below.
4. Copy the JavaDDT files to the C:\ drive.
   The software portion of the project structure (at the time of this writing) is:
   JavaDDT (this is where the pom.xml file resides)
   - Src
     - Main
       - Java
         - (empty folder)
     - Test
       - Java
         - JavaDDT Class files (*.java)
       - Resources
         - ddt.properties — JavaDDT settings
         - automation.xsl — The xslt transation file used to produce report by the current reporting mechanism.
         - Phantomjs.exe — Headless web driver server from Phantom.
         - IEDriverServer.exe — Internet Explorer web driver server.

ChromeDriver.ext	Google Chrome web driver server.
Selenium-server-standalone…	The Selenium server (… indicates the version you are using.)

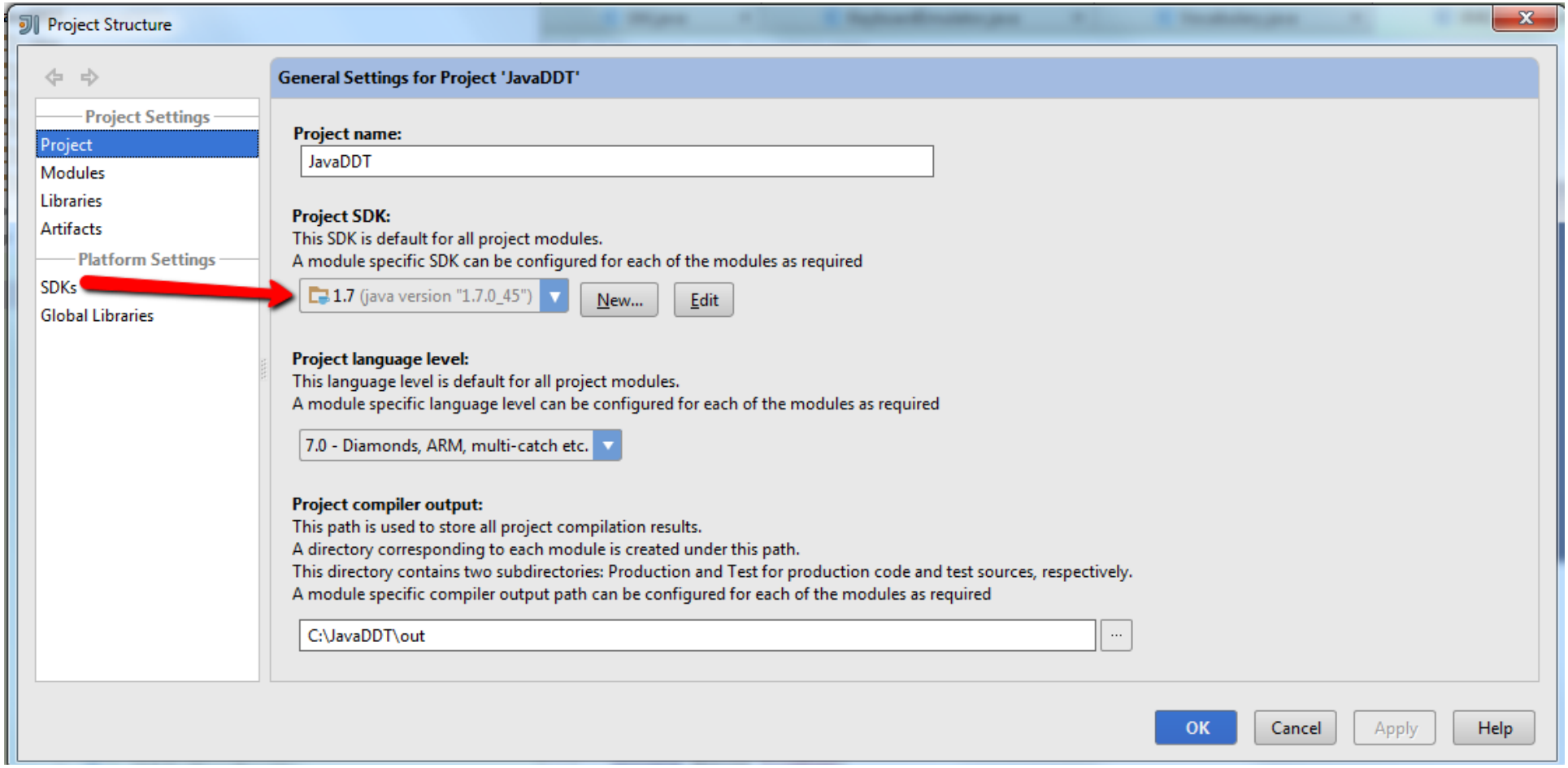
The default location of the input to the Test Runner is in the Data folder under the JavaDDT folder.
Any folders structure underneath the Data folder is totally up to the test designer but, for coherency reasons, one should at least contemplate the option of structuring the data folder and sub-folders somewhat similar to the application (web site) structure (navigation tabs, sub tabs, etc.)

5. If all went as planned, you should be able to to the following:
   a. Open a cmd window
   b. Change Directory to the C:\JavaDDT folder
   c. Issue the command "mvn clean compile"
   d. If all is OK mvn will load a bunch of stuff filling the command window with hundreds of commands and finally compiling JavaDDT finishing with SUCCESS message.
      If things did not go OK then the 'mvn' command will not be understood and you will have to scratch your head and figure out what went awry.
      Please note that Maven's actions may get timed out for no other reason than network traffic on your network or apache site so, if the initial command did not work, do try again.
6. Verify the various web server drivers listed above are in the resources folder.
7. Make sure that FireFox version 25 is used as (as of this writing) later versions do not work with Selenium WebDriver II.
   You should now be able to run tests by opening a cmd window, changing directory to the c:\JavaDDT folder and issueing the "mvn test" command.
   Of course, you can encode this in a batch file to save time.
8. Load and install the most recent IntelliJ Idea (use the free community version download)
9. Using Intellij Idea, open the JavaDDT project (by navigating to the c:\JavaDDT folder where Intellij will recognize it has a pom.xml file and will have a clickable icon for you to click on)
10. In Intellij Idea with JavaDDT opened, navigate to File > Project Structure and ensure that the SDKs has the path named JAVA_HOME selected as the SDK. See image below.
    You can now work as a developer with the software.

Setting workstation's System variables for JavaDDT.



Project Structure – Ensure correct reference to the Java SDK.



APENDIX X – CSS Selectors

Knowledge of CSS Selector is extremely beneficial for the QA Engineer responsible for creation of test input. This appendix contains a list of CSS selectors with brief examples to get the curious user get started.

Selector	Example	Selects
*	$("*")	All elements
#id	$("#lastname")	The element with id="lastname"
.class	$(".intro")	All elements with class="intro"
.class,.class	$(".intro,.demo")	All elements with the class "intro" or "demo"
element	$("p")	All <p> elements
el1,el2,el3	$("h1,div,p")	All <h1>, and <p> elements
:first	$("p:first")	The first <p> element
:last	$("p:last")	The last <p> element
:even	$("tr:even")	All even <tr> elements
:odd	$("tr:odd")	All odd <tr> elements
:first-child	$("p:first-child")	All <p> elements that are the first child of their parent
:first-of-type	$("p:first-of-type")	All <p> elements that are the first <p> element of their parent
:last-child	$("p:last-child")	All <p> elements that are the last child of their parent
:last-of-type	$("p:last-of-type")	All <p> elements that are the last <p> element of their parent
:nth-child(n)	$("p:nth-child(2)")	All <p> elements that are the 2nd child of their parent
:nth-last-child(n)	$("p:nth-last-child(2)")	All <p> elements that are the 2nd child of their parent, counting from the last child
:nth-of-type(n)	$("p:nth-of-type(2)")	All <p> elements that are the 2nd <p> element of their parent
:nth-last-of-type(n)	$("p:nth-last-of-type(2)")	All <p> elements that are the 2nd <p> element of their parent, counting from the last child

Selector	Example	Selects
:only-child	$("p:only-child")	All <p> elements that are the only child of their parent
:only-of-type	$("p:only-of-type")	All <p> elements that are the only child, of its type, of their parent
parent > child	$("div > p")	All <p> elements that are a direct child of a element
parent descendant	$("div p")	All <p> elements that are descendants of a element
element + next	$("div + p")	The <p> element that are next to each elements
element ~ siblings	$("div ~ p")	All <p> elements that are siblings of a element
:eq(*index*)	$("ul li:eq(3)")	The fourth element in a list (index starts at 0)
:gt(*no*)	$("ul li:gt(3)")	List elements with an index greater than 3
:lt(*no*)	$("ul li:lt(3)")	List elements with an index less than 3
:not(*selector*)	$("input:not(:empty)")	All input elements that are not empty
:header	$(":header")	All header elements <h1>, <h2> ...
:animated	$(":animated")	All animated elements
:focus	$(":focus")	The element that currently has focus
:contains(*text*)	$(":contains('Hello')")	All elements which contains the text "Hello"
:has(*selector*)	$("div:has(p)")	All elements that have a <p> element
:empty	$(":empty")	All elements that are empty
:parent	$(":parent")	All elements that are a parent of another element
:hidden	$("p:hidden")	All hidden <p> elements
:visible	$("table:visible")	All visible tables
:root	$(":root")	The document's root element
:lang(*language*)	$("p:lang(de)")	All <p> elements with a lang attribute value starting with "de"

Selector	Example	Selects
[*attribute*]	$("[href]")	All elements with a href attribute
[*attribute=value*]	$("[href='default.htm']")	All elements with a href attribute value equal to "default.htm"
[*attribute!=value*]	$("[href!='default.htm']")	All elements with a href attribute value not equal to "default.htm"
[*attribute$=value*]	$("[href$='.jpg']")	All elements with a href attribute value ending with ".jpg"
[*attribute|=value*]	$("[title|='Tomorrow']")	All elements with a title attribute value equal to 'Tomorrow', or starting with 'Tomorrow' followed by a hyphen
[*attribute^=value*]	$("[title^='Tom']")	All elements with a title attribute value starting with "Tom"
[*attribute~=value*]	$("[title~='hello']")	All elements with a title attribute value containing the specific word "hello"
[*attribute*=value*]	$("[title*='hello']")	All elements with a title attribute value containing the word "hello"
:input	$(":input")	All input elements
:text	$(":text")	All input elements with type="text"
:password	$(":password")	All input elements with type="password"
:radio	$(":radio")	All input elements with type="radio"
:checkbox	$(":checkbox")	All input elements with type="checkbox"
:submit	$(":submit")	All input elements with type="submit"
:reset	$(":reset")	All input elements with type="reset"
:button	$(":button")	All input elements with type="button"
:image	$(":image")	All input elements with type="image"
:file	$(":file")	All input elements with type="file"
:enabled	$(":enabled")	All enabled input elements
:disabled	$(":disabled")	All disabled input elements
:selected	$(":selected")	All selected input elements

Selector	Example	Selects
:checked	$(":checked")	All checked input elements